

**Adrien Coyette,  
David Faure,  
Juan González-Calleros,  
Jean Vanderdonckt (Eds.)**



# User Interface Extensible Markup Language

UsiXML'2011



Proceedings of the  
2<sup>nd</sup> International Workshop on  
User Interface eXtensible Markup Language  
September 6, 2011 - Lisbon, Portugal





# UsiXML 2011

Proceedings of the 2nd International Workshop on  
USer Interface eXtensible Markup Language

*Sponsored by:*

*UsiXML, ITEA2 project #08026*

*ITEA2 Program of the European Commission*

*Eureka Project #3674*

*Also sponsored by:*

*ACM's Special Interest Group on Computer-Human*

*Interaction (ACM SIGCHI), Belgium division (SIGCHI.BE)*

*Supported by:*

*Thales Research & Technology France, Paris (France)*

*Université catholique de Louvain, Louvain-la-Neuve (Belgium)*

## Foreword

UsiXML'2010, the 1<sup>st</sup> International Workshop on User Interface eXtensible Markup Language, was held in Berlin, Germany (June 20, 2010) during the 2<sup>nd</sup> ACM Symposium on Engineering Interactive Computing Systems EICS'2010 (Berlin, 21-23 June, 2010). This workshop is aimed at investigating open issues in research and development for user interface engineering based on User Interface eXtensible Markup Language (UsiXML), a XML-compliant User Interface Description Language and at reviewing existing solutions that address these issues. In particular, the “ $\mu$ 7” concept is explicitly addressed by discussing how and when each dimension can be supported: multi-device, multi-user, multi-linguality, multi-organisation, multi-context, multimodality, and multi-platform. Twenty-three papers have been accepted by the International Program Committee for this edition.

UsiXML'2011, the 2<sup>nd</sup> International Workshop on User Interface eXtensible Markup Language, was held in Lisbon, Portugal (September 6, 2011) during the 13<sup>th</sup> IFIP TC 13 International Conference on Human-Computer Interaction Interact'2011 (Lisbon, 5-9 September 2011). This second edition was dedicated to “Software Support for User Interface Description Language” (UIDL'2011) in order to investigate what kind of software we need to properly support the usage of any User Interface Description Language. Twenty-seven papers have been accepted out of thirty-seven submissions by the International Program Committee for this edition. Questions addressed by this workshop include the following ones, but not limited to:

What are the major challenges (e.g., conceptual, methodological, technical, organizational) for software support for a UIDL?

- For which kinds of systems or applications are UIDLs appropriate, efficient, and desirable?
- When and how could we measure the effectiveness, the efficiency of UIDLs and their software support?
- How could we measure the quality of the user interface resulting from a development method based on a UIDL and supported by some software?
- In which ways will UIDL affect HCI practice in the future and how will they evolve?
- What kinds of software support are particularly desirable for a UIDL and what kind of implementation reveal themselves to be the most appropriate, efficient for this purpose?

We also wish to thank our corporate sponsor Thales Research & Technology France and the support of our academic sponsor: Université catholique de Louvain.

**Adrien Coyette, David Faure, Juan Manuel Gonzalez Calleros, and Jean Vanderdonckt**

*Workshop General Co-Chairs*

Permission to make digital or hard copies of portions of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyright for components of this work owned by authors must be honored. Abstracting with credits permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permission to republish has to be obtained from Thales Research & Technology France, Paris (France). Request permission to republish a specific article has to be obtained from the authors.

Published by Thales Research and Technology France, September, 2011.

ISBN: 978-2-9536757-1-9



## Enhancing user interface design: Model-based approach ensures greater versatility in multiple contexts



The UsiXML project is developing an innovative model-driven language to simplify and improve user interface design for the benefit of both consumer and industrial end-users. It will provide particular benefits for industry in terms of productivity, usability and accessibility by supporting the ‘ $\mu$ 7’ concept of multiple device, user, culture/language, organisation, context, modality and platform applications.

While the European Union offers a huge market for European companies, the many different languages and cultures, multiple organisations and numerous contexts remain strong constraints to the wide distribution of nationally developed products.

This is a particular problem in software-based and software-intensive systems. Here the key challenge for human-system interaction is the design of simple and natural multimodal user interfaces – such as voice command, text to speech and gesture recognition – with enough features for users of different levels of expertise and capabilities.

### More Efficient Model-Based Approach

New ways are needed to design such user interfaces efficiently to cope with increased complexity and the evolution of operational use – including robustness to organisational changes, devices and modalities. To achieve this objective, a more efficient model-driven approach is needed.

A large proportion of today’s infrastructure tools, software tools and interactive applications are implemented on top of XML platforms. This ITEA 2 project therefore proposes to enhance the XML-based user interface extensible mark-up language (UsiXML) by adding versatile context-driven capabilities that will take it far beyond the state of the art and lead to its standardisation.

UsiXML will define, validate, and standardise an open user interface description language (UIDL), increasing productivity and reusability, and improving usability and accessibility of industrial interactive applications using the  $\mu$ 7 concept.

The seven ‘ $\mu$ 7’ dimensions offer a powerful new approach to cope with technological advances and environmental variations such as: new input/output devices; users evolving over time and new user profiles appearing constantly; applications submitted to internationalisation with new languages, markets and cultures; applications that need to be extended to multiple organisations; and new contexts and constraints imposed to use various modalities depending on context and platform. This is a major breakthrough as it will no longer be necessary to develop individual unique interface solutions for each application.

### Helping Address the Whole European Market

Development of a standard language and a universal engineering framework will provide benefits in terms of time-to-market, productivity, reuse, propagation-of-change and usability/accessibility guarantees. UsiXML will help industries address the European market as a whole, instead of remaining in local niche markets. The results will improve the competitiveness of European industries and enable the needs of European citizen to be better satisfied.

The market relevance also relates to the strong increase in demand for new types of user interface, driven by sectors such as the home, healthcare and mobility – and device heterogeneity. The complexity of user-interface design and the associated costs are increasing. Thus, a dramatic improvement in design efficiency is required, particularly to meet tough US competition.

### Adding Versatile Context-Driven Capabilities

The success of the ITEA 2 project will reduce total application costs and development time by enhancing the UsiXML interface modelling language through the addition of versatile context-driven capabilities.

UsiXML is an XML-compliant mark-up language that describes a user interface for multiple contexts such as character, graphical, auditory or multimodal interfaces. Thanks to UsiXML, non-developers can shape the user interface of any new interactive application by specifying it in UsiXML, without requiring the programming skills usually found in mark-up and programming languages.

This project offers a practical application of model-driven architecture (MDA) and engineering (MDE) that will show immediate benefits in day-to-day software engineering. The impact of UsiXML on European technological and commercial advancement will be mainly found in:

- Advancement of European state-of-the-art in modelling and model transformation techniques for human-computer interaction;

- Technological transfer from the academic partners to large and small industries;
- Stronger positioning in the standardisation bodies; and
- The very high performance/price ratio of the UsiXML solution as no hardware investment will be needed – giving the companies marketing it a strong edge.

### **Demonstrating Real Benefits in Use**

Innovations in UsiXML will help European software vendors and industrial systems makers to increase productivity in software development and reduce development costs. The three major outcomes will be:

1. A development methodology for multi-target user interfaces integrating the whole  $\mu 7$  concept;
2. A software environment for developers guaranteeing the quality and usability of the resulting user interfaces; and
3. A UsiXML language release under standardisation and XML compliant.

These results will reduce time to market, speed-up productivity, improve factorisation, speed change propagation and better assess usability and accessibility.

While one of the main goals is standardisation, companies need to be shown that there is a real benefit for them, in their domain and with the constraints they face in their everyday business. The ITEA 2 project will validate the UsiXML User Interface Description Language (UIDL) and framework in a wide range of applications offering a broad spectrum of characteristics. This will encourage the build-up of the momentum required for the adoption of UsiXML as a general-purpose user-interface definition language throughout Europe.

### **UsiXML Project technical data**

Project start: October 2009

Project end: September 2012

#### *Partners*

Thales Research & Technology, Université catholique de Louvain, Bilbomatica, Vector Software Company, PY Automation, Defimedia, See & Touch, Namahn, ProDevelop, Baum Engineering, Aérodrone, W4 S.A., SymbiaIT, Institut Télécom, Institutul National de Cercetare-Dezvoltare in Informatica, Université Joseph Fourier, Laboratoire d'Informatique de Paris 6, Université de Troyes, University of Namur, DAI-Labor, University of Rostock, Universidad Politecnica de Valencia, University of Castilla-La Mancha, University of Madeira, Agence Wallonne des Télécommunications.

#### *Countries*

Belgium, France, Germany, Portugal, Romania, Spain.

#### *Contacts*

**Project Leader:** David Faure, Thales Research & Technology, France

David.Faure@thalesgroup.com

**Scientific Coordinator:** Jean Vanderdonckt, Université catholique de Louvain, Belgium

Jean.Vanderdonckt@uclouvain.be

#### **Project website:**

UsiXML Language Web Site: <http://usixml.org>

UsiXML project web site: <http://www.usixml.eu>

#### *Support*

ITEA2 is the EUREKA program #3674 and supports UsiXML project.



# UsiXML 2011 Workshop Organization

**General Co-Chairs:** Adrien Coyette (*Université catholique de Louvain, Belgium*)  
David Faure (*Thales Research & Technology, France*)  
Juan Manuel Gonzalez Calleros (*Benemérita Universidad Autónoma de Puebla, Mexico*)  
Jean Vanderdonckt (*Université catholique de Louvain, Belgium*)

**Program Committee:** M. Abed (*Université de Valenciennes, France*)  
C. Agüero (*Thales Research & Technology, The Netherlands*)  
D. Akoumianakis (*Technological Education Institution of Crete, Greece*)  
M. Ph. Armbruster (*Technical University of Berlin, Germany*)  
M. Blumendorf (*DAI Labor, Germany*)  
B. Bomsdorf (*University of Fulda, Germany*)  
P. Bottoni (*University of Rome “La Sapienza”, Italy*)  
K. Breiner (*University of Kiel, Germany*)  
G. Calvary (*Université Joseph Fourier, France*)  
J. M. Cantera Fonseca (*Telefonica, Spain*)  
K. Coninx (*University of Hasselt, Belgium*)  
A. Coyette (*Université catholique de Louvain, Belgium*)  
J. Creissac Campos (*University of Minho, Portugal*)  
A.-M. Dery-Pinna (*Ecole Polytechnique Universitaire de Sophia-Antipolis, France*)  
S. Dupuy-Chessa (*Université Joseph Fourier, France*)  
D. Faure (*Thales Research & Technology, France*)  
P. Forbrig (*University of Rostock, Germany*)  
E. Furtado (*University of Fortaleza, Brazil*)  
J. Gallud (*University of Castilla-La Mancha, Spain*)  
J. Garcia-Molina (*University of Murcia, Spain*)  
J. M. González Calleros (*Benemérita Universidad Autónoma de Puebla, Mexico*)  
P. Gonzalez (*University of Castilla-La Mancha, Spain*)  
G. Gronier (*Centre de Recherche Public Henri Tudor, Luxembourg*)  
J. Guerrero Garcia (*Benemérita Universidad Autónoma de Puebla, Mexico*)  
H. Kaindl (*University of Vienna, Austria*)  
S. Kanai (*University of Hokudai, Japan*)  
Ch. Kolski (*Université de Valenciennes, France*)  
J. Leite (*Campus Universitário – Lagoa Nova, Brazil*)  
V. M. Lopez Jaquero (*University of Castilla-La Mancha, Spain*)  
M.D. Lozano (*University of Castilla-La Mancha, Spain*)  
K. Luyten (*University of Hasselt, Belgium*)  
I. Marin (*Fundacion CTIC, Spain*)  
F.J. Martinez (*University of Zapatecas, Mexico*)  
G. Meixner (*DFKI, Germany*)  
F. Montero (*University of Castilla-La Mancha, Spain*)  
J. Muñoz-Arteaga (*Universidad Autónoma Aguascalientes, Mexico*)  
Ph. Palanque (*IRIT, France*)  
O. Pastor (*Polytechnic University of Valencia, Spain*)  
H. Paulheim (*University of Darmstadt, Germany*)  
V. Penichet (*University of Castilla-La Mancha, Spain*)  
J. Plomp (*VTT, Finland*)  
S. Praud (*Thales R&T, France*)  
J.C. Preciado (*Universidad de Extremadura, Spain*)  
C. Pribeanu (*ICI, Romania*)

G. Rossi (*University of La Plata, Argentina*)  
A. Seffah (*Université de Troyes, France*)  
R. Tesoriero (*Université catholique de Louvain, Belgium and  
University of Castilla-La Mancha, Spain*)  
Ph. Thiran (*University of Namur, Belgium*)  
D. Tzovaras (*Center for Research & Telematics - Hellas, Greece*)  
J. Vanderdonckt (*Université catholique de Louvain, Belgium*)  
K. Van Hees (*Katholieke Universiteit Leuven, Belgium*)  
M. Winckler (*IRIT, France*)  
J. Ziegler (*University of Duisburg Essen, Germany*)

# Table of contents

## Keynote papers

3D Digital Prototyping and Usability Assessment of User Interfaces based on User Interface Description Languages: Lessons learned from UsiXML and XAML.....	1
<i>Kanai, S.</i>	
UsiXML Extension for Avatar Simulation Interacting within Accessible Scenarios.....	21
<i>Serra, A., Navarro, A., Naranjo, J.-C.</i>	

## Full papers

### Methodological aspects

Support Tool for the Definition and Enactment of the UsiXML Methods.....	27
<i>Boukhebouze, M., Ferreira, W.P., Koshima, N.A., Thiran, Ph., Englebert, V.</i>	
Towards Methodological Guidance for User Interface Development Life Cycle.....	35
<i>Cano Muñoz, F.J., Vanderdonckt, J.</i>	
Improving the Flexibility of Model Transformations in the Model-Based Development of Interactive Systems.....	46
<i>Wiehr, Ch., Aquino, N., Breiner, K., Seissler, M., Meixner, G.</i>	
Towards Evolutionary Design of Graphical User Interfaces.....	53
<i>Guerrero-García, J., González-Calleros, J.M., Vanderdonckt, J.</i>	
Challenges for a Task Modeling Tool Supporting a Task-based Approach to User Interface Design.....	63
<i>Pribeanu, C.</i>	
A Model for Dealing with Usability in a Holistic Model Driven Development Method.....	68
<i>Panach, J.I., Pastor, O., Aquino, N.</i>	
Proposal of a Usability-Driven Design Process for Model-Based User Interfaces .....	78
<i>Montecalvo, E., Vagner, A., Gronier, G.</i>	
Towards a new Generation of MBUI Engineering Methods: Supporting Polymorphic Instantiation in Synchronous Collaborative and Ubiquitous Environments.....	86
<i>Vellis, G., Kotsalis, E., Akoumianakis, D., Vanderdonckt, J.</i>	

### Multi-target user interfaces

Technique-Independent Location-aware User Interfaces.....	96
<i>Tesoriero, R., Vanderdonckt, J., Gallud, J.A.</i>	
Adaptive User Interface Support for Ubiquitous Computing Environments.....	107
<i>Desruelle, H., Blomme, D., Gionis, G., Gielen, F.</i>	
Supporting Models for the Generation of Personalized User Interfaces with UIML .....	114
<i>Bacha, F., Marçal de Oliveira, K., Abed, M.</i>	
Architecture for Reverse Engineering of Graphical User Interfaces of Legacy Systems.....	122
<i>Ramón, O.S., Cuadrado, J.S., Molina, J.G.</i>	
Model-based Reverse Engineering of Legacy Applications User Interfaces.....	128
<i>Montero, F., López-Jaquero, V., González, P.</i>	

UsiXML Concrete Behaviour with a Formal Description Technique for Interactive Systems .....	134
<i>Barboni, E., Martinie, C., Navarre, D., Palanque, Ph., Winckler, M.</i>	
An Abstract User Interface Model to Support Distributed User Interfaces .....	144
<i>Peñalver, A., López-Espín, J.J., Gallud, J.A., Lazcorreta, E., Botella, F.</i>	
<b>Software support for UIDLs</b>	
A Graphical UIDL Editor for Multimodal Interaction Design Based on SMUIML .....	150
<i>Dumas, B., Signer, B., Lalanne, D.</i>	
FlexiXML, A Portable User Interface Rendering Engine for UsiXML .....	158
<i>Campos, J.C., Alves Mendes, S.</i>	
Model-Driven Engineering of Dialogues for Multi-platform Graphical User Interfaces .....	169
<i>Mbaki, E., Vanderdonckt, J., Winckler, M.</i>	
Inspecting Visual Notations for UsiXML Abstract User Interface and Task Models .....	181
<i>Sangiorgi, U., Tesoriero, R. Beuvs, F., Vanderdonckt, J.</i>	
Adaptive Dialogue Management and UIDL-based Interactive Applications .....	189
<i>Honold, F., Poguntke, M., Schüssel, F., Weber, M.</i>	
<b>UIDLs for Specific Domains of Activity</b>	
An Extension of UsiXML Enabling the Detailed Description of Users Including Elderly and Disabled .....	194
<i>Kaklanis, N., Moustakas, K., Tzouvaras, D.</i>	
Issues in Model-Driven Development of Interfaces for Deaf People .....	202
<i>Bottoni, P., Borgia, F., Buccarella, D., Capuano, D., De Marsico, M., Labella, A., Levaldi, S.</i>	
Concurrent Multi-Target Runtime Reification of User Interface Descriptions .....	214
<i>Van Hees, K., Engelen, J.</i>	
User Interface Description Language Support for Ubiquitous Computing .....	222
<i>Miñón, R., Abascal, J.</i>	
A Theoretical Survey of User Interface Description Languages: Complementary Results .....	229
<i>Guerrero-Garcia, J., González-Calleros, J.M., Vanderdonckt, J., Muñoz-Arteaga, J.</i>	
Automated User Interface Evaluation based on a Cognitive Architecture and UsiXML .....	237
<i>Osterloh, J.-P., Feil, R. Lüdtk, A., González-Calleros, J.M.</i>	
User Interface Generation for Maritime Surveillance: An initial appraisal of UsiXML V2.0 .....	242
<i>Robinson, Ch. R., Cadier, F.</i>	

# 3D Digital Prototyping and Usability Assessment of User Interfaces based on User Interface Description Languages: Lessons learned from UsiXML and XAML

Satoshi Kanai

Div. of Systems Science and Informatics, Graduate School of Information Science and Technology  
Hokkaido University, Kita-14, Nishi-9, Kita-ku, Sapporo, 064-0807 (Japan)  
kanai@ssi.ist.hokudai.ac.jp – <http://www.sdm.ssi.ist.hokudai.ac.jp/>

## ABSTRACT

Usability-conscious design while shortening the lead time has been strongly required in the manufactures of information appliances in order to enhance their market competitiveness. Prototyping and user-test of their user interfaces at early development stage are the most effective method to fulfill the requirements. However, fabricating the physical prototype costs much, and they become available only in late stage of the design. To solve the problem, advanced tools for UI prototyping were proposed and developed where a UI-operable 3D digital prototype can be fabricated in a less-expensive way based on user interface description languages (UIDLs), and where the user test and usability evaluation can be performed in more systematic and efficient way than in the physical prototype. In the tools, two conventional UIDLs were adopted: UsiXML and XAML. And their specifications were expanded to support not only declarative description of static structures and dynamic behaviors of the UI, but 3D geometric model of appliance housings and physical UI objects placed on them. Case studies of the automated user tests, usability assessments and UI redesigns utilizing our developed tools are shown.

## Author Keywords

Prototyping, usability-conscious design, UIDL, UsiXML, XAML, user test, information appliances.

## General Terms

Design, Experimentation, Human Factors, Verification.

## ACM Classification Keywords

D2.2 [Software Engineering]: Design Tools and Techniques – *Modules and interfaces; user interfaces*. D2.m [Software Engineering]: Miscellaneous – Rapid Prototyping; reusable software. H5.2 [Information interfaces and presentation]: User Interfaces – *Ergonomics; Graphical user interfaces (GUI); Prototyping*. J.6 [Computer Applications]: Computer-aided Engineering.

## INTRODUCTION

With stiffer global market competition of information appliances, usability-conscious design while shortening the lead time have been required in the manufactures. The

manufactures are placing a premium on increasing efficiency and consciousness of usability in the UI software development of their appliances. The “usability” is defined as “the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use”.

There are several methods of evaluating usability [6]. Among the methods, the “user test” is recognized as the most objective and effective one where end users are directly involved in the evaluation.

In the user-test, designers make end users operate a working “prototype” of the appliance, observe the user’s operational situation, and closely investigate ergonomic issues of the UI design.

However in the current UI software development for the prototype, its specifications are still described by written documents, and the software is implemented based on the documents. This makes the prototype implementation process inefficient if a redesign of the UI is needed after the user test.

Moreover, the “physical” prototypes of the appliances are mostly used in the user-tests. However, fabricating the physical prototypes costs much. For example, as shown in Figure 1, a working physical prototype of a compact digital camera costs a few thousand dollars which is around one hundred times more expensive than the final product. These prototypes also become available only in late stage of the design.



(a) User test

(b) Physical prototype

Figure 1. User test and a physical prototype.

Results of the user-test must be analyzed manually by the usability engineers, and a long turnaround time is needed before finding major usability problems. If problems of the UI design appear at this time, it is often too late for changes within their development schedule.

To solve the problems, digital prototyping of the UI has been introduced in the user-test. A digital prototype is software substitute where its UI functions can work almost in the same way as those in the physical prototype while it can be built in much inexpensive way.

## RELATED WORK

### 2D and 3D digital prototypes

So far, as shown in Figure 2, both 2D and 3D digital prototypes have been proposed and used for simulation and user-test of UI operations in the information appliances. Commercial digital prototyping tools have been already available such as [17,21,22] as shown in Figure 2-(a). And those for conceptual UI design were also studied in [14,15].

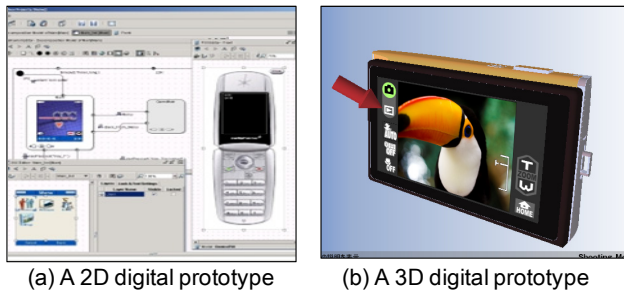


Figure 2. 2D and 3D digital prototypes.

However, since 2D digital prototypes could only be built in these tools and its UI simulation were only limited to 2D and lacked reality, the user performance obtained from these prototypes were not necessarily the same as those of physical prototypes. Former studies including ours [11, 20] showed that operation time and missed operation patterns in a 2D digital prototype were very different from those of the physical prototype and serious usability's problems were overlooked in 2D case.

On the other hand, "3D" digital prototypes allows users to manipulate UI elements placed on 3D housing models of the appliances and to perform more realistic UI simulation than 2D ones. In our former study [11], the missed operation patterns in a 3D digital prototype were also highly correlative to those of the real product.

Unfortunately, there have been few dedicated tools of 3D digital prototyping for information appliances [8,11,20]. In [8], they added a logging function to a VRML player and applied it to the user test of mobile phones. In [20], they developed a tool for 3D augmented prototyping for some hand held information appliances with state-based UI simulation.

### Issues of current 3D digital prototypes

To assure reliability of the user test results to some extent in the early design stage, 3D digital prototypes are more

likely to be suitable for testing and evaluating the logics of the UI, and for clarifying the weaknesses and what needs improvement in the UI design.

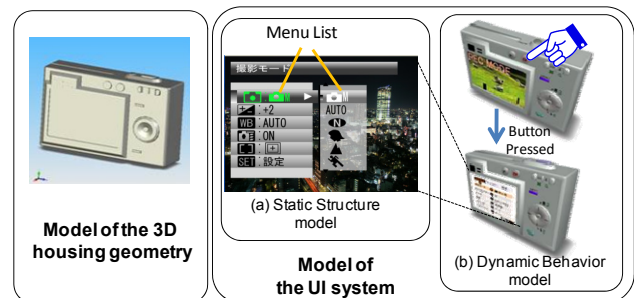


Figure 3. Modeling constituents of the UI operable 3D digital prototype.

As shown in Figure 3, the modeling of UI operable 3D digital prototypes consists of two parts; the model of the UI system and of the 3D housing geometry. Moreover the model of the UI system is divided into the static structure model and the dynamic behavior model. The static structure model of UI describes graphical properties of individual 2D components displayed on the UI screen such as menu-list, button, list-box, slider and image component, and also describes containers for the component layout such as window and tabbed dialog-box. While the dynamic behavior model of UI describes how graphical properties of the 2D components of the UI change in interaction and enables us to simulate the state change of the UI part in the appliance.

Conventional UI operable 3D digital prototypes were built and run using the Web3D authoring tools and their players [8,25,26,27]. However, the following technical issues remain when we use the Web3D as the UI operable 3D digital prototype for user test and usability assessment;

#### *Lack of the static structure model of the UI*

The static structure of the 2D components displayed on a UI screen such as menu list or icon placements cannot be directly modeled in the Web3D formats. So a huge number of digital image files representing snapshots of the UI screen must be built using the 2D drawing or painting tools before the UI simulation and the user test. This preparatory work makes the simulation turn-around very slow.

#### *Lack of the dynamic behavior model of the UI*

The Web3D formats usually provide script-based control function which enables 3D kinematic animations, change of the graphical properties of 3D objects and texture mapping etc. But the function cannot simulate the dynamic behaviors of the 2D components displayed inside the UI screen. The Web3D formats do not also provide any declarative dynamic behavior model of the UI system which is based on state-based or event-based formalisms. These formalisms of the UI fit to the screen transition diagrams in early UI design stage [4,13], and are indispensable to the specification. The lack of the declarative dynamic behavior model forces UI designers to code the behavior using pro-



programming language. But the designers are not necessarily programming professionals, and the task makes the cost of UI simulation and user testing expensive.

#### *Lack of user test and usability assessment functions*

The Web3D formats do not provide any functions of user test execution and usability assessment based on the operational logs. Doing these works manually on the digital prototype makes the usability assessment time-consuming and the assessment results inaccurate.

To solve these issues, the dedicated functions of modeling the static structure of the UI-screens, of modeling the event-based or state-based dynamic behavior of the interaction, and of supporting the computer-aided test execution and the usability assessment must be added to the traditional Web3D authoring tools and players.

To achieve them, our research group has been developing advanced tools for UI prototyping were proposed and developed where a UI-operable 3D digital prototype can be fabricated in a less-expensive way based on user interface description languages (UIDLs), and where the user test and usability evaluation can be performed in more systematic and efficient way than in the physical prototype. In the tools, two conventional UIDLs were adopted for UI specification and implementation in the final development stage; UsiXML [24, 28, 29] and XAML [18, 31]. And their specifications were expanded to support not only declarative description of static structures and dynamic behaviors of the UI, but 3D geometric model of appliance housings and physical UI objects placed on them.

In the following sections, the functions and features of the two developed tools each of which is respectively based on UsiXML or XAML are introduced. Case studies of the automated user tests, usability assessments and UI redesigns

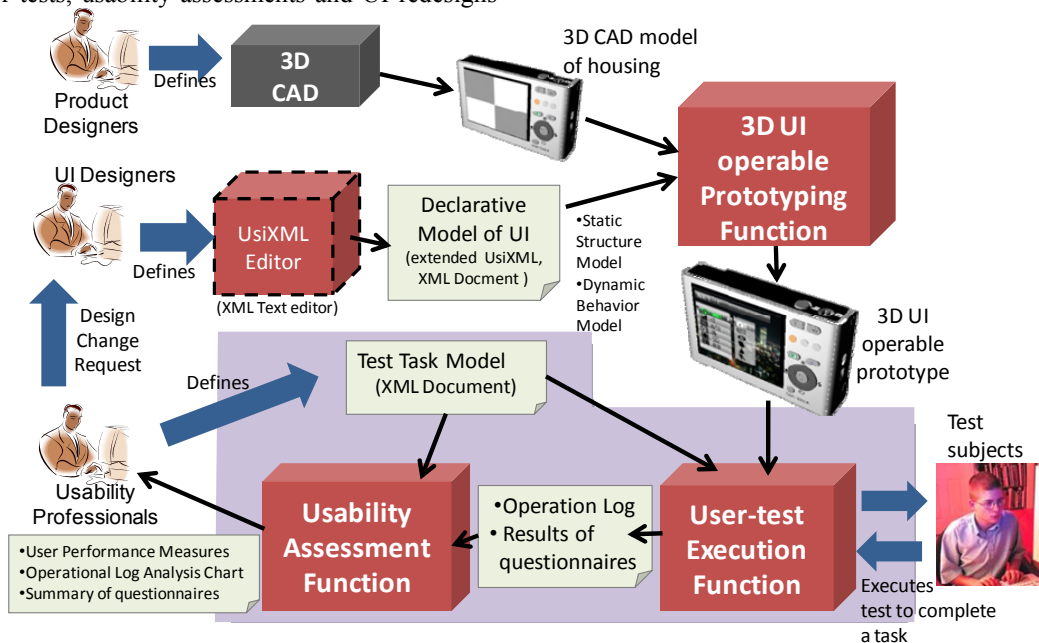
utilizing our developed tools when applied to UI prototyping of digital cameras on the market are shown.

### **UsiXML-BASED 3D DIGITAL PROTOTYPING AND USABILITY ASSESSMENT**

#### **An Overview**

As the first approach, the 3D digital prototyping and usability assessment tools based on Usi-XML were developed by our group [11]. Figure 4 shows the functional overview of the tools. The features of the tools are summarized as follows:

- (1) The model-based specification of UsiXML, which is one of the XML-compliant UI description languages, was extended to enable the UI designer to specify the declarative model not only of the logical UI elements displayed on the screen but of the physical UI elements such as buttons and dials placed on appliance's housings.
- (2) 3D UI prototyping and simulation functions were developed where the extended UsiXML was combined with the 3D CAD models of the housings and where the UI interaction were simulated based on the declarative model of the UI behavior described by the UsiXML.
- (3) The automated usability assessment functions were developed which in such a way that they were tightly connected to the declarative model of the UI and to the simulation functions.
- (4) An example of the usability assessment and the UI redesign using the 3D digital prototype of a digital camera using our tool was shown the effectiveness and reliability of our proposed tool.



**Figure 4. An overview of the UsiXML-based prototyping, user-test and usability assessment tools.**

## UsiXML and its extensions

### UsiXML

Several XML-based user interface mark-up languages have been recently proposed to make UI prototyping of PC applications reduced and structured: UIML [23], XUL [30], and UsiXML [24]. Each of them specifies models for defining the UI and allows us to describe the UI model in declarative terms. They enable the UI designers or UI programmers to specify what elements are to be shown in the UI and how should they behave in XML documents. This concept becomes an advantage for defining 3D digital prototypes of handy information appliances from the following standpoints:

- (1) The static structure of the 2D component objects displayed on the UI screen is explicitly and declaratively modeled and described by the XML document. The snapshot of the UI screen can be automatically drawn in the simulation based on the static structure model if we realize the real-time rendering function for the model. It can eliminate the preparatory work of the UI simulation, and makes its turn-around efficient.
- (2) The dynamic behavior of the UI interaction has to be described by script or ordinary programming language in most of the UI mark-up languages (UIML, XUL and XAML). However, in the UsiXML, the behavior can also be explicitly described as an event-based model. The model can eliminate the coding of UI dynamic behavior simulation if an execution function of the behavior model in the simulator of the 3D digital prototype is realized.
- (3) The user test and the usability assessment can be automated if the static structure and the dynamic behavior models of the 3D digital prototype are reused for analyzing the property of the subject's operations in the usability assessment functions. It can make the cycle of prototyping-testing-redesigning very efficient.

Therefore, we introduced UsiXML to our study, because it can describe the dynamic behavior model of the UI in a declarative way and model the UI at a different level of abstraction. UsiXML was originally proposed by Vanderdonck *et al.* [16, 28]. It aims at expressing a UI built with various modalities of interaction working independently. The UI development methodology of UsiXML is based on the concept of MDA (Model Driven Architecture).

### Issues of the UsiXML from the aspect of 3D digital prototypes.

The concept and specification of UsiXML is advanced in UI prototyping, but it has still the following issues when we directly use it for developing UI operable 3D digital prototypes and for the usability assessment;

- (1) As shown in Figure 5-(a), the CUI of UsiXML specifies the static structure model of 2D UI component objects displayed on the UI screen, but the current CUI only specifies the static structure model for WIMP

(Windows-Icons-Menus-Pointers)-type GUI. In UsiXML, there are no specifications for the physical UI elements such as buttons, dials, lumps and dials placed on the information appliance's housing shown in Figure 5-(b), which are essential for modeling the UI operable 3D digital prototypes.

- (2) Many experimental automated tools have been developed for UsiXML. However, there is no 3D UI prototyping and simulation tool available to UsiXML at present. The event-based dynamic behavior model is specified in the CUI, but it has not been reported yet how the dynamic behavior of the UI is described concretely, nor how the model of the CUI should be implemented on a particular Web3D format.
- (3) UsiXML has been originally developed for UI prototyping, but at present there is no specification and no supporting tool concerning user testing and usability assessments which utilize the UI prototype. Therefore, we cannot incorporate the functions of user testing and usability assessment into UI prototyping based on UsiXML.

### Extensions of the CUI model of UsiXML

The current specifications of the CUI in UsiXML mainly consist of the static structure model of the objects displayed on the UI screen and the dynamic behavior model of the interactions of the UI. The static structure model further consists of the UI object model and the object container model. The UI object model expresses the individual GUI component object displayed on the UI screen such as buttons, list-boxes, image components, etc., while the object container model does the whole-part relationships among the multiple GUI objects such as a window and a tabbed dialog box. The dynamic behavior model consists of the event-based rewriting rules of the UI screen in interaction and of the procedures and variables to refer to the internal data of the device.

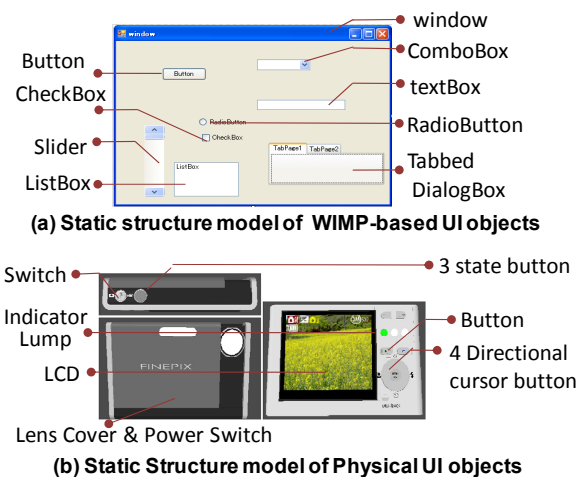


Figure 5. Types of UI component objects modeled in the static structure model.

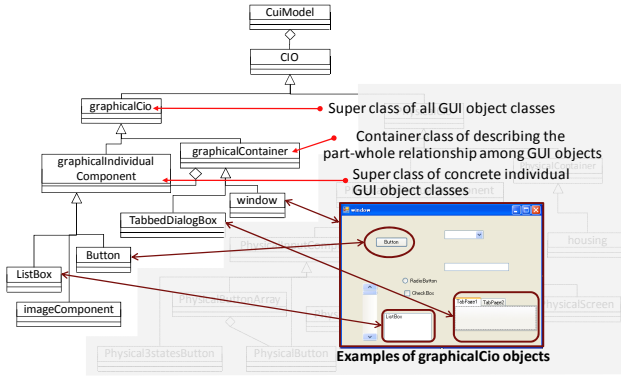


Figure 6. UML class diagram and its example of the original CUI model of UsiXML.

In this research, we extended this static-structure-model part of UsiXML so as to fit it to the UI operable 3D digital prototyping. Figure 6 indicates the UML class diagram and its example which expresses a part of the original CUI model structure in the UsiXML. In the structure, *graphicalCio* is the super class of all GUI object classes in the model, and *graphicalContainer* is the container class for the GUI objects. The concrete classes of the GUI objects and the object containers of UI are defined as a subclass of these two classes.

On the other hand, Figure 7 indicates the class diagram and its example of our extension of the CUI model. We extended the class structure of the CUI model to express the physical UI objects such as physical buttons, dials and lumps placed on the 3D geometric model of the appliance' housing. First, we added a new class *physicalCio* to the same class hierarchy level as one of the *graphicalCio* class. Then we further created two new classes of *PhysicalIndividualComponent* and *PhysicalContainer* as subclasses of the *graphicalCio*. The *PhysicalIndividualComponent* class expresses the one for modeling each the physical UI object, and the *PhysicalContainer* class does the physical housing of the appliances which play a role of the virtual container in aggregating the physical UI objects. Moreover, as the subclasses of *PhysicalIndividualComponent*, we added a *PhysicalButton* class and *PhysicalScreen* class to the subclasses of *PhysicalIndividualComponent* in order to express concrete buttons and LCDs placed on the housing. Figure 6 shows the correspondence between the physical UI objects in a digital camera and the classes describing them.

#### Design of the XML document structure of the extended CUI model

The current version of the UsiXML does not specify the explicit XML encoding rule of the CUI model. Therefore, we specified a tag structure of the XML document of our extended CUI model independently. This tag structure is imported to the 3D UI operable prototyping functions and is used for the 3D UI simulation. Figure 8 shows an example of the tag structure in XML document and their presentations in the UI screen image.

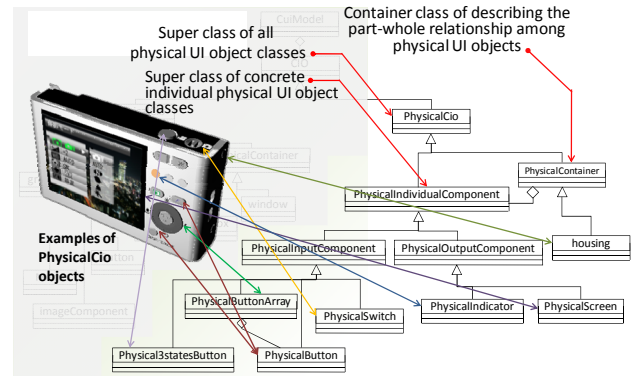


Figure 7. UML class diagram and its example of our extended CUI model of UsiXML.

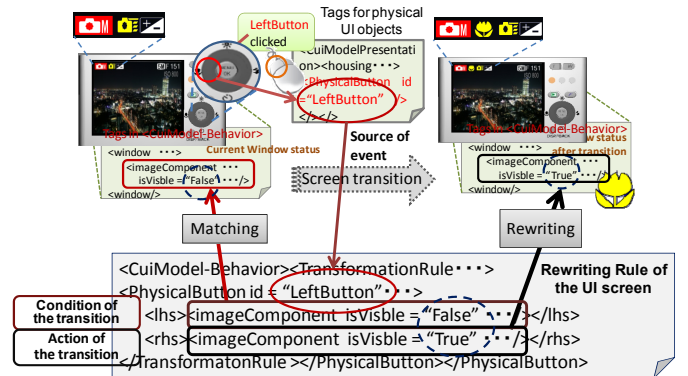


Figure 8. UI screen transition rule described in XML document of the extended UsiXML.

The structure consists of a *<CuiModel-Presentation>* tag and a *<CuiModel-Behaviour>* tag. The former represents our extended static structure model of the CUI which express the objects displayed in the UI screen, while the latter does the dynamic behavior model which corresponds to the UI screen transition. And concrete CUI objects are described inside these two tags in our XML document.

To describe the UI screen transition, we set up a *<TransformationRule>* tag inside the *<CuiModel-Behaviour>* tag which describes the general graph rewriting rule mechanism defined in the original UsiXML specification. As shown in Figure 8, in the *<CuiModel-Behaviour>* tag, we put the pair of a condition tag *<lhs>* and an action tag *<rhs>* together by each tag corresponding to the subclass of *PhysicalInputComponent* class. The condition tag expresses a condition where the screen transition occurs because of an event coming from the physical UI objects. And the action tag expresses the state of the UI screen after the screen transition occurs.

In the user-test execution function, if an event occurs in an object belonging to any subclass of the *PhysicalInputComponent* class, the function tries to find a *<TransformationRule>* tag which has the same tag id as the source of the event from all tags inside the *<CuiModel-Behaviour>*

tag. And if the current tag and its attribute values in the `<CuiModel-Presentation>` tag are exactly identical to the ones written in the `<lhs>` tag in the `<TransformationRule>` tag, then these attribute values are overwritten as the one in the `<rhs>` tag. This overwriting mechanism implements the UI screen transition on the 3D UI operable prototype based on UsiXML.

### 3D UI operable prototyping function

#### 2D CUI object renderer

We developed a 3D UI operable prototyping function where the extended UsiXML is combined with the 3D CAD models of the housings and the UI interaction were simulated based on the dynamic-behavior model of the UsiXML. The 3D UI operable prototyping function consists of the 2D CUI object renderer and the 3D UI operable simulator which is a remodeling of Web 3D player (Virtools [27]).

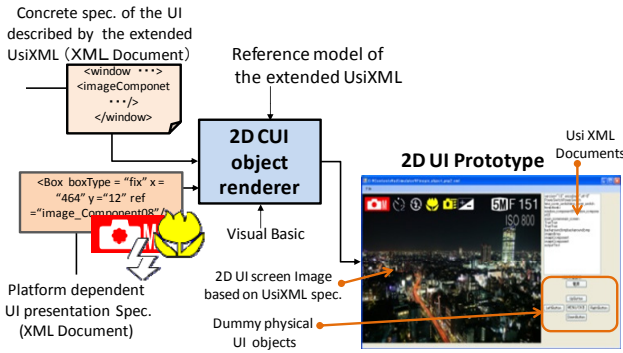


Figure 9. Functions of 2D CUI object renderer.

The 2D CUI object renderer is a VisualBasic application developed by ourselves. Figure 9 shows the function of the renderer. The renderer interprets the XML document of the extended UsiXML and accepts another XML document which defines the platform dependent UI presentation specification such as the concrete position of each GUI object and object containers on the UI screen. It renders the 2 dimensional UI screen image on the fly according to the UI screen transition rule described in the XML document. It also renders the dummy physical UI objects on the same 2D UI screen. So the renderer also enables the UI designer to do the 2D UI software prototyping when the renderer is used alone. The 2D CUI object renderer is executed during the 3D UI simulation cooperating with the Web 3D player to provide the main function for the UI simulation.

The 2D CUI object renderer enables UI designers to eliminate their preparatory works of generating a huge number of snapshot images of the UI screen, and makes the turnaround time of the 3D UI simulation short.

#### 3D UI operable prototype simulator

Figure 10 shows the 3D UI operable prototyping function. The function consists of the 2D CUI renderer and the 3D UI operable prototype simulator which is a commercial Web 3D player (Virtools [27]). We remodeled the players

so that the Web 3D player runs with the CUI renderer simultaneously, and they exchange events via socket communication during the 3D UI simulation for the user-test.

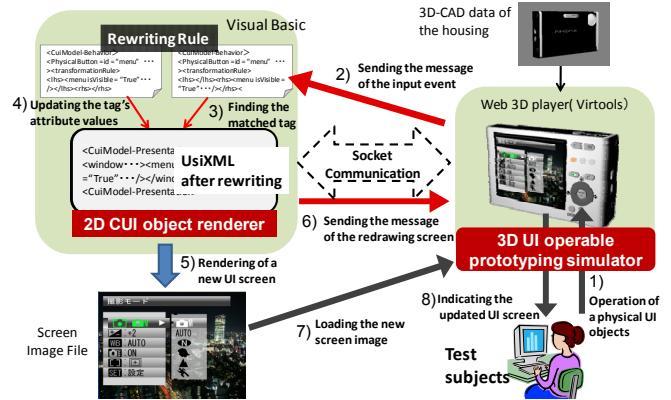


Figure 10. 3D UI simulation function based on UsiXML.

In the preliminary step of the 3D UI simulation, 3D CAD data of the housing is imported to the Web 3D player from 3D-CAD systems (CATIA, Solidworks, etc.) in the format of 3D-XML[1]. The 3D-XML is a universal lightweight XML-based format for the sharing of 3D CAD models. The format includes the assembly and each part has its own unique part-name. In the model, a 3D object which is the source of an event or the destination of an action is modeled as a single part. A button, a switch knob, an LCD screen and an LED are typical examples of these objects.

In the Virtools player, a UI designer links an event and an action described in the `<lhs>` tag and the `<rhs>` tags in the `<TransformationRule>` tag in the dynamic behavior model of UsiXML to messages of the Virtools player. A message consists of a unique event-name, part-name and event-type. For an example, an event of “*button\_1\_pushed*” in the UsiXML model is tied to a message consisting of “*message-1*” (event-name), “*button-part-1*” (part-name) and “*on\_clicked*” (button-type). Consequently this linking operation builds all logical links between the messages in the Virtools and events or actions in dynamic behavior model of UsiXML.

As shown in the processing sequence in the Figure 10, the 3D UI simulation is executed as the following procedure:

- (1) The user manipulates the 3D housing model of the appliances and operates a physical UI object such as a 3D button placed on the housing model using the mouse on the player.
- (2) The operation of the physical UI object is recognized as an event in the player and it sends a unique message of the event to the 2D UI object renderer via socket communication.
- (3) The renderer analyses the incoming message to pick up the event, and tries to find a `<TransformationRule>` tag which has the same tag id as the



source of the event from all tags in the *<CuiModel-Behavior>* tag.

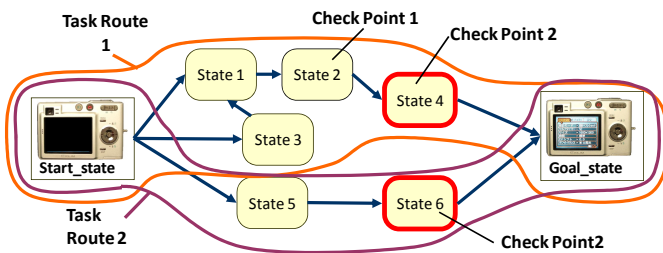
- (4) If the current tag and its attribute values in the *<CuiModel-Presentation>* tag are identical to the ones written in the *<lhs>* tag in the *<TransformationRule>* tag, then these attribute values are updated to the new ones according to the content of the *<rhs>* tag.
- (5) The renderer then redraws a new UI screen image after the screen transition in a scratch file according to the updated attribute values.
- (6) The renderer sends another message of the screen redraw event to the player.
- (7) The player loads the new screen image from the updated scratch file.
- (8) The texture rendered on the face in the 3D housing model which corresponds to the UI screen changes to the new screen image.

By repeating the procedure, the 3D UI simulation on the 3D housing model which cooperates with the 2D UI simulation is realized in the prototyping function. The UI simulation rule is completely described in the XML document of UsiXML in declarative way, and the simulation execution is completely managed in the developed renderer.

### User test execution function

#### Test task and task model

In the user test, a subject is asked to complete a set of tasks by operating the UI, and actual operations for the task are analyzed to evaluate the usability. In our tools, we designed a “test task model” and made a logical link between the task model and the dynamic behavior model of the UI to automate the usability assessment. Figure 11 shows the test task model. This task model is originally developed for the “state-based” dynamic behavior model of the UI screen [9].



**Figure 11. Test task model.**

A task consists of a start state, goal state and a list of task routes. And a task route consists of a list of checkpoints. A checkpoint is a state where a correct sequence of UI operation must pass. A start state and a goal state refer to the state in the UI behavior model.

Generally multiple correct operations of the UI exist in order to achieve a goal. Therefore multiple task routes can be allowed for one task in the model. Moreover, lower and

upper bounds for the number of operations or an allowable elapsed time between every two neighboring checkpoints can be also defined. If the elapsed time of a subject’s operation stays within these bounds, the operation is judged to be correct. In this way, usability professionals can adjust a range of correct operations in the task when determining the number of error operations and the task completion.

#### State evaluation and operation logging

The dynamic behavior model of the UsiXML is expressed by a set of transformation rules which describe how the attribute values of the objects displayed in the UI screen have to be changed in response to the input event. Therefore it is the “event-based” dynamic behavior model, and the model does not have the explicit notion of “state” of the UI.

However, in the user test, the task model was originally designed for the “state-based” dynamic behavior model of the UI system [9]. And the notion of the state is indispensable for defining test task, recording user operation logs and identifying missed operation. Without the notion of the state, it is very difficult for the usability professionals to capture the situations of user operations.

To solve the problem, we added the state evaluation function in the user test execution function. In this function, a set of conditions which describes attribute values of a CUI object to be taken in a particular “state” are defined in an XML document in advance. And the function always evaluates whether the condition holds or not in the UI simulation of the user test. If a set of attribute values of the displayed CUI object is perfect match for the condition, then the function reports that the UI transits to the certain predefined state.

In some cases, there might be several different modes with a same set of attribute values in the behavior model. In this case, our state evaluation function cannot identify these modes as different states. Inserting an extra attribute for distinguishing one mode from others into the original set of attributes can solve this problem.

Using this mechanism, the user test execution system can recognize which state the UI is in during the simulation. We also developed the operation logging function based on the state evaluation function. The logging function records all subject’s operations in the form of the combination of a time, a previous state, a next state and an input event coming from the user interaction. The function saves these logs in a XML document. We integrated the state evaluate function and the operation logging function with the user test execution function, and enabled the usability professionals to manage the user operation log for every task during the user test.

#### Test execution

Shown in an example of Figure 12, at the beginning of every new test session, the user test execution function indicates a goal of each task in the form of an imperative sentence. “Switch shooting mode from still to video” and “set

the self-timer for 20sec” are the typical examples of the goal. The goal is indicated on the other window just above the 3D digital prototype.

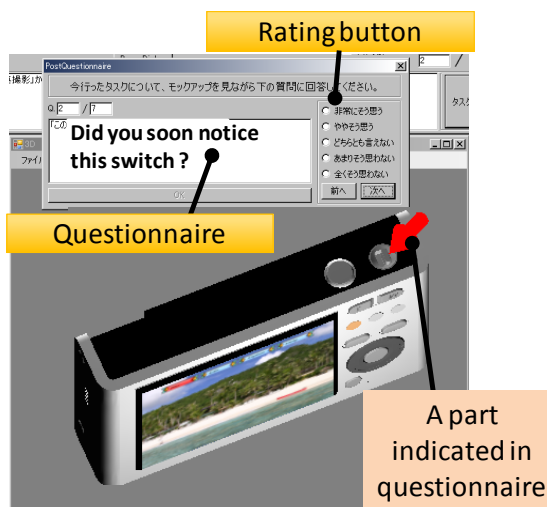


**Figure 12. 3D digital prototype under user-test.**

The subject is asked to complete the task by operating the UI of the 3D digital prototype in the session. The subject manipulates the 3D prototype by rotating, translating and enlarging it and operates the physical UI objects on the prototype by clicking or dragging them with a mouse in the virtual space. If the physical UI objects, such as buttons, are located on a different surface that cannot be seen from the 2D UI screen such as LCD, the subject have to rotate the prototype so as to make these objects face him/her.

During the UI operation, the operation logging function records a sequence of state-transitions of the UI as a list of combinations of state and event together with the time stamps.

At the end of the every test session, the operation logging function compares the actual sequence of state-transitions with all pre-defined task routes, allowable number of operations and elapsed time between checkpoints. Then the logging function judges whether the subject’s operation of the session for this task was correct or not, and identify which checkpoint state the subject made mistakes in his/her operation.



**Figure 13. Digital questionnaire on the prototype.**

If the operation is judged to be wrong, a set of digital questionnaires are progressively popped up on another screen at the end of the test session. One questionnaire is displayed corresponding to one checkpoint state at which the subject made a mistake. A portion on the UI object in the 3D digital prototype related to each questionnaire is automatically pointed by the tool as shown in Figure 13.

The subject is asked to answer to each questionnaire by choosing his/her impression from five grades. For example, when the questionnaire is “Did you soon notice this button?”, the rating is the one of “Strongly agree: 5”, “Agree: 4”, “Yes and No: 3”, “Disagree: 2” and “Strongly disagree: 1”. The subject answers this rating only by clicking one of the radio buttons placed on the questionnaire as shown in Figure 13. The rating is stored to clarify what needs improvement in the design in the usability assessment function. The detail of this digital questionnaire is explained in the next section.

### Usability assessment function

#### User performance measures

The usability assessment function investigates the operation log data by comparing it with the test task and the dynamic behavior model of the UI. The function outputs the measures of usability assessment as a result of the analysis. The analysis can be automatically processed, and the function outputs measures of the user performances.

We adopted the following three measures based on three basic notions of usability (effectiveness, efficiency and satisfaction) defined in [7];

- 1) The number of user events inputted in each task and in each subject,
- 2) Elapsed time in each task and in each subject,
- 3) Personal task achievement ratio, and
- 4) Scores for SUS questionnaire [2].

#### Operational log analysis chart

An actual sequence of operations is compared with one of the correct task routes defined in the test task, and the result of the comparison is graphically displayed in the form of “operational log analysis chart” on the lower part of the screen.

Figure 14 shows the notation of this operational log analysis chart. Each rectangle shows a state, and a line between two rectangles does a transition between states. A left-most rectangle in the chart indicates a start state, and a right-most rectangle does a goal state. The upper most horizontal white straight line indicates transitions on a correct task route, and every rectangle with orange edges on this line except both ends corresponds to each checkpoint. While the blue rectangles and blue lines indicate actual operation sequence of the subject. If a subject does UI operations whose elapsed time or number of events between two neighboring check points exceeds the predefined bounds, the tool judges that a subject did a wrong operation on the

UI, and draws additional blue rectangles and blue lines in downward direction corresponding to these wrong operations.

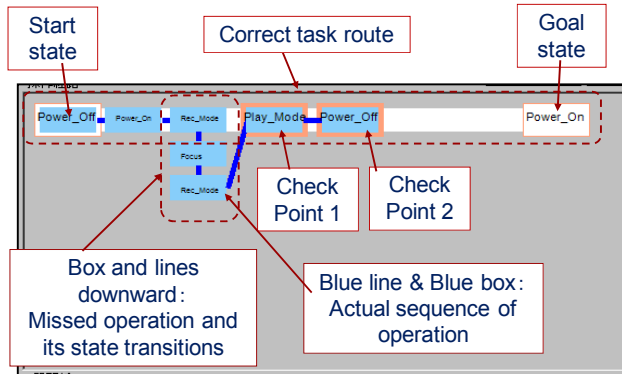


Figure 14. Operational log analysis chart.

Therefore, as the depth of the chart becomes larger, the usability professionals can easily recognize at a glance which states the subject did more missed operation in the task.

#### Digital questionnaires

The operational log analysis chart can clarify particular states in the UI screen transition where many subjects made errors in the UI operations. However, the chart cannot provide enough information to enable UI designers to find the cause of missed operations and to redesign the UI in order to improve the user's performance.

To solve this problem, the digital questionnaire execution function has been developed to identify the causes of the missed operations. The structure of the digital questionnaires is built based on an extension of the cognitive walkthrough method which is dedicated by the HCI (Human Computer Interaction) model.

To construct the digital questionnaires, we first introduced the extended HCI (Human-Computer-Interaction) model. The extended HCI model [3] is an extension of the CW method whose questionnaire is formulated based on an extended model of Norman's HCI[19] that explicitly distinguishes between object and action, and between perceiving and understanding. The questionnaires were originally used for the usability evaluation of Web sites. The questionnaires based on the extended HCI model are easier to understand for end users (Table 1).

However, as shown in Table 1, the questionnaires based on the extended HCI model still have abstract expressions. We further make them more straightforward and concrete when using them in the user tests of information appliances so that the end users can understand them more easily as shown in Table 1. For example, a questionnaire of the extended HCI model which examines the perception of the object to be manipulated is expressed as "Will users be able to perceive the object to be manipulated?". We changed it to more plain one; "Did you soon notice this button?" when applying it to the test of a digital camera.

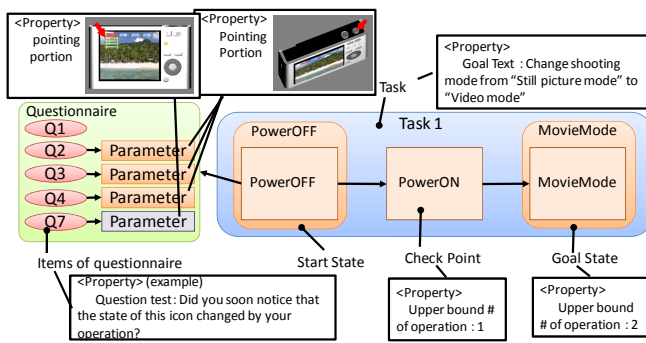
Moreover, we also implemented a function that automatically points to a 3D object which corresponds to "this button" or "here" on the 3D digital prototype when indicating a certain portion in the questionnaire as shown in Figure 14. This function enables end users involved in the test to understand each item of the questionnaire more easily, and also enables usability professionals to save a quite bit of manpower for constructing the questionnaire.

The proposed digital questionnaire enables many end users to take part in the cognitive walkthrough evaluation and to answer the questions by actually manipulating the 3D digital prototype whose UI can work as same as the final appliance does. This feature can greatly increase the reliability of the user test's results.

Cognition Process of Extended HCI Model	Items of questionnaires based on Extended HCI [Hori & Kato 2007]	Items of questionnaires used in our system
Formation of Intension of manipulation	Does the user try to accomplish the correct action ?	Did you easily understand what you should do for the appliance by reading the task ?
Perception of objects	Can the user perceive the object to be manipulated ?	Did you soon notice this [input element name]?
Interpretation of objects	Can the user understand that the perceived object is the correct object ?	Did you soon understand that you should operate this [input element name] ?
Perception of actions	Can the user perceive his/her actions of manipulation ?	Did you soon understand how you should operate this [input element name] ? (by pushing, sliding etc.)
Interpretation of actions	Can the user come up with actions of manipulation which should be applied to the object ?	N.A.
Execution of actions	Can the user certainly execute the correct action?	N.A.
Perception of the effect	Can the user notice the change of the state ?	Did you soon notice that the state of this [output element name] changed by your operation?
Interpretation of the results	Can the user understand what state the system is after the state change ?	Did you easily understand how the state of the appliance changed as a result of the operation by observing the state change of [input element name] ?
Evaluation of the results	Can the user understand that he/she advances toward the solution of the task by observing the system state ?	Did you soon understand whether your operation is correct or not by observing the state change of [input element name] ?

Table 1. Extended HIC model and the questionnaires.


For defining the digital questionnaire, as shown in the Figure 15, the usability professionals assign one questionnaire to a check point in the test task model. The professional also specifies a particular portion of the UI screen image or the 3D model of the housing which should be indicated on the 3D prototype. The standard template of the questionnaire is predefined as a stencil with standard properties in the Visio, and the usability professionals can paste the stencil to the particular task, which is graphically displayed in the Visio, and input the sentence of a question in the property value [10].



**Figure 15. Association of the digital questionnaire with the test task model.**



**Figure 16. The appliances for the user test.**

Task goal	Setting a self timer to 10 sec from power-off state.
Minimum # of operation	2
Physical UI objects to be operated	
Objects indicated in Questionnaires	<ul style="list-style-type: none"> <li>• Front lens cover</li> <li>• Downward cursor button</li> <li>• Self time icon displayed in LCD</li> </ul>

**Figure 17. The task of the user test.**

## A case study of usability assessment and redesign

### Task Setting

A case study was done which consisted of the user test, the usability assessment and the UI redesign based on the results of the assessment. A compact digital camera (Fuji FinePix-Z1) on the market shown in Figure 16 was selected for the user test.

The goals of the case study were:

- to investigate whether the UI operable 3D digital prototype and our tool can clarify the weaknesses in the

UI design where many subjects often make mistakes in their UI operation,

- to investigate whether the digital prototype and the tool can clarify why many subjects often make mistakes in the operation and what needs improvement in the UI design,
- to investigate how small the differences in the assessment results are between the digital prototype and the real product, and
- to evaluate how efficiently the redesign of the UI can be done using the UsiXML and our tool.

A UI operable 3D digital prototype of this camera was built as shown in Figure 12 and used for the assessment. For the digital prototype, we modeled the dynamic behavior model of the camera UI which has 34 states and 224 transitions including neighboring transitions of correct operation sequences of the task. A real product was simultaneously used for the assessment, and the result was compared with that of the 3D digital prototype.

The prototype was operated by 14 subjects (male and female students of age 20-29) who had not used the same model. 7 subjects took part in the test using the UI operable 3D digital prototype, and the other 7 subjects used the real camera.

We defined the task of “Setting a self-timer to 10 seconds from power-off state” in the test which is one of the occasionally-used operations. In the task, as shown in Figure 17, first the user has to turn on the power switch by sliding the front lens cover, and then to switch the mode from the manual shooting to the self-timer setting with 10 seconds by pushing a downward cursor button once. If the camera reaches to this goal state (self-timer 10 seconds), a small circular white icon which symbolizes the self timer appears on the top of the LCD. The subject has to notice that this icon indicates the goal state and that he/she completed the task.

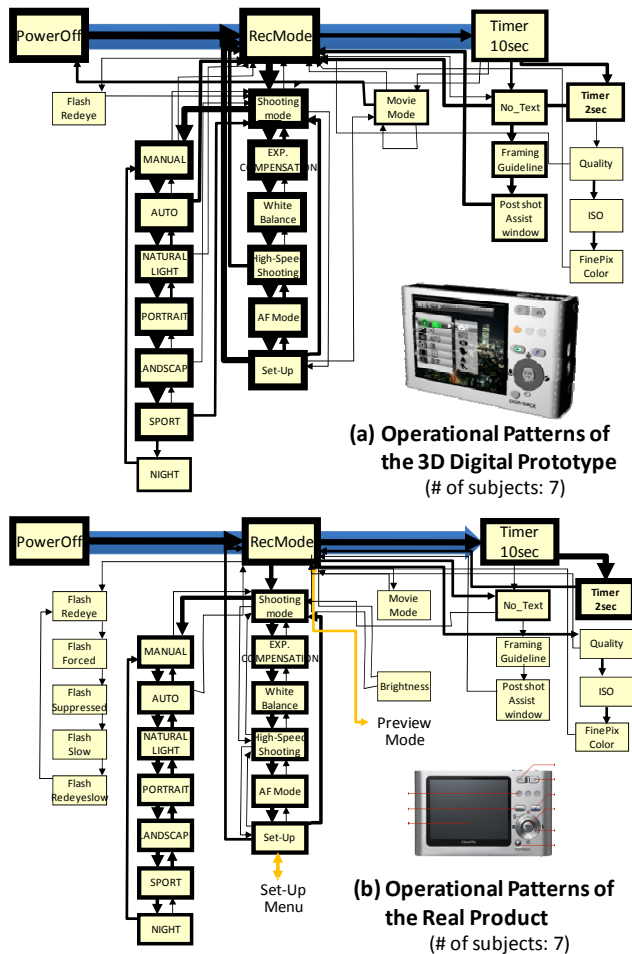
### Analysis of operational patterns

To investigate the subjects’ actual operational patterns both of the 3D digital prototype and the real product, actual sequences of operations including missed operations of each subject are put together. The sequences are schematically drawn as a “summarized operational log analysis chart”.

This chart can be made by superimposing an operational log analysis chart for one subject shown in Figure 14 onto ones of the other subjects.

In this summarized chart, the notation of correct and missed operations is the same as the one in the personal version described in the previous section. But the width of a directed path on the chart is proportional to the number of subjects who passed over the transition corresponding to the line. Therefore, a wider directed path indicates that more subjects passed over the routes of operation to complete the task.





**Figure 18. Comparison of the summarized operation log analysis charts of the user test.**

Figure 18 shows the two summarized operational log analysis charts; the chart for the subjects who operated the 3D digital prototype (Figure 18(a)) and the other for the ones who did the real product (Figure 18(b)).

Both analysis charts show the clear facts that;

- at the “Rec\_Mode” state, many users stepped into the wrong path aiming to the “Shooting mode” state instead of the correct path to the “Time\_10sec” state,
- even at the “Timer\_10sec” state which is a goal of the task, many users went past the state and continue operating to reach the “Time\_2sec” state, and
- the pattern of missed operations of subjects who operated the 3D digital prototype is very similar to that of the real product.

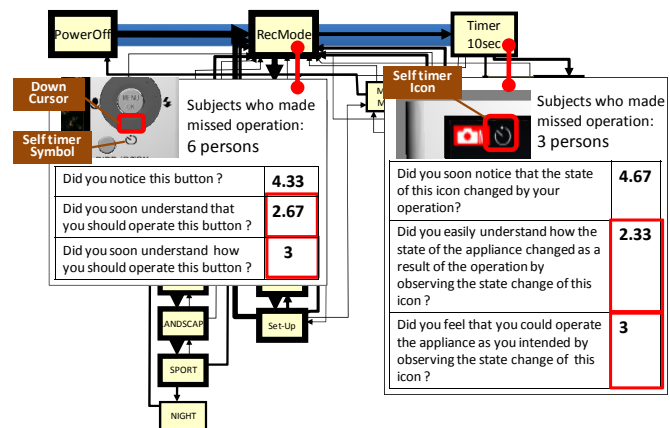
From this comparison, the differences of the operational patterns were small between the UI operable 3D digital prototype and the real product. It was also shown that the 3D digital prototype could find the weakness in the UI design where many subjects often take missed operations similar to the those missed operations performed by the ones using the real product.

### Analysis of digital questionnaires

When reading only from the summarized operational log analysis chart, we could not discover the reasons why so many subjects made mistakes in those particular states and what needs improvement in the UI design. So we further analyzed the rating from the subjects in the digital questionnaires indicated on the 3D digital prototype. Moreover, the ratings obtained in the digital prototype were compared with those in the real product. To the subjects who used the real product, the questionnaires were manually indicated to them, and the ratings were written by themselves.

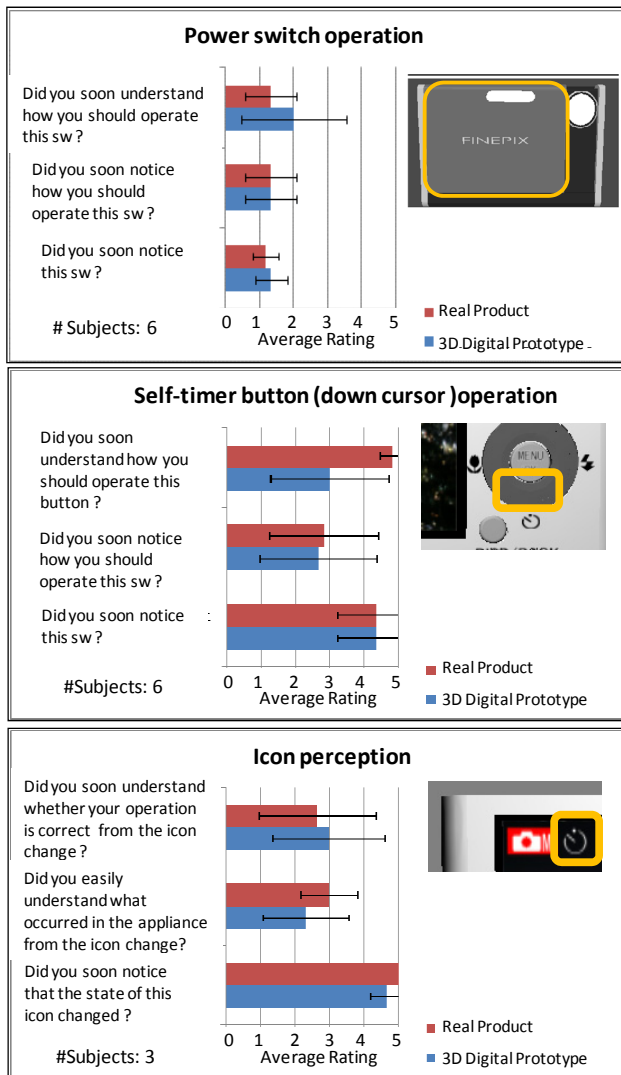
The average ratings from the subjects for the digital questionnaires indicated at the “Rec\_Mode” state and the “Timer\_10 sec” state were shown in Figure 19. “Rec\_Mode” state means that the camera is in the manual shooting mode, and “Timer\_10 sec” state that it is in the self-timer setting mode with 10 seconds and is the goal state.

The ratings at the “Rec\_Mode” state from the 3D digital prototype suggest that many subjects could recognize a down cursor button itself, but could not notice that they could move to the self-timer setting mode from the manual shooting mode by pushing this cursor button. Therefore, from the rating analysis, we found that the small symbol indicating the self-timer printed on the housing surface near the cursor button needs to be changed to a new one which can give us the self-timer function at a glance.



**Figure 19. Average rating for the digital questionnaires at 2 states in question in the original UI design.**

On the other hand, the ratings at the “Timer\_10sec” state suggest that the subjects could notice the change of the system status caused by their operations, but could not understand what occurred in the camera and whether they correctly accomplished the task. This means that this white icon displayed on the LCD in Figure 19 could be noticed by many subjects, but did not enable them to notice that the self-timer settings had already been set to 10 seconds. Therefore, from the rating analysis, we finally found that the small timer icon indicated on the LCD in the original UI design need to be improved to the new one which can give us the setting value of the self-timer at a glance.



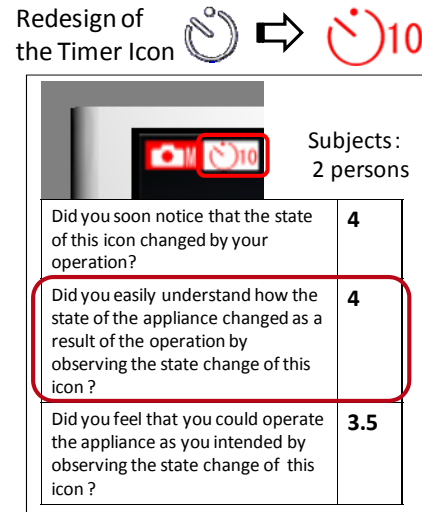
**Figure 20. The differences in the ratings in three questionnaires between the 3D digital prototype and the real product.**

Figure 20 shows the difference in the ratings in three questionnaires between the subjects who used the UI operable 3D digital prototype and the ones who used the real product. There are strong correlations of the ratings between the digital prototype and the real product in all three cases. Therefore, the combination of the UI operable 3D digital prototype and the digital questionnaires enables the UI designers to reveal what needs improvements in the UI and how they should be improved as minutely as a real product does.

#### *UI redesign based on the digital questionnaires*

The rating analysis in the digital questionnaires revealed that there are two candidates which should be redesigned in the original UI design; a small symbol indicating the self-timer printed on the housing, and the small self-timer icon indicated on the LCD shown in Figure 19.

Based on the analysis, we evaluated how efficiently the redesign of the UI can be done using the UsiXML and our tool. In this study, only the second redesign candidate was implemented.



**Figure 21. Improved average rating for the digital questionnaires at “Time 10sec” state in the modified UI design.**

As shown in the Figure 21, we redesigned the shape and color of the timer icon to the new ones so that the background color becomes conspicuous and the timer setting value is explicitly drawn in the icon. An additional test was executed for the new four subjects who used the 3D-digital prototype with the redesigned icon. The result of the test showed that two of four subjects could complete the task without wrong operations. And the rest could also complete the task with small wrong operations. Moreover, the result of the ratings of the digital questionnaire of the redesigned icon indicates that more subjects could easily find that the self-timer settings had already been set to 10 sec.

In this redesign work, it only took 10 minutes to redraw the icon image and 1 minute to rewrite a small part of the tag contents in the XML document of the UsiXML.

From the whole results of the case study, we obtained the following conclusions;

- The summarized operational log analysis chart based on the 3D digital prototype enabled UI designers to discover the weakness in the UI design where many subjects make mistakes, and also that the digital questionnaires enabled them to clarify what needs improvement and who they should be improved in the design.
- There was a strong correlation of the operational log analysis chart and the ratings in the questionnaire between the 3D digital prototype and the real product. Therefore, the UI operable 3D digital prototype could

replace a real product or a physical prototype while keeping the ability to discover the usability problems of the UI logic.

- The UI operable 3D digital prototype based on the UsiXML and the automated usability assessment functions can complete the works of prototyping-test-redesign more efficient than the current manual based assessment can.

## XAML-BASED 3D DIGITAL PROTOTYPING AND USABILITY ASSESSMENT

### Issues in UsiXML-based prototyping

Usi-XML-based 3D prototyping and usability assessment tools in the previous sections enabled us to achieve the realistic simulation fidelity of the UI, to declaratively and explicitly describe the static structure and dynamic behaviours and to execute the very efficient and systematic usability assessments. However, in these tools, there were still the following technical issues to be improved in terms of prototyping;

- (1) The UI simulation environments of 2D and 3D were not fully integrated. Structure of 2D UI components such as menus and icons could not be directly rendered in the UI simulation function. As a result, every UI

screen had to be rendered on-the-fly as a texture-mapped image on the 3D prototype, and huge number of UI screen snapshots had to be rendered every time the UI screen changes. This causes the simulation execution very inefficient.

- (2) Due to the texture-mapping and the limited image resolution, the appearance of the UI screen in the 3D became much degraded when the 3D model is zoomed up. It might let the test subjects feel unmotivated for the simulation-based user test.
- (3) An expensive commercial Web3D player (Virtools) was needed for the 3D UI simulation. This forced the manufacturers to make an additional investment for the prototype and hindered widespread use of the proposed technology.
- (4) So far, there is no sophisticated or commercial visual authoring tool or editor which can help UI designers easily build and modify the extended UsiXML models in a visual way.
- (5) The tools did not support simulation of touch sensitive interface which becomes very common in recent appliance UIs such as i-Pod and digital cameras.

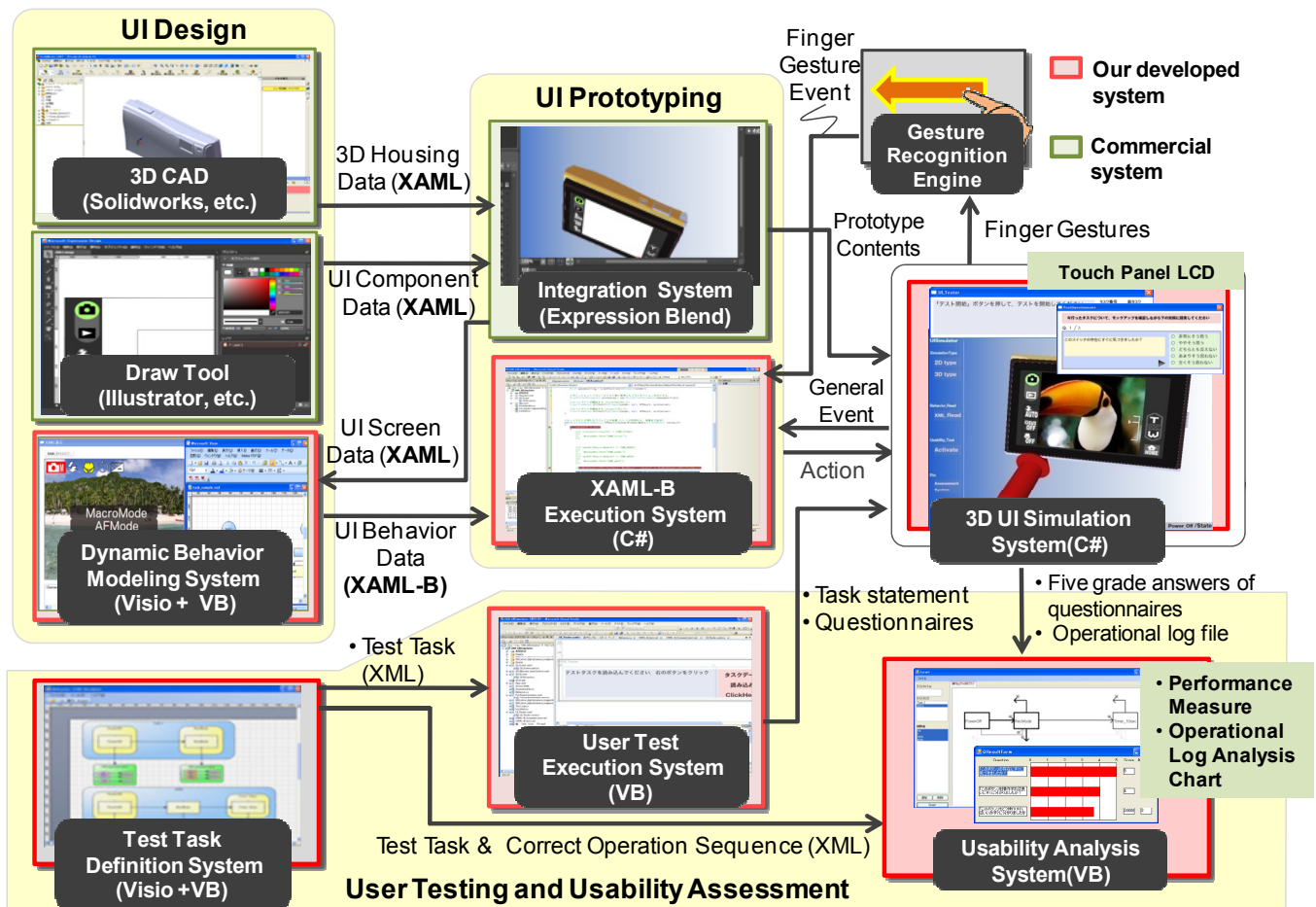


Figure 22. An overview of the XAMLS-based prototyping, user-test and usability assessment tools.

The technical features of the proposed systems are summarized as the followings:

- ## XAML-based 3D UI Prototyping

XAML (eXtensible Application Markup Language) was developed by Microsoft [18,30] as a UI specification targeted for Windows applications. XAML is an XML based mark-up language which specifies the static structure of a UI running on the WPF (Windows Presentation Foundation). WPF is a UI framework to create applications with a rich user experience, and combines applications UIs, 2D graphics and 3D graphics and multimedia into one framework. Its vector-graphic rendering engine makes the UI implementation fast, scalable and resolution independent. WPF separates the appearance of an UI from its behavior. The static structures of 2D UI screen appearances is declaratively specified in XAML, while the behavior has to be implemented in a managed programming language like C#.

**Modeling Concept of XAML-B.** We extended XAML specification so that the UI designers can declaratively define dynamic behavior only by writing a XML document

The diagram illustrates the XAML-B architecture, divided into two main models:

- State-based Model (Red Outline):**
  - State**: The root of the state-based model, containing *Source\_State* and *Destination\_State*.
  - Behavior**: Contains *Source\_State* and *Destination\_State*. It is associated with **Action** via a composition relationship (filled diamond).
  - Action**: Contains *Behavior* and is associated with **Lhs**, **Rhs**, and **MethodCall** via composition relationships (filled diamonds).
- Event-based Model (Blue Outline):**
  - Event**: Contains *Behavior* and is associated with **UIElement** via a composition relationship (filled diamond).
  - UIElement**: Contains *Event* and is associated with **RuleTerm** via a composition relationship (filled diamond).
  - RuleTerm**: Contains *RuleTerm* and is associated with **LogicalExpression** via a composition relationship (filled diamond).
  - LogicalOperator**: Contains *LogicalOperator* and is associated with **LogicalExpression** via a composition relationship (filled diamond).

Additional relationships and components:

- Lhs** and **Rhs** are associated with **RuleTerm** via a composition relationship (filled diamond).
- MethodCall** is associated with **RuleTerm** via a composition relationship (filled diamond).
- RuleTerm** and **LogicalOperator** are associated with **LogicalExpression** via a composition relationship (filled diamond).
- UIElement** is associated with **RuleTerm** via a composition relationship (filled diamond).
- LogicalExpression** is the final component in the event-based model hierarchy.

The diagram is labeled **XAML-B** at the bottom.

In these stages, two sub-models of XAML-B were respectively used to define the dynamic behavior; the state-based model for early design stage and the event-based model for detailed design stage. The UML diagram describing class structures of XAML-B is shown in Figure 23. The specification of the XAML-B includes both state-based model, event-based model and the reference to the original XAML specifications. The details of these two models are explained as the following sub-sections.

Figure 24-(a) shows an example of the state-based model in case of the power-on UI behavior in a digital camera. A set of the classes of XAML-B included inside the state-based model in Figure 23 is used. The state-based model consists of the classes of *State*, *Event*, *Behavior* and *Action*.

- 14 -



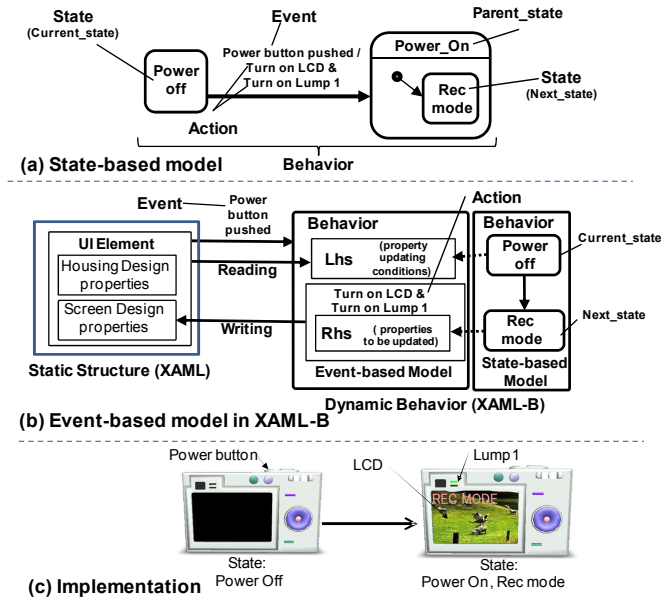


Figure 24. Examples of state-based modeling and event-based modeling of UI behaviors in XAML-B.

**Event-based model for detailed design stage.** Once the UI design enters the detailed design stage, the number of states tends to explode when using the state-based model. So, an event-based model is used at the detail design stage. The event-based model enables the designers to describe the detail control of UI components indicated on the screen whose notion is originally included in XAML specification.

Figure 24-(b) shows an example of the event-based model which is described based on the state-based model example of Figure 24-(a). The event-based model was made up on the basis of UsiXML [16, 28], which is the other XML-compliant UI description language and has declarative description of event-based dynamic behavior. A set of the classes of XAML-B included inside the event-based model in Figure 23 is used.

The event-based model consists of *Behavior*, *Lhs*, *Action* and *Event*. Notions of *Behavior*, *Action* and *Event* are the same as those in the state-based model.

An *Lhs* expresses the conditions under which each *Action* become executable. An *Action* consists of the combination of *MethodCall* and *Rhs*.

An *Rhs* specifies how the attribute values of XAML should change corresponding to the change of UI appearance.

A *MethodCall* specifies an external procedure such as an animation clip or a sound clip. A pair of an *Lhs* and an *Rhs* describes a state-transition rule.

And a *RuleTerm* expresses a condition that attribute values in XAML must fulfill before and after an *Action* occurs.

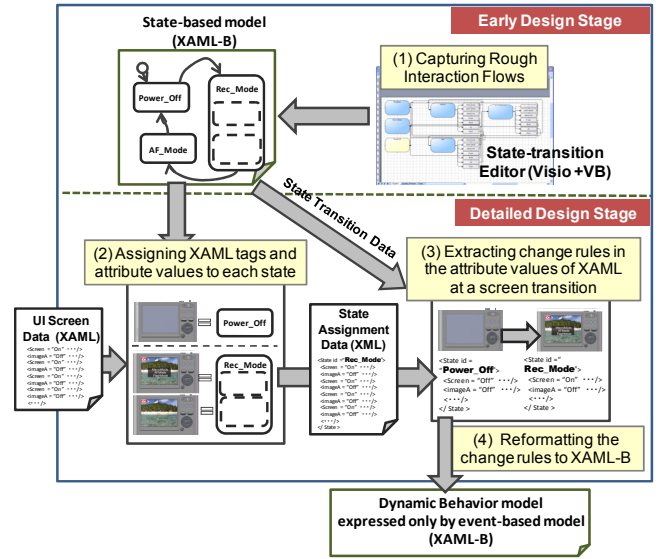


Figure 25. Dynamic behavior modeling process in the XAML-based system.

*Dynamic behavior modeling system and the modeling process*

We developed a prototype of dynamic behavior modeling system which was implemented by the combination of Visio and Visual Basic. Figure 25 shows the functions of the modeling system. It can support both of the state-based and event-based modeling processes.

The modeling flow of the system is also shown in Figure 25. In the early design stage, as in the upper part of Figure 25, rough interaction flows are initially captured as a state-based model.

In order to support efficient modeling work in this stage, a state-transition editor was implemented. In the editor, state, event and action can be graphically created and edited on the Visio by UI designer. This modeling result is stored in a XAML-B document.

In the detailed design stage, as shown in the lower part of Figure 25, first a set of tags and their attribute values of one UI screen which has been modeled as XAML document in the integration tool are assigned to one state. A new state tag is created in the XAML-B document, and a set of the XAML tags and their attribute values representing the state of the UI is packed inside the state tag.

Two different sets of these tags and the attribute values each of which is assigned to a source or a destination state are then compared to each other. Taking XOR between these tags and attribute values automatically makes the change in attribute values of an Action tag in the event-based model. Finally reformatting the rule of change to an XML document makes a final XAML-B description corresponding to this state-transition.

#### *Building process of 3D digital prototype*

According to the process described in the previous section, a 3D digital prototype is built in the following processes shown in Figure 22:

- (1) A 3D housing model is created in a commercial CAD system (Solidworks 2008) and is exported to a model integration system (Expression Blend) as a XAML document.
- (2) Each graphical component (text, icon, etc.) of the UI is defined in a draw tool (Illustrator etc.) and is exported to an integration tool as a XAML document.
- (3) In the integration tool, a set of the graphical components are combined to make one XAML document which defines each UI screen. And the 3D location of the UI screen is also specified onto the 3D housing model by this XAML document.
- (4) The XAML document is imported to a dynamic behavior modeling system. The behavior of the UI is defined by a UI designer according to the process described in 2.3, and is outputted as a XAML-B document from the system.
- (5) Finally, the XAML-B execution system reads this document and drives UI simulation on the 3D housing model responding to input events coming from the user of the digital prototype.

#### *Gesture recognition function*

Gestural interfaces become available in current information appliances with touch sensitive interfaces. So a gesture recognition function is installed in our prototyping system. Several types of user finger gestures inputted from a UI screen of a 3D digital prototype can be recognized as events, and can be processed in the XAML-B execution system.

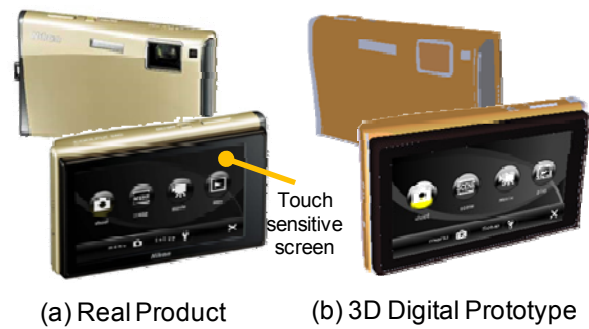
In the XAML-B specification, several gesture types that the function can recognize are described in the “*Gesture\_Type*” attribute placed inside the “*Event*” tag.

Real-time gesture recognition is needed for smooth operation of UI simulation. InkGesture engine [5] is being used in the system. Four types of finger gestures of “Leftward”, “Rightward”, “Upward” and “Downward” can be recognized as Events in our system. The recognized gesture can then be processed as one of the events in XAML-B execution system.

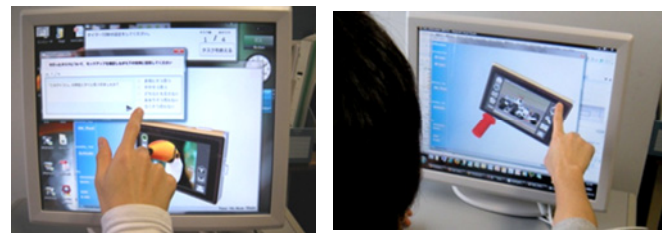
This recognition function enables the users to operate touch sensitive interfaces modeled on the 3D digital prototype using not only mouse gestures but direct finger gestures.

#### **User test and usability assessment systems in XAML-based system**

In our system, user test and usability assessment can be done on the 3D digital prototype. The test and assessment processes are as same as the ones described in the previous sections of the UsiXML-based assessment system.



**Figure 26. A digital camera and its digital prototype.**



**Figure 27. User-test situation with touch sensitive screen where a digital prototype is displayed.**

#### **A case study of user test**

##### *An example of the 3D digital prototype*

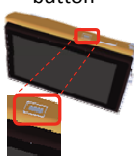


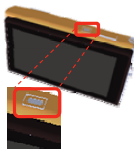
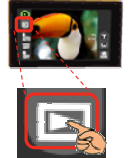

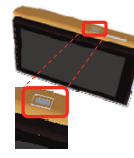


- (6) A digital camera (Nikon-S60) with a touch-sensitive screen shown in Figure 26-(a) was adopted as an example of prototyping, user test and usability assessment. As shown in Figure 26-(b), the 3D housing model of the camera was modeled in Solidworks 2008. Over 60 states and 100 state transitions were modeled in the dynamic behavior model described by XAML-B. The sound of finger touch is also emulated in the digital prototype to avoid missed-operations.

As shown in Figure 27, a 19-inch touch sensitive LCD monitor shown was used in the test, and the user can directly input by finger touch and finger gesture on the touch screen displayed as a part of the 3D digital prototype which is displayed on the touch sensitive LCD monitor. This enabled the users to operate the UI on the 3D digital prototype as realistically as the one on the real camera.

##### *User test settings*

The user test was done using the 3D digital prototype and the operational log analysis chart and five grade evaluation results were compared to those obtained from the one using the real camera. 35 subjects who had no experience of using this camera attended the test. Among them, 17 subjects operated the 3D digital prototype and 18 subjects the real camera.

Two tasks shown in Figure 28 which include the basic UI operations were given to the subjects. In task 1, they asked to set the self-timer duration for 10 sec from the power-off state. In task 2, they asked to enter the preview mode state from a power-off state and to indicate the first picture by turning over the pictures indicated on the touch screen.

	Task description	# of correct routes	Correct operation sequence and states at which digital questionnaires appear				
Task1	Set the self-timer duration for 10 sec from power-off state	1	State	Power off	Shooting	Timer Select	Timer_10s
			Correct operation	Push power button 	Push timer icon 	Push 10s icon 	
Task2	Enter preview mode from a power-off state, and display the pre-specified picture on the touch screen using finger gesture	1	State	Power off	Shooting	Play_Mode	Play_First
			Correct operation	Push power button 	Push playback icon 	Slide a finger horizontally 	
		2	State	Power off	Shooting	Home	
			Correct operation	Push power button 	Push home icon 	Push playback icon 	

**Figure 28. The test tasks and their correct operations.**

In the task 2, two correct sequences of operations exist as shown in Figure 28; the one of pushing an up-arrow or a down-arrow icon indicated at the corner of the touch screen, or the one of directly sliding the finger leftward or rightward on the touch screen.

The five grade evaluation results for the questionnaires were also obtained from the subjects who took a missed operation.

#### *Usability assessment results*

The operational log analysis charts and 5 grade evaluation results for the questionnaires at a state of missed-operation in case of the real product and the digital prototype in task 1 are shown in Figure 8.

From Figure 29, it was found that many subjects (12/18 in case of real camera, and 9/17 persons in case of digital prototype) took missed operations at the “Shooting” state. In this state, they should push the small timer icon indicated lower left of the touch screen, but they found themselves lost deep in the menu hierarchy toward incorrect states. Two missed-operation patterns in Figure 29-(a) and (b) were very similar to each other. Moreover, the results of the five grade evaluation for the questionnaire at the “Shooting” state showed that most of the missed subjects did not notice the icon to be pushed and did not understand that they should operate it. This figure also showed that

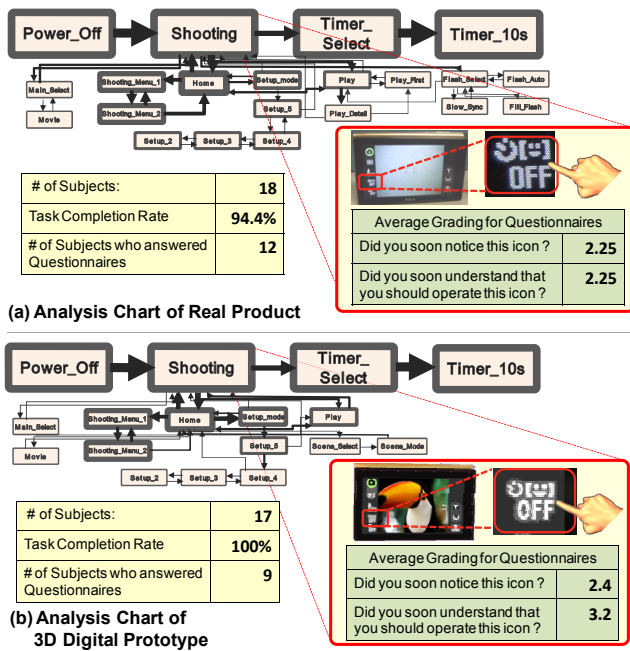
same tendencies of the evaluation results were observed both in the digital prototype and in the real camera.

While in case of the task 2, as shown in Figure 30, most of the subjects (8/10 in the real camera and in the digital prototype) did not complete the task using the finger gestures, but could it by taking alternative correct operation (pushing the icons). The five grade evaluation results for the questionnaire at the “Play\_Mode” state also showed that the subject who took this alternative operation (“Play\_Mode” -> “Play\_Detail” -> “Play First”) did not notice that the camera could accept direct finger gestures and that they should make them to complete the task.

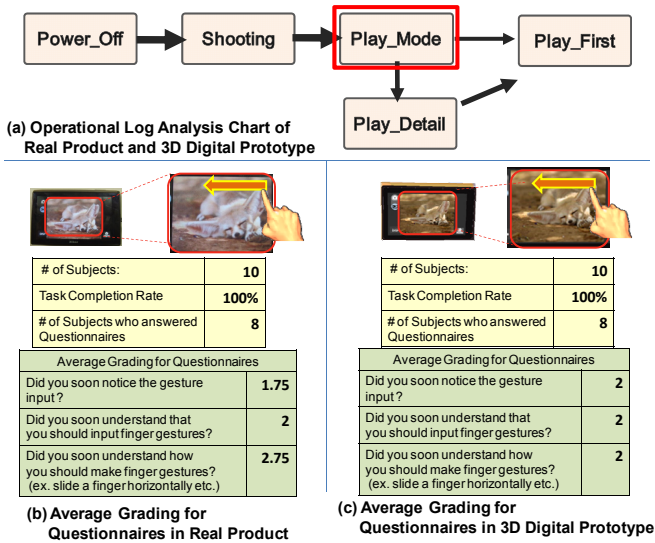
These test results showed the following clear facts:

- In the “Shooting” state and the “Play\_Mode” state, the icon indicated on the UI screen could not adequately make most users think of the correct input operations intended by the designers. So, these icons should be strongly redesigned to improve usability.
- There were strong correlations of user operation sequence and the ratings of the questionnaires between the 3D digital prototype and the real product. This suggests that the 3D digital prototype with touch sensitive interface could replace a physical prototype while keeping the ability to find usability problems from the prototype.





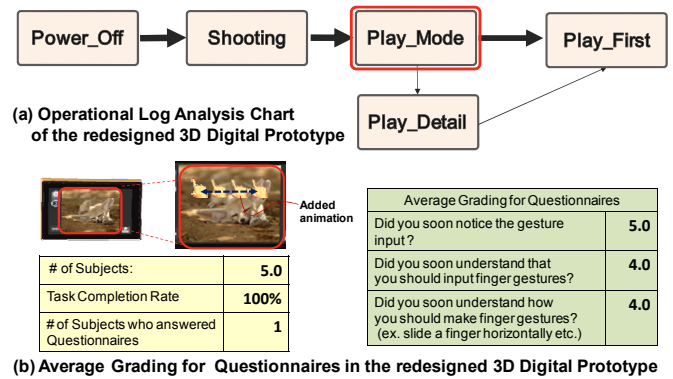
**Figure 29. The operational log analysis charts and five grade evaluation results in Task1.**



**Figure 30. The operational log analysis charts and five grade evaluation results in Task2.**

#### *The effect of UI redesign on the digital prototype*

The result of task 2 revealed that the icons indicated on the UI screen at "Play\_Mode" state could not make the users think of direct finger gesture to turn over the indicated pictures. To solve the problem, a reciprocal motion animation of a finger icon was newly added on the screen when entering this state. After this redesign, an additional user test for the task 2 was done by 7 new subjects, and the results were analyzed. Figure 31 shows the operational log analysis charts and five grade evaluation results for this new test.



**Figure 31. The operational log analysis charts and five grade evaluation results in Task2 after UI redesign.**

Six of the seven subjects could take direct finger gestures to turn over the pictures this time. Also from the grade evaluation, one subject who took a missed operation could even notice the finger gesture input at this state and actually inputted the gesture.

Only 10 minutes were needed for creating and inserting this animation file name into the original XAML-B document in the redesign. This fact showed that our proposed systems enabled UI designers to realize very rapid turnaround of design-test cycle compared to the one of physical prototypes.

#### **CONCLUSION**

UsiXML-based and XAML-based systems of prototyping, user testing and usability assessment were proposed which enabled 3D digital prototyping of the information appliances. To declaratively describe the static structure and the dynamic behavior of the user interface of the appliances with physical user interface elements, Usi-XML and XAML were originally extended, and its UI simulation system were developed. Gesture recognition function enabled the subjects to manipulate the touch-sensitive UI of the 3D digital prototype in the user test. User test execution and analysis of missed operations could be fully automated. The results of the user test and usability assessments for the digital camera showed that major usability problems appearing in real product could be fully extracted even when using the digital prototype, and that the proposed systems enabled rapid turnaround time of design-test-redesign-retest cycle.

The user test and usability assessment could be fully automated by the proposed system. But there are still open-problems to be solved in our research. The major one is whether the proposed two-stage modeling process of UI behaviors is actually understandable and accessible for most interaction designers compared to the current prototyping tools like Flash.

As a result of the development of our two XML-based computer-aided prototyping and usability assessment tools for UI, we are concluding that, so far, it is the best way to combine the UsiXML-based model-driven hierarchical de-



velopment framework with XAML-based implementation. UsiXML provides clarity in capturing and describing the UI system ranging from the conceptual design to the concrete stage in declarative way, while XAML does excellent ability and fidelity of the 2D and 3D integrated UI simulation in much inexpensive environments.

At this moment, the dynamic behavior modeling system is still a prototype phase, and “usability” of the system itself is still not fully considered and improved. Therefore our future research should include the usability evaluation on the proposed dynamic behavior modeling method and system by interaction designers themselves. Statechart-based modeling process [4] which was adopted in our research can inherently offer interaction designers an ability of state-based step-by-step hierarchical modeling process of UI. The process also enables a good compatibility of UI code generation process. Confirming the effectiveness of this concept by experiments will be included in our future research.

#### ACKNOWLEDGMENTS

We gratefully acknowledge the supports of the Grant-in-Aid for Scientific Research under the Project No.19650043 (Term: 2007-2008) and No.21360067 (Term: 2009-2011) funded by the Japan Society for the Promotion of Science, and of the Grant-in-Aid for Seeds Excavation (Term: 2009) funded by Japan Science and Technology Agency.

#### REFERENCES

1. 3D XML: [www.3ds.com/3dxml](http://www.3ds.com/3dxml).
2. Broak, J. SUS: a ‘quick and dirty’ usability scale, Usability Evaluation in Industry. Taylor and Francis (1996).
3. Hori, M. and Kato, T. A Modification of the Cognitive Walkthrough Based on an Extended Model of Human-Computer Interaction (in Japanese). *Trans. Information Processing Society Japan* 48, 3 (2007), pp. 1071–1084.
4. Horrocks, I. *Constructing the User Interface with Statecharts*. Addison-Wesley, Harlow (1999).
5. Microsoft.Ink Gesture class, [http://msdn.microsoft.com/en-us/library/microsoft.ink.gesture\(v=VS.85\).aspx](http://msdn.microsoft.com/en-us/library/microsoft.ink.gesture(v=VS.85).aspx)
6. ISO13407 Human-centred design processes for interactive systems. International Standard Organization, Geneva (1999).
7. ISO9241-11 Ergonomic requirements for office work with visual display terminals (VDTs) – Part 11: Guidance on usability. . International Standard Organization, Geneva (1998)
8. Kuutti, K., ?. Virtual prototypes in usability testing. In Proc. of the 34<sup>th</sup> Hawaii Int. Conf. on System Sciences HCCISS’2001, 5, 2001.
9. Kanai, S., Horiuchi, S., Shiroma, Y. and Kikuta, Y. Digital usability assessment for information appliances using User-Interface operable 3D digital mock-up. *Research in Interactive Design*, 2, Springer: VC\_HUCEID2006 (2006), p. 235.
10. Kanai, S., Horiuchi, S., Shiroma, Y., Yokoyama, A. and Kikuta, Y. An Integrated Environment for Testing and Assessing the Usability of Information Appliances Using Digital and Physical Mock-Ups. Lecture Notes in Computer Science, vol. 4563. Springer, Berlin (2007), pp. 478–487.
11. Kanai, S., Higuchi, T. and Kikuta Y. 3D digital prototyping and usability enhancement of information appliances based on UsiXML. *International Journal on Interactive Design and Manufacturing* 3, 3 (2009), pp. 201–222.
12. Kanai, S., Higuchi, T. and Kikuta Y. XAML-Based Usability Assessment for Prototyping Information Appliances with Touch Sensitive Interfaces. *Research in Interactive Design*, 3, Springer: PRIDE-P189, 2010.
13. Kerttula, M. and Tokkonen, T. K. Virtual design of multi-engineering electronics systems. *IEEE Computer*, 34, 11 (2001), pp. 71–79.
14. Landay, J.A.? Sketching Interfaces: Toward more human interface design. *IEEE Computer* 34, 3 (2001), pp. 56–64.
15. Lin, J., ? DENIM: Finding a tighter fit between tools and practice for web site design. In Proc. of ACM Conf. on Human Factors in Computing Systems CHI’2000. ACM Press, New York (2000), pp. 510–517.
16. Limbourg, Q. and Vanderdonckt, J. UsiXML: A User Interface Description Language Supporting Multiple Levels of Independence. In *Engineering Advanced Web Applications*, M. Matera, S. Comai, S. (Eds.). Rinton Press, Paramus (2004), pp. 325–338.
17. Cybelius Maestro, [http://www.nickom.co.jp/product\\_English.html](http://www.nickom.co.jp/product_English.html)
18. MacVittie, L.A. XAML in a Nutshell. O’Reilly Media, (2006).
19. Norman, D. A. Cognitive engineering. In *User Centered Systems Design: New Perspectives in Human-Computer Interaction*. D.A. Norman and S.W. Draper (Eds.). Lawrence Erlbaum Associates, Hillsdale (1986), pp. 31–61.
20. Park, H., Moon, H.C. and Lee, J.Y. Tangible augmented prototyping of digital handheld products. *Computers in Industry* 60, 2 (2009), pp. 114–125.
21. Protobuilder, [http://www.gaio.co.jp/product/dev\\_tools/pdt\\_protobuilder.html](http://www.gaio.co.jp/product/dev_tools/pdt_protobuilder.html).
22. RAPID PLUS, [http://www.e-sim.com/products/rapid\\_doc/index-h.htm](http://www.e-sim.com/products/rapid_doc/index-h.htm).
23. UIML, User Interface Markup Language v3.1 Draft Specification, <http://www.uiml.org/> (2004)
24. UsiXML, <http://www.usixml.org/>

25. Viewpoint, <http://www.viewpoint.com/pub/technology/>
26. VRML97, Functional specification and VRML97 External Authoring Interface (EAI). ISO/IEC 14772-1:1997 and ISO/IEC 14772-2 (2002)
27. Virtools, [www.virttools.com](http://www.virttools.com)
28. Vanderdonckt, J. A MDA-Compliant Environment for Developing User Interfaces of Information Systems. In *Proc. of 17th Conf. on Advanced Information Systems Engineering CAiSE'05* (Porto, 13-17 June, 2005). O. Pastor & J. Falcão e Cunha (Eds.). Lecture Notes in Computer Science, vol. 3520. Springer-Verlag, Berlin (2005), pp. 16-31.
29. Vanderdonckt, J. Model-Driven Engineering of User Interfaces: Promises, Successes, and Failures. In *Proc. of 5th Annual Romanian Conf. on Human-Computer Interaction ROCHI'2008* (Iasi, September 18-19, 2008), S. Buraga, I. Juvina (eds.). Matrix ROM, Bucarest (2008), pp. 1–10.
30. XML User Interface Language (XUL) 1.0, <http://www.mozilla.org/projects/xul/xul.html> (2001)
31. Extensible Application Markup Language (XAML), <http://msdn.microsoft.com/en-us/library/ms747122.aspx> (2008).

# UsiXML Extension for Avatar Simulation Interacting within Accessible Scenarios

Abel Serra, Ana Navarro, Juan Carlos Naranjo

ITACA-TSB Polytechnic University of Valencia

Edificio G8 - Camino de Vera s/n, 46022 Valencia (Spain)

{absersan, annacer, jcnaranjo}@itaca.upv.es – <http://www.tsb.upv.es>

## ABSTRACT

Nowadays the design of products which make our life more comfortable is increasing however this is not an easy task. The simulation is presented as a very good option to achieve this purpose due it allows economize design costs. For that reason, it is important implement mechanisms that allow analyzing in an early stage of the design whether a product is accessible and easy-used, especially by our target group of population (people with disabilities and elders). After investigation, UsiXML is defined as an acceptable language to represent the simulation models, however within UsiXML a lack exists when is needed to host avatar models with a structure which be capable to represent elders and users with any kind of disability (physical, cognitive or behavioral). This paper defines the proposed extension of the UsiXML language to fulfill the need before mentioned and then support both static models (Ontology or less structured information) as XML based model definition in order to achieve the representation of physical, cognitive and psychological & behavioral attributes of the human according to a disability.

## Author Keywords

Model-based approach, User Centered Design, User model, User Interface Description Language.

## General Terms

Accessibility, Human Factors, User models, Simulation.

## ACM Classification Keywords

D2.2 [Software Engineering]: Design Tools and Techniques – *Modules and interfaces; user interfaces*. D2.m [Software Engineering]: Miscellaneous – Rapid Prototyping; reusable software. H5.2 [Information Systems]: Miscellaneous H4.m [Simulation and modeling]: *Model Development*. I.6.5

## INTRODUCTION

Due to the ageing of the population, the research towards improving the quality of life of elderly and disabled people is being a far-reaching trend and it is becoming important within the research community [1]. For doing so is necessary to use techniques those increase the effectiveness of the products as well as also save costs in the process of design. At this point the Virtual Reality Simulation comes up to satisfy this premises, thanks to it the designers can evaluate the developing product in an early stage of the pro-

cess. There are human behavior simulation models which are capable to represent Human-Computer Interaction [3], however, now emerges the need of implement a model that allow represent the three main faces of the human (physical, cognitive and behavioral) [10]. In order to achieve this objective it is necessary to find a language which is capable to host all these human models.

Since the simulation will run elicitations of the three models, a common description language that allows both a dynamic and structured definition of the models as well as the instantiation of the models is therefore needed. In addition, an optimal representation language for the physical, cognitive and behavioral human facets and the tasks they perform should be:

- Extensible.
- Universal.
- Easy to transfer.
- Easy to parse.

UsiXML [7,12] is the approach selected by FP7 VERITAS [15] project that satisfies the previous premises. UsiXML provides interoperability and reusability of human data, which in turn allows extending its applicability within various research contexts. Within our scope, UsiXML (which stands for User Interface eXtensible Markup Language) is a XML-compliant markup language is structured according to four basic levels of abstractions defined by the Cameleon Reference Framework (CRF) [2] and describes the user User Interface (UI) [9] for multiple contexts of use such as Character User Interfaces (CUIs), Graphical User Interfaces (GUIs), Auditory User Interfaces, and Multimodal User Interfaces. UsiXML is considered as a User Interface Description Language (UIDL).

## MATERIALS AND METHODS

The UsiXML [12] UIDL will be used for describing the Generic Virtual User Models structure, which represent on one hand the disability and in the other the affected tasks by the disability. We will analyze this standard in order to see the potential and the limitations for describing our target models.

Developing User Interfaces (UIs) [9,14] for interactive applications can be a very tricky and complicated procedure because of the complexity and the diversity of existing development environments and the high amount of programming skills required by the developer to reach a usable UI,

i.e. markup languages (e.g., HTML), programming languages (e.g., C++ or Java), development skills for communication, skills for usability engineering. In the case of VERITAS [15], where the same UI should be developed for multiple contexts of use such as multiple categories of users (e.g., having different preferences, speaking different native languages, potentially suffering from disabilities), different computing platforms (e.g., a mobile phone, a Pocket PC, a laptop), and various working environments (e.g., stationary, mobile) the problem is even harder.

UsiXML comes to fill the gap that has been created by the fact that the available tools for creating UIs are mainly target at the developers. To this end, UsiXML is a standard that can be equally used from experienced developers, as well as, from other experts as analysts, human factors experts, designers, or novice programmers, etc., also. Non-developers can shape the UI of any new interactive application by specifying and describing it in UsiXML, without requiring any programming skills usually demanded in markup languages (e.g., HTML) and programming languages (e.g., Java or C++).

Among the **benefits** of using UsiXML is, that it supports platform independence, modality independence and device independence. Thus a UI can be described using UsiXML in a way that remains autonomous with respect to the devices used in the interactions (mouse, screen, keyboard, voice recognition system, etc.), to the various computing platforms (mobile phone, Pocket PC, Tablet PC, laptop, desktop, etc.) and of any interaction modality (graphical interaction, vocal interaction, 3D interaction, or haptics). In UsiXML project, they have enriched this, by promoting the seven  $\mu$ 7 dimensions of the UsiXML, which covers the following  **$\mu$ 7 dimensions**:

1. Multi-device
2. Multi-user
3. Multi-linguality
4. Multi-organisation
5. Multi-context
6. Multi-modality
7. Multi-platform

After analyzing the UsiXML standard, we conclude that it is a powerful language that can be used for VERITAS purposes. However we have noticed that the standard does not cover ontological description, structures and relationships amongst items, (these ontologies will represent the research performed), so it needs to be adapted or extended in order to satisfy the input received (the ontologies [9]). The extension will have to reflect the complete structure of the ontologies.

## RESULTS

The objective is to develop a UIML [5,13] / UsiXML [7,12] interaction modeling framework that will be used for the representation of the physical, cognitive and behavioral attributes of the human according to a specific disabilities.

The Physical, Cognitive and Psychological & Behavioral Abstract User Models previously developed are initially represented using OWL ontologies [6, 11]. However, the simulation platform uses a different approach, a XML representation, based on UsiXML standard. This implies that the ontologies defined need to be transformed into UsiXML schemas. The methodology here presented will allow the automatic parsing of these two representations of knowledge. In this paper we present all the process, from the analysis of the problem up to the schema implemented.

The first phase of the research was the study of the disabilities, mainly how affect to the human, taking into account which capabilities are being damaged, and in which grade. The output of this work is a set of tables which contain all the information and parameters that will be used in the second state of the process, these tables are called the Abstract User Models.

The next task is very important due the quality of this research will set the success of the study, since the data collected are the input for the next step: the extraction of the data that is seemly relevant for simulation purposes, or what it's the same, data that can be parameterized, focusing first on the quantitative information, since it can be immediately represented by the simulation engine. It will allow us to represent an avatar with the characteristics of an elderly or disabled user. This avatar should be able to perform a task according to its level of capability as close as possible to reality.

The data extracted need to be structured and ranked, that's why in the following iteration, a parameterized user model based on the Abstract User Model previously defined needs to be developed, for doing so, this interaction receives as input all the data collected in the previous stage and then, it is proceed to the design of the ontologies, which will host all parameters and ranges of values, describing all kind of the target users. Within VERITAS, this is known as the Generic Virtual User Model.

Once the ontologies are developed, the next phase of the process is the translation from the ontologies up to the schema UsiXML. The extension of the standard is performed in this task, based on the ontologies, the goal is implements the equivalent structure within the original schema (UsiXML), and following the same structure of it. This task is complicated due the lost of the information caused by the differences of the representations when the information from the ontologies wants to be expressed in XML format.

The UsiXML extension (which hosts the Generic Virtual User Model), is consisted of three main parts: Physical, Cognitive and Behavioral, all of them with its own hierarchy. The goal in this stage is achieves the optimal management of the data, since the simulator has to analyze and process it, in order to bring to the avatar the best performance during the simulation.

Following the Veritas models are presented, describing the attributes and the hierarchy implemented and the reasons for choose each structure. The relations between them are also described cause is one of the main points of the model potential.

The models are linked with the disability model with the attribute “affected by”. This parameter will establish, for each limb, which disabilities could be affected, doing this way, the information about the disabilities and its consequences are accessible within the model.

Other important link which is needed to implement is that one that relate the task model with the user model, this connection is vital due provide important data about how the disabilities affect the performance of the tasks, the model express this information at low level, since the task model consist of primitive tasks, which are grouped setting complexes tasks, such as driving. This parameter is “affected task” which takes part of the disability model. Following the details of the models are presented:

### Disability model

The disability model describes the disability details. This part is clearly relevant since it allows us to link the disability with the body features. This model embraces all the disabilities which the project is working on, all the attributes about the disability are accessible and structured in order to establish links between the disability and the tasks. This links give us valuable information about the disabilities from the point of view of the tasks. The parameter “disability” is composed by:

- *Type*: physical, cognitive or P&B.
- *Name*: the name of the disability.
- *Age related*: whether the disability appear due the ageing of the user.
- *Functional Limitations ICF Classification*: the index within the ICF scale.
- *Short description*: A brief explanation of the disability.

The parameter “disabilityDetails” contains the detailed description of the disabilities acting as literature for the simulator users. The schema described is presented in Fig. 1.

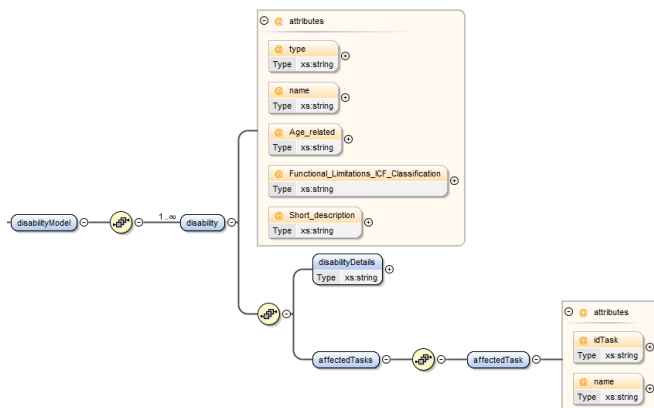


Figure 2. Disability Model.

### Physical Model

This model is the result of the study of the human’s body physiognomy, performed in the previous section. For our purpose, it is decided to structure the human body in six main parts: upper-limb, lower limb, neck, pelvis, gate and truck. These limbs are essential for representing the possible movements of a human body. With these body-limbs, basically all relevant movements of the human body can be simulated. Each limb is expressed in term of its capabilities, each of them have three main attributes: “measure units”, “minValue” and “maxValue”. In Figure 2, we can see the schema of the finger, it is composed by its abilities, such as flexion. Their attributes describe the range of capabilities and they will see modified according to the disabilities.

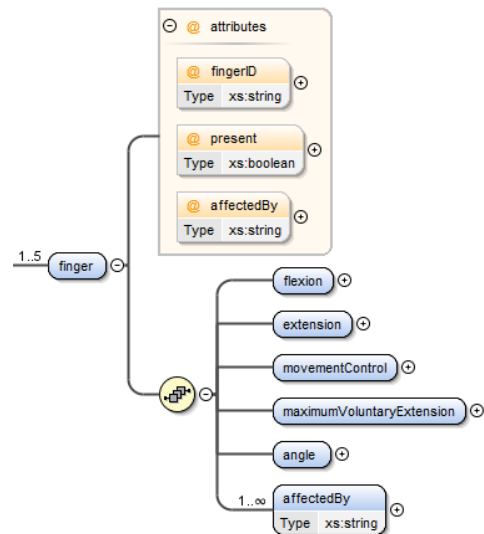
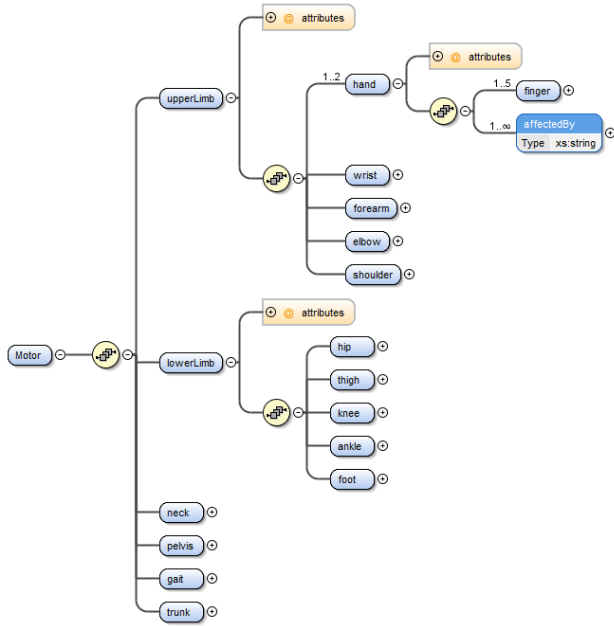


Figure 3. Schema of the finger.

The schema shows the parameter “affectedBy” which contain the relation between the human parameters and the disabilities. The schema was implemented following the point of view of the simulator, for that reason was necessary structure the body in two main parts (upperLimb and lowerLimb) and then separate the following attributes: neck, pelvis, gait and trunk, due after research it is noticed that the disabilities analyzed affect mainly to this limbs, thus this approach has the efficiency needed for the simulation engine. Figure 3 shows the schema implemented.

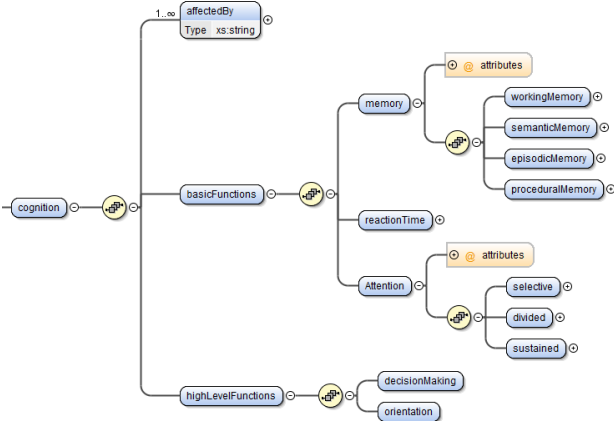
### Cognitive Model

The cognitive model represents a set of cognitive processes. This model describes the human mind processes which can be categorized into basic functions such as reaction time, working memory, procedural memory and attention (and subcategories of each one) and high level cognitive processes, such as decision making and orientation. The approach model a cognitive disability by the effects that the specific disability has on each cognitive process (how the disability affects on each function separately, for instance, a person with Alzheimer, will have the memory affected in a certain degree).



**Figure 4. Motor schema.**

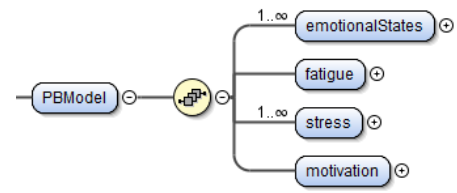
Consequently, we have followed here the same approach. In Figure 5, we can see the Cognitive model developed. The schema represents the cognitive processes before mentioned. The parameter “Affected\_by” will be use to indicate which disability affects on each cognitive attribute. Figure 4 shows the schema implemented.



**Figure 5. Cognition description.**

### Behavioral Model

The approach followed was to model a P&B state by the effects that the specific state has on the human processes (specially cognition), for instance, how the psychological state affects on each function separately, for instance, a person with stress, will have the memory affected in a certain degree). Consequently, we have followed here the same approach. The P&B UsiXML model describes the dimensions of the 4 psychological and behavioral facets selected in VERITAS: emotional states, fatigue, stress and motivation. This model allows us to link the P&B state with with the affected attributes.



**Figure 6. Physical & Behavioral Model.**

### Emotional States

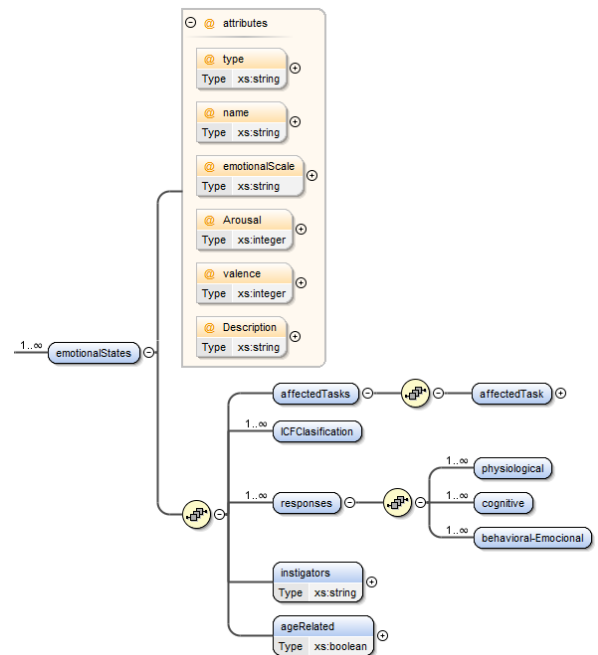
This model allows us to represent the facets of the emotions and how they affect on a user and has six attributes:

- *Type*: the type of emotion: positive and negative emotions
- *Name*: the name of the emotion: joy, sadness, fear
- *Description*: Brief description of the state.
- *Emotional Scale*: value with the emotional level that helps to indicate the type
- *Arousal*: value that shows the arousal level of the emotions
- *Valence*: value that shows the valence of the emotions

From emotional states there are several elements hanged:

- *AffectedTasks*: tasks affected by this emotional state.
- *ICFClassification*: set of functional limitations that have this state.
- *Responses*: Responses triggered by the emotional state. They can be physiological, cognitive and behavioral/emotional.
- *Instigators*: Stimuli that trigger the emotional states.
- *ageRelated*: if this state is related with age or not.

Figure 7 shows the schema with all the information within the emotional states facets.



**Figure 7. Emotional States model.**

## Stress

This model allows us to represent the facets of the stress and how they affect on a user. It has 4 main attributes:

- *Type*: the type of stress: Chronic, short term
- *Name*: the name of the stress: chronic stress, acute stress, eustress
- *Description*: Brief description of the state.
- *Level of Stress*: value with the arousal level of stress

From stress there are several elements hanged:

- *AffectedTasks*: tasks affected by this emotional state.
- *ICFClassification*: set of functional limitations that have this state.
- *Responses*: Responses triggered by stress. They can be physiological, cognitive and behavioral/emotional.
- *Stressors*: Stimuli that trigger the stress responses.
- *ageRelated*: if this state is related with age or not.

Figure 8 shows the schema with all the information of the stress dimensions.

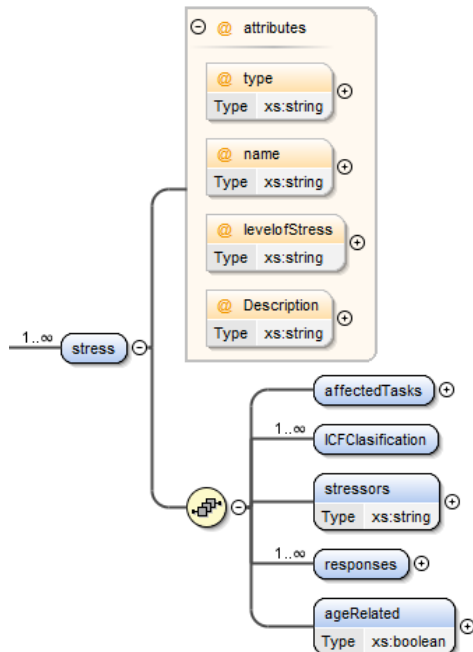


Figure 8. Schema of the Stress.

## Fatigue

This model allows us to represent the facets of fatigue and how they affect on a user. It has 4 main attributes:

- *Type*: the type of fatigue: mental, physical
- *Name*: the name of the state.
- *Description*: Brief description of the state.
- *Level of Fatigue*: value with the level of fatigue.

From fatigue there are several elements hanged:

- *AffectedTasks*: tasks affected by fatigue.
- *ICFClassification*: set of functional limitations that have this state.

- *Responses*: Responses triggered by fatigue. They can be physiological, cognitive and behavioral/emotional.
- *Causes*: Causes that trigger the fatigue responses.
- *ageRelated*: if this state is related with age or not.

Figure 9 shows the schema with all the information of the stress dimensions.

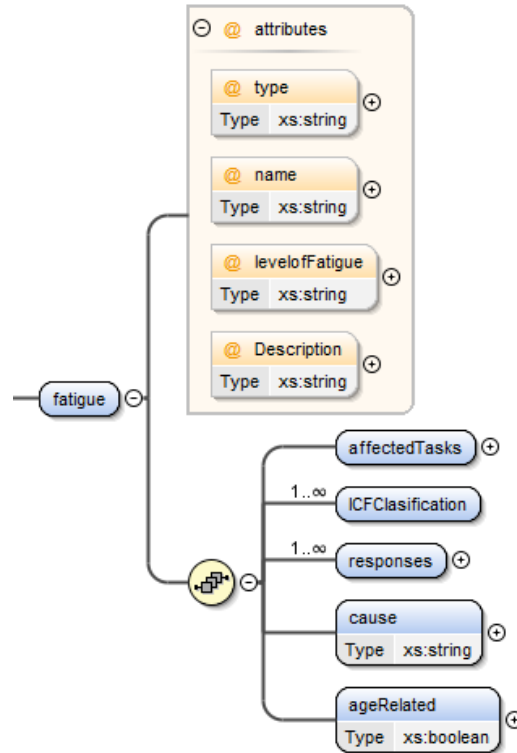


Figure 9. Fatigue Schema.

## Motivation

This model allows us to represent the facets of motivation and how they affect on a user. It has 6 main attributes:

- *Type*: the type of motivation: intrinsic, extrinsic
- *Name*: the name of the state.
- *Description*: Brief description of the state.
- *Motivation Level*: value with the level of fatigue
- *Expectancies*: internal expectancies of the user
- *Valences*: internal valences of the user.

From motivation there are several elements hanged:

- *AffectedTasks*: tasks affected by motivation.
- *ICFClassification*: set of functional limitations that have this state.
- *Responses*: Responses triggered by motivation. They can be physiological, cognitive and behavioral/emotional.
- *Causes*: Causes that trigger the motivational responses.
- *ageRelated*: if this state is related with age or not.

Figure 10 shows the schema with all the information of the motivation dimensions.



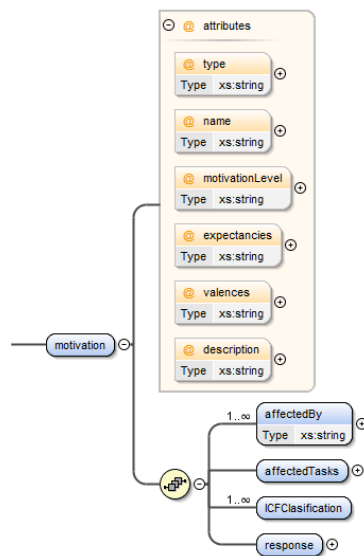


Figure 10. Motivation Schema.

## CONCLUSIONS AND FUTURE WORK

The main objective of the present work is to implement the optimal methodology for translating or mapping the Physical, Cognitive and Psychological & Behavioral Abstract User Models ontologies into consistent UsiXML models.

With this aim, we have performed the following steps: (1) an analysis of the AUMs tables and existent ontologies, (2) definition of the methodology for translating the ontologies into UsiXML schemas, (3) implementation of the Generic Virtual Model schema (extension of the UsiXML) following the methodology and (4) development of a tool that allows the automatic generation of the Generic Virtual User Models. The analysis of the AUMs has let us to extract a high number of relevant features of the different models. These features have been included in the Generic Virtual User Model, resulting in a schema that represents with fidelity the different dimensions of the human being especially regarding motor abilities, cognitive functions and behavioral and psychological aspects. This parameterization of the main physical and cognitive disabilities and psychological states is one of the most important achievements of the work. This parameterization is reflected on the Generic Virtual User Model schema. One of the main challenges of the analysis process was to select the most relevant and unequivocal attributes and parameters, since the AUMs contain a huge amount of information that cannot always be parameterized. The UsiXML has been extended to cover the needs of the Physical, Cognitive and P&B AUMs, since the generic version does not support VERITAS ontologies.

## REFERENCES

1. AAL Web-Site, <http://www.aal-europe.eu>
2. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J. A Unifying Reference Framework for Multi-Target User Interfaces. *Interacting with Computers* 15, 3 (2003), pp. 289–308.
3. Fernandez-Llatas, C., Mocholi, J., Sala, P., and Naranjo, J. Process Choreography for Human Interaction Computer-Aided Simulation Human-Computer Interaction. In *Proc. of 14<sup>th</sup> Int. Conf. on Human-Computer Interaction HCI International'2011 (Orlando, July 9-14, 2011)*. Part I, Design and Development Approaches. Lecture Notes in Computer Science, Vol. 6761. Springer, Berlin (2011), pp. 214–220.
4. Heflin, J. (Ed.). OWL Web Ontology Language Use Cases and Requirements. W3C Recommendation, Online resource <http://www.w3.org/TR/webont-req/>
5. Helms, J., Schaefer, R., Luyten, K., Vermeulen, J., Abrams, M., Coyette, A., and Vanderdonckt, J. *Human-Centered Engineering with the User Interface Markup Language*. In Seffah, A., Vanderdonckt, J., Desmarais, M. (eds.), “Human-Centered Software Engineering”, Chapter 7, HCI Series, Springer, 2009, pp. 141–173.
6. Introduction to Ontologies, OpenStructs: TechWiki, [http://techwiki.openstructs.org/index.php/Intro\\_to\\_Ontologies](http://techwiki.openstructs.org/index.php/Intro_to_Ontologies)
7. Limbourg, Q. and Vanderdonckt, J. UsiXML: A User Interface Description Language Supporting Multiple Levels of Independence. In Matera, M., Comai, S. (eds.), “Engineering Advanced Web Applications”, Rinton Press, 2004, pp. 325–338.
8. McGuinness, D.L. and van Harmelen, F. (Eds.) OWL Web Ontology Language Overview. W3C Recommendation, 10 February 2004, <http://www.w3.org/TR/2004/REC-owl-features-20040210/>. Latest version available at <http://www.w3.org/TR/owl-features/>
9. Patel-Schneider, P.F., Hayes, P., and Horrocks, I. (Eds.). OWL Web Ontology Language Semantics and Abstract Syntax. W3C Recommendation, 10 February 2004, Latest version available at <http://www.w3.org/TR/2004/REC-owl-semantics-20040210/>. <http://www.w3.org/TR/owl-semantics/>
10. Serra, A., Navarro, A., and Naranjo, J. Accessible and Assistive ICT UIML/UsiXML task modeling definition.
11. Smith, M.K., Welty, Ch., and McGuinness, D.L. (Eds.). OWL Web Ontology Language Guide. W3C Recommendation, 10 February 2004, <http://www.w3.org/TR/2004/REC-owl-guide-20040210/>. Latest version available at <http://www.w3.org/TR/owl-guide/>
12. User Interface eXtensible Markup Language (UsiXML) [www.usixml.org](http://www.usixml.org)
13. User Interface Markup Language (UIML), <http://www.uiml.org/>
14. Vanderdonckt, J. Model-Driven Engineering of User Interfaces: Promises, Successes, and Failures. In *Proc. of 5<sup>th</sup> Annual Romanian Conf. on Human-Computer Interaction ROCHI'2008* (Iasi, September 18-19, 2008), S. Buraga, I. Juvina (eds.). Matrix ROM, Bucarest (2008), pp. 1–10.
15. Veritas Consortium Web-Site: <http://veritas-project.eu>



# Support Tool for the Definition and Enactment of the UsiXML Methods

Mohamed Boukhebouze, Waldemar P. Ferreira, Neto Amanuel Koshima,  
Philippe Thiran, Vincent Englebert

PReCISE Research Center, University of Namur,  
rue Grandgagnage, 21 – B-5000 Namur (Belgium)

{mboukheb, waldemar.neto, ammanuel.koshima, philippe.thiran, vincent.engagebert}@fundp.ac.be

## ABSTRACT

In this paper, we propose a supporting tool for UsiXML-methods based on a new meta-model called SPEM4-UsiXML. This meta-model relies on the OMG standard SPEM 2.0 meta-model, which uses a UML profile to define the elements of a method. SPEM4UsiXML allows expressing the core elements of the UsiXML methods (like development path, development step, and development sub-step). In addition, the meta-model separates the operational aspect of a UsiXML method (Method Content), from the temporal aspect of a method (Process Structure). Like SPEM, there is a lack of method enactments supporting in SPEM4UsiXML. To deal with this limitation, the proposed tool allows the enactment of the UsiXML methods by transforming a SPEM4UsiXML model to a BPEL model so that the a BPEL engine can be used to execute the transformed SPEM4UsiXML models.

## Author Keywords

BPEL, Method enactment, SPEM, User Interface Description Language.

## General Terms

Theory.

## ACM Classification Keywords

D2.2 [Software Engineering]: Design Tools and Techniques – *Modules and interfaces; user interfaces*. D.2.8 [Software Engineering]: Metrics|complexity measures, performance measures. H.4 [Information Systems Applications]: Miscellaneous.

## INTRODUCTION

UsiXML (User Interface eXtensible Markup Language) is a User Interface Description Languages (UIDL) [12] that describes the User Interface (UI) independently of any computing platform [14]. This independency is achieved by relying on the CAMELEON framework, which describes the UI at four main levels of abstractions: task & domain, abstract UI, concrete UI, and final UI.

In UsiXML, the CAMELEON framework is realized by adopting a Model Driven Engineering (MDE) approach to specify a set of models representing the UI at different levels of abstraction. Besides, UsiXML uses a set of transformations to derive a UI model from another model. For ex-

ample, a high-level model (e.g. task & domain model) can be transformed into low-level analysis or design model (e.g. concrete UI model) [9]. Another example of a UsiXML transformation is the extraction of high-level model from a set of low-level models or from code [9].

According to Limbourg *et al.* [9], UsiXML transformations may be combined to form a UsiXML method. A UsiXML method, which is also called *development path* [9], is the process to follow for developing a user interface based on UsiXML models. In a UsiXML method, transformations are considered as development steps that can be decomposed into nested development sub-steps. In turn, a development sub-step realizes a basic goal assumed by a developer while constructing a UI.

To reap all the benefits, a UsiXML method needs to be designed and evaluated by describing formally its content (its semantics) and its form (its abstract/concrete syntax). For this reason, a UsiXML method needs to be compliant with a well-defined meta-model so that the core elements of UsiXML methods (e.g., development path, development step and development sub-step) can be formally defined. In addition, the enactment of UsiXML methods needs to be supported by a tool. By enactment of a UsiXML method we mean the ability of a tool to support the UsiXML models transformation according to the method specification. In order to achieve the UsiXML method enactment with a tool, the UsiXML method meta-model needs to be expressiveness to allow the execution of the UsiXML transformation.

In this paper, we propose a support tool method that allows the definition and the enactment of UsiXML methods. The definition of a UsiXML method (in this tool) is based on a SPEM meta-model [11]. SPEM is an OMG standard that provides a great usability using UML profiles. In addition, it contains generalization classes that allow the refinement of the vocabularies used to describe the concepts or the relationships between concepts. In order to support the specific key elements of the UsiXML methods (e.g. development path, development step and development sub-step), our proposed tool uses a SPEM meta-model specific to UsiXML methods. This specific meta-model is called SPEM4UsiXML.

Like SPEM, the SPEM4UsiXML meta-model allows the description of a method process structure without introducing its own formalism to precisely describe the process behavior models. [11] argues that the separation of SPEM method structure from the behavior of the method opens up the possibility to reuse existing externally-defined behavior models. A method described with the SPEM 2.0 meta-model can be enacted by mapping it to a business flow or an execution language such as BPEL [11] or XPDL [16] and then executing this representation of processes using enactment engine such as a BPEL engine [11].

In order to provide a flexible and independent transformation systems, this work implements UsiXML model transformation engine as Web services. Each Web service enacts a specific development sub-step by using associated transformation rules.

In this way, a UsiXML method can be seen as a Web services composition. Our method support tool allows the enactment of a UsiXML method by transforming a SPEM4UsiXML model to a BPEL based on a set of mapping rules and by executing it using a BPEL engine.

The rest of the paper is organized as follows. Section 2 gives an overview of UsiXML methods. Section 3 introduces the SPEM4UsiXML meta-model. Section 4 presents the transformation of a SPEM4UsiXML model to a BPEL. Section 5 demonstrates the prototype of the support tool for the UsiXML methods. Finally, the paper ends with a conclusion and future works.

#### UsiXML METHODS

In this section, the background definition for UsiXML methods is given. A UsiXML method is a process that transforms progressively the UsiXML models in order to obtain specifications that are detailed and precise enough to be rendered or transformed into code [9]. A UsiXML method is also used to synthesize abstract models from detailed models. To achieve the UsiXML transformations, different types of transformation mechanisms can be used [9]:

- *Reification* is a transformation of a high-level model into a low-level model.
- *Abstraction* is a transformation that extracts a high level model from a set of low-level models.
- *Translation* is a same level models transformation based on a context of use change. In this work, the context of use is defined as a triple of the form (U,P,E) where *E* is a possible or actual environments considered for a software system, *P* is a target platform, *U* is a user category.
- *Code generation* is a process of transforming a concrete UI model into a source code.
- *Code reverse engineering* is the inverse process of code generation.

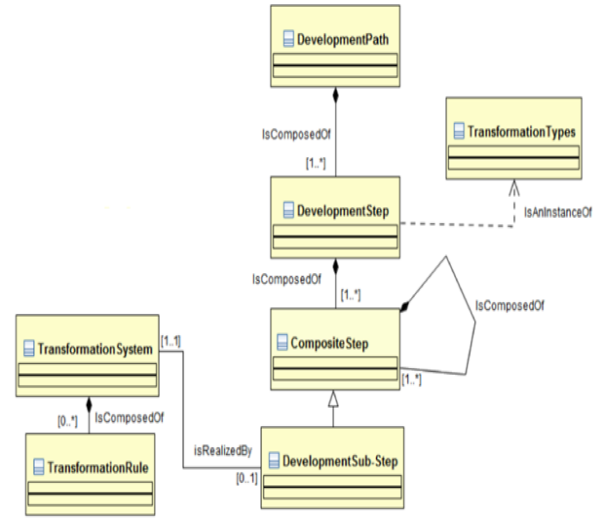


Figure 1. Transformation path, step and sub-step [9].

These different UsiXML transformation types are instantiated by development steps [9]. These development steps may be combined to form a UsiXML method. The process of combining development steps into a UsiXML method is called a development path. Several types of development paths exist, for example [9,15]:

- *Forward engineering* is a composition of reification(s) and code generation enabling a transformation of a high-level viewpoint into a lower level viewpoint.
- *Reverse engineering* is a composition of abstraction(s) and code reverse engineering, which enables a transformation of a low-level viewpoint into a higher-level viewpoint.
- *Context of use adaptation* is a composition of a translation with another type of transformation that enables a viewpoint to be adapted in order to reflect a change of context of use of a UI.

Figure 1 represents an overview of the UsiXML method meta-model. This meta-model assumes that the development steps are decomposed into nested development sub-steps. A development sub-step may consist of activities to select concrete interaction objects, navigation objects, etc. This could be realized by a transformation mechanism (e.g., graph transformation [3] and [13]) based on sets of transformation rules [13]. Composite Step is a generalization class that is used to express a development path in a tree-structure. It represents a set of development sub-steps as leaves and a development step as root of a tree.

Based on the meta-model shown in Figure 1, three major elements of the UsiXML method are considered such as work, product and producer:

- The *work* represents what must be done. It is defined in terms of development step and development sub step.

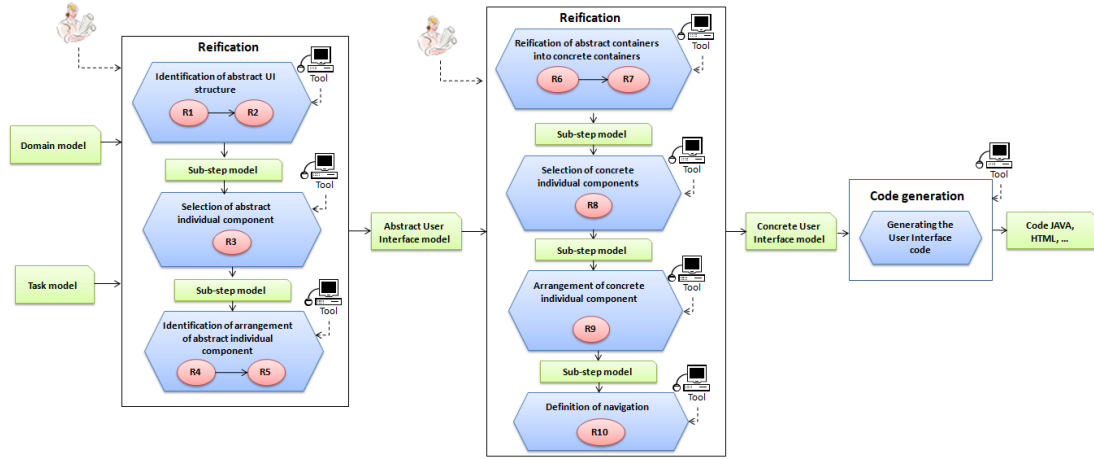


Figure 2. Forward Transformational Development of UIs.

- The *product* represents the artifact that must be manipulated by a development step and a development sub step (i.e. created, used or changed). It can concern a UI model or a UI code. In turn, a model can be a UsiXML model that is used/generated by a development step or a sub-step model that is used/generated by a development sub-step.
- The *producer* represents the agent that has the responsibility to execute a work unit. It is defined in terms of person, role, team, tool, service, etc.

Figure 2 shows an illustration of the forward engineering method. This method is fully explained in [13]. The starting point of the forward engineering is a task and a domain model (products). These models are transformed into an abstract UI based on the transformation rules specified in works. Afterwards, the abstract UI model is transformed into a concrete UI model (products). Finally, the code is generated (products). In order to achieve these transformations, a sequence of development steps (sequence of reification and code generation) needs to be performed. Each development step may involve a set of development sub-steps. For example, the first development step involves a development sub-step like identification of Abstract UI structure. This sub-step consists in the definition of groups of abstract interaction objects (an element of the abstract user interface). Each group of abstract interaction objects corresponds to a group of tasks (in task model), which are tightly coupled together. To achieve its work, the sub-step can use a sequence of rules. For example, identification of Abstract UI structure uses sequences of two rules; R1: for each leaf task of a task tree, create an Abstract Individual Elements; and R2: create an Abstract Container structure similar to the task decomposition structure. Indeed, each development step takes a UsiXML model(s) as input and transforms it to another UsiXML model(s) by involving a set of development sub-steps, which in turn manipulates sub-steps models by using a set of rules. Note that, each development step (and development sub-step) has a producer responsible of their execution. For example, the first

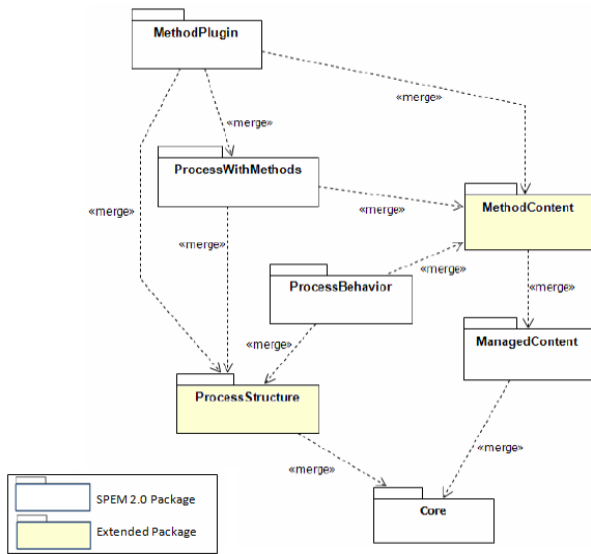
development step can have a human actor who verifies the transformation done in this step. Whereas a transformation tool can execute the rules sequence of the sub-step "identification of abstract UI structure".

In the next section, we present our proposed meta-model for the UsiXML method, SPEM4UsiXML.

#### SPEM4UsiXML

UsiXML User interface designers need to rely on robust and well defined method meta-model in order to specify the elements of a UsiXML method. In the literature, several method standard meta-models have been introduced like SPEM [11], OPEN [4] and ISO 24744 [6]. These standards describe the core elements of a method in different ways. Each standard is built on different main principles. SPEM 2.0 [11] is an OMG standard that reuses the UML diagrams to describe the elements of a method. Whereas OPEN [OPF 2005] defines an industry-standard meta-model that provides a significant detail to describe different elements of a method. However, both SPEM and OPEN standards do not support the method enactment. ISO 72444 [6] uses a dual-layer modelling to allow the method engineer to configure the enactment of the method from the meta-model level by using the Clabject and the Powerptype concepts. However, the object-oriented programming languages (like JAVA) do not support the dual-layer [5,8].

Although these standard meta-models can be adopted to describe the UsiXML methods, it is more suitable to define a specific method meta-model in order to support the specific key elements of the UsiXML methods (e.g., development path, development sub-path). For this reason, we propose in this paper a new meta-model for the UsiXML methods. The proposed meta-model is based on SPEM 2.0. This choice is justified since SPEM 2.0 provides a great usability since it is a UML profile. Moreover, SPEM 2.0 contains generalization classes that allow the refinement of the vocabularies used to describe the concepts or the relationships between concepts. These abstract generalization classes allow creating a UsiXML method meta-models specific to a certain domain (e.g., UI Development).



**Figure 3. Structure of the SPEM4UsiXML meta-model.**

The goal of the proposed meta-model, SPEM4UsiXML (SPEM for UsiXML), is to define the elements necessary for the description of any UsiXML method. The SPEM4UsiXML extends the SPEM 2.0 ([11]) by adding new classes. In addition, like SPEM, SPEM4UsiXML separates the operational aspect of a UsiXML method from the temporal aspect of a UsiXML method. This means that SPEM4UsiXML reuses the UML diagrams for the presentation of various UsiXML method concepts.

As depicted in *Figure 3*, the SPEM4UsiXML meta-model uses seven main meta-model packages inherited from SPEM: *Method Content* describes the operations aspect of a UsiXML method; *Process Structure and Process Behaviour* describes the temporal aspect of a UsiXML method, *Process With Methods* describes the link between these two aspects; *Core* provides the common classes that are used in the different packages; *Method Plug-in* describes the configuration of a UsiXML method; *Managed Content* describes the documentation of a UsiXML method.

SPEM4UsiXML extends the classes of the *Method Content* and the *Process Structures*. Indeed, SPEM4UsiXML adds new classes for the SPEM method content meta-model package in order to specify several development steps, sub sub-steps, products and producers. Moreover, SPEM4UsiXML adds new classes in the SPEM process structure package in order to specify the control flow of development steps, sub-steps, products and producers that are used in the UsiXML method process.

In this paper we focus only on the description and the enactment of the dynamic aspect of the method (i.e. method process). For this reason, we present the *Process Structure* package of SPEM4UsiXML in the next section.

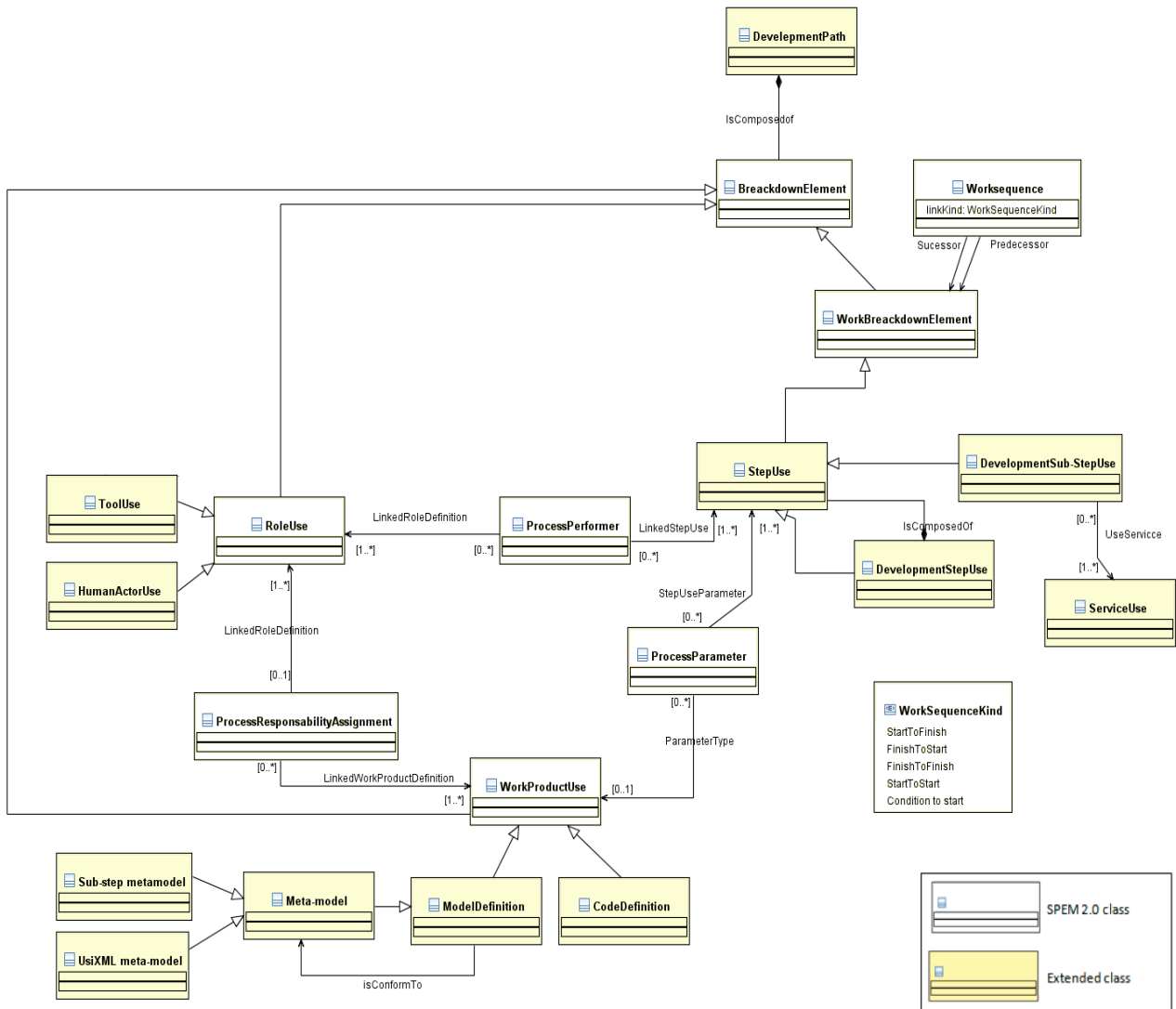
### Process Structure Package

As shown in *Figure Error! Reference source not found.*, SPEM4UsiXML adds new classes to the SPEM *Process Structure* package. The white classes represent the classes of SPEM that are not modified, whereas the yellow classes represent the classes extended by SPEM4UsiXML.

- *Development Path*: defines the properties of a UsiXML method.
- *Breakdown Element*: is a generalization class that defines a set of properties used by the element of a UsiXML method (Product, Development step and producer).
- *Work Breakdown Element*: provides specific properties for Breakdown Elements that represent a *Development Step* and a *Development Sub-Step*.
- *Step Use*: is a generalization class that defines a set of properties used by the element of the Development Step, the Composite Step and the Development Sub-Step.
- *Composite Step Use*: is a generalization class that is used to define a tree-structure with a set of development sub-step as a leaf and a development step as the root.
- *Development Step Use*: defines the transformation steps of the UsiXML method that are performed by Roles Use instances. A Development Step Use is associated to an input and an output Work Products Use.
- *Development Sub-Step Use*: defines the sub-steps of a Development Step Use. As sub-step can be achieved using an autonomous component called service (Service Use), so that the enactment of the development sub-step is independent of any transformation system.
- *Role Use*: represents a performer of a Development Step Use or a Development Sub-Step.
- *Work Product Use*: represents an input and/or output type for a Development Step. It can concern a model (Model Use) or a code (Code Use).

The SPEM4UsiXML Method process structure package contains also some useful elements inherited from SPEM 2.0 like:

- *Process Responsibility Assignment*: links Role Uses to Work Product Uses by indicating that the Role Use has a responsibility relationship with the Work Product Use.
- *Process Performer*: links Role Uses to Development Step Use by indicating that these Role Use instances participate in the work defined by the Development Step Use.
- *Work Sequence*: represents a relationship between two Work Breakdown Elements in which one Work Breakdown Elements depends on the start or finish of another Work Breakdown Elements in order to begin or end. Indeed, a Work Sequence has 5 types:



**Figure 4. SPEM4UsiXML Process Structure package. *StartToStart* expresses that a Work Breakdown Element (B) cannot start until a Work Breakdown Element (A) start;**

1. *StartToFinish* expresses that a Breakdown Element (B) cannot finish until a Work Breakdown Element (A) starts;
2. *FinishToStart* expresses that a Work Breakdown Element (B) cannot start until a Work Breakdown Element (A) finishes;
3. *FinishToFinish* expresses that a Work Breakdown Element (B) cannot finish until a Work Breakdown Element (A) finishes.
4. *ConditionToStart* expresses that a Work Breakdown Element can be started only if the condition is satisfied.

Figure 5 gives an example of a forward engineering method expressed in SPEM4UsiXML. In this method, various development steps are represented by dashed rectangles. Each development step can be composed by a set of devel-

opment sub-steps. Development sub-steps are represented by pentagon (e.g. identification of an abstract UI structure, etc.) The development steps (and the development sub-steps) can be assigned to a producer who has a responsibility to execute or control an execution of the different development (sub)steps.

This UsiXML method needs to be enacted by a tool in order to allow supporting the transformation of the UsiXML models according to the method specification. However, the SPEM4UsiXML method meta-model provides a high level description, which is not precise enough to allow the execution of the UsiXML transformation. For this reason, the SPEM4UsiXML process needs to be mapped to an execution language. In the next section, we detail the mapping of SPEM4UsiXML process to a BPEL process.



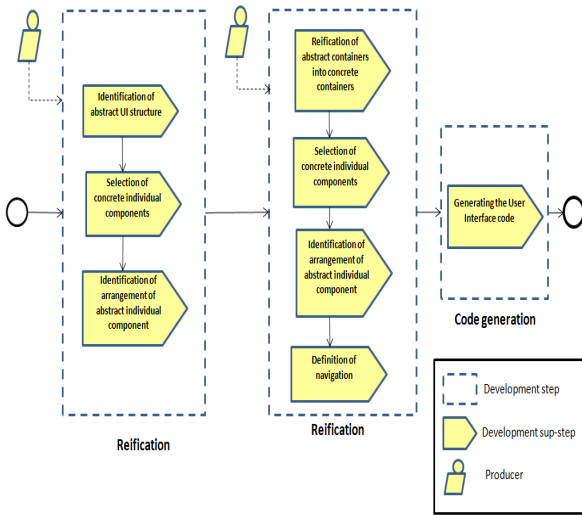


Figure 5. UsiXML Forward Engineering method expressed in SPEM4UsiXML.

#### UsiXML method Enactment

SPEM4UsiXML process package allows the description of a method process structure, but it does not introduce the formalism for enacting a method process. It rather proposes to reuse an existing externally-defined enactment model such as BPEL. For this reason, in the next section, we detail how we can map SPEM4UsiXML process to a BPEL process. The separation of SPEM4UsiXML (like SPEM) method process structure from the behavior of the method process opens up the possibility to utilize enactment machines for many different kinds of behavior modeling approaches [11]. The motivation behind this separation is to give a method design options to choose process behavior models that fits his/her needs.

Although, the separation provides a flexible way to represent the behavioral aspects of SPEM processes, it does not define the mapping rules to link the elements of SPEM process with the behavioral models. In the literature, several initiatives have been conducted to define mapping rules that allow automatically generating a specific executable model from a SPEM process [17] and [2]. For example, Feng *et al.* [17] propose a set of well-defined mapping rules to transform a SPEM process to a workflow expressed in XPD [16]. Another example is the work proposed by Bendraou *et al.* [2], which introduces transformation rules into BPEL.

Because SPEM4UsiXML extends SPEM with additional classes that specify elements of a UsiXML method (e.g. development steps and sub sub-steps), a set of mapping rules should be defined in order to link the elements of SPEM4UsiXML process with the OASIS standard BPEL. Indeed, a UsiXML process can be considered as a Web service composition orchestration where each Web service enacts a specific development sub-step transformation so that the transformation will be flexible and independent to any transformation system. As a result, an enactment ma-

chine for BPEL models can be used to run a UsiXML method. In light of this, we propose a set of mapping rules between a subset of SPEM4UsiXML concepts and the BPEL language in Table 1.

#### UsiXML method Support tool

This section describes the UsiXML support tool that is dedicated to define and enact a UsiXML method. The tool is developed as an Eclipse plug-in that includes a SPEM4UsiXML model editor as well as a SPEM4UsiXML-to-BEPEL transformer engine. Figure 6 shows a screenshot of the SPEM4UsiXML model editor that is build based on the Eclipse Graphical Modeling Framework (GMF) [10]. This framework provides a generative component and a runtime infrastructure for developing graphical editors based on a well-defined meta-model.

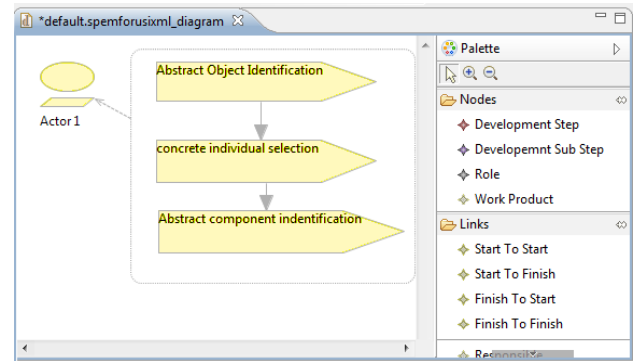


Figure 6. The SPEM4UsiXML model editor.

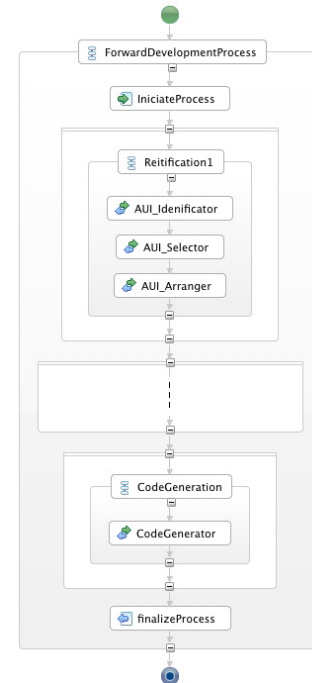


Figure 7. The BPEL Process of the UsiXML forward engineering method.

	SPEM4USiXML	BPEL	Description
Concept	Development Path	Process	A development process in SPEM4USiXML can be mapped to process in BPEL.
	Development Sub	Scope Activity	Development step is a block which is composed of one or more development sub-steps. It can be mapped to Scope in BPEL.
	Development Sub-steps	Invoke Activity	A development sub-step is a concrete step where a service(s) is invoked, hence, it can be mapped to invoke activity in BPEL.
	Role	Partner Links	A role is an actor who executes an action(s). A role could be mapped to a parent link in BPEL.
	Product	Product	Products of SPEM4USiXML are models and source codes which can be represented using variables in BPEL
Relation-ship	Start to Start	Flow Activity with Links	In order to start development step A, development step B must start first. This relationship can be expressed using flows
	Start to Finish	Flow Activity with Links	Development step A needs to start before development step B finishes its activity. This relation could also be expressed using flow and links.
	Finish to Start	Sequence Activity	A sequence represents the sequences of execution of development sub-steps. It can be mapped to a sequence in BPEL.
	Finish to Start	Flow Activity with Links	This relationship can also be expressed using flow and links to specify development step A needs to be finished so as to B finish its activity.
	Condition to Start	If Activity	Only the subsequent activities that the condition is true are started. This relationship can be expressed as an If

**Table 1. Mapping from SMEP4USiXML to BPEL.**

The UsiXML support tool is based on an ATL transformation language to specify the mapping between a SPEM4UsiXML method and a BPEL process. The mapping rules are described and executed using the ATL toolkit. The ATL toolkit [7] is a model transformation tool that allows generating a target model from a source model based on mapping rules. Figure 7 illustrates the generated BPEL process for the UsiXML forward engineering method that was explained above.

## CONCLUSION AND DISCUSSION

In this paper, we proposed a support tool for the definition and the enactment of the UsiXML methods. The tool is based on a new meta-model for UsiXML method description, called SPEM4UsiXML. This meta-model is based on the OMG standard, SPEM 2.0, which uses a UML profile to define elements of a method. The core element of the SPEM4UsiXML is the development steps that are instances of transformation types. Development steps are decomposed into development sub-steps. A development sub-step can be executed by using a Web service. SPEM4UsiXML separates the operational aspect of a method (Method Content), from the temporal aspect of a methodology (Process Structure). This allows using any modeling language to describe the process behavior like BPEL. Unfortunately, the SPEM4UsiXML meta-model cannot support the enactment of a UsiXML method on a specific endeavor. To deal with this limit, the proposed support tool (for UsiXML methods)

transforms a SPEM4UsiXML model to a BPEL process so that a UsiXML method is considered as a Web service composition where each Web service enacts a specific development sub-step of the method. Consequently, a BPEL engine can be used to execute the SPEM4UsiXML models. However, BPEL language expresses a UsiXML method process in a fully automated way meaning that a human producer is not able to interact with the development sub-steps until the end of the process execution. For example, a human producer is not able to monitor the input to a development sub-step at runtime, s/he cannot cancel the process execution or s/he is not able to execute a development sub-step. For this reason, in the future work, we plan to address this problem by extending BPEL with set of human interactions points in order to allow a human producer to interact with the method execution. This extension should allow the generation of a user interface for the UsiXML method in order to help the human producer to interact with the method at runtime. In addition, in the future, we also plan to develop a monitoring tool that allows to control the enactment of the SPEM4UsiXML methods based a historic model. This historic model keeps trace of enactment operations whenever they occur so that problems in a method can be identified and corrected based on predefined patterns (e.g. a delay in the execution of a step).

## ACKNOWLEDGMENTS

The second author would like to acknowledge of the ITEA2-Call3-2008026 UsiXML (User Interface extensible Markup Language) European project and its support by Région Wallonne DGO6.

## REFERENCES

1. Alves, A., Arkin, A., Askary, S., Barreto, C., Bloch, B., Curbera, F., Ford, M., Golland, Y., Guhzar, A., Kartha, N., Liu, C.K., Khalaf, R., Koenig, D., Marin, M., Mehta, V., Thatte, S., Rijn, D., Yendluri, P., and Yiu, A. *Web services business process execution language*, version 2.0 (OASIS standard). WS-BPEL TC OASIS, <http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html> (2007)
2. Bendraou, R., Combemale, B., Crégut, X., and Gervais, M.P. *Definition of an executable SPEM 2.0*. IEEE Computer Society (2007)
3. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J. A Unifying Reference Framework for Multi-Target User Interfaces. *Interacting with Computers* 15, 3 (2003), pp. 289–308.
4. Consortium OPEN (2010), <http://www.open.org.au/>
5. Gutheil, M., Kennel, B., Atkinson, C. A systematic approach to connectors in a multi-level modeling environment. In *Proc. of 11<sup>th</sup> Int. Conf. on Model Driven Engineering Languages and Systems MoDELS'2008* (Toulouse, September 28 - October 3, 2008). Czarnecki, K., Ober, I., Bruel, J.M., Uhl, A., Völter, M. (eds.). Lecture Notes in Computer Science, vol. 5301. Springer, Berlin (2008), pp. 843–857.
6. International Standard Organization, International Electrotechnical Commission, I.O.: ISO/IEC 24744. Software Engineering - Metamodel for Development Methodologies, JTC 1/SC 7 (2007).
7. Jouault, F. And Kurtev, I. Transforming models with ATL. In *Proc. of Satellite Events at the MoDELS 2005 Conference*, Bruel, J.M. (Ed.), Lecture Notes in Computer Science, vol. 3844, pp. 128–138. Springer Berlin / Heidelberg (2006),
8. Kuehne, T. and Schreiber, D. Can programming be liberated from the two-level style: multi-level programming with DeepJava. In *Proc. of the 22<sup>nd</sup> Annual ACM SIGPLAN Conference on Object-oriented programming systems and applications OOPSLA'2007 (October 2007)*. ACM Press, New York (2007), pp. 229–244.
9. Limbourg, Q. and Vanderdonckt, J. Multipath transformational development of user interfaces with graph transformations. In *Human-Centered Software Engineering*, Seffah, A., Vanderdonckt, J., Desmarais, M.C. (Eds.). Human-Computer Interaction Series, Springer, London (2009), pp. 107–138.
10. Moore, B. and Ebrary, I. Eclipse development using the graphical editing framework and the eclipse modeling framework. IBM, International Technical Support Organization (2004)
11. Software Systems Process Engineering Meta-Model Specification version 2.0 (2008). Object Management Group Document Number: formal/08-04-02. Standard document, <http://www.omg.org/spec/SPEM/2.0/PDF>
12. Souchon, N. and Vanderdonckt, J. A Review of XML-Compliant User Interface Description Languages. In *Proc. of 10<sup>th</sup> Int. Conf. on Design, Specification, and Verification of Interactive Systems DSV-IS'2003* (Madeira, 4-6 June 2003). J. Jorge, N.J. Nunes, J. Cunha (Eds.). Lecture Notes in Computer Science, vol. 2844, Springer-Verlag, Berlin (2003), pp. 377–391.
13. Stanculescu, A., Limbourg, Q., Vanderdonckt, J., Michotte, B., and Montero, F. A Transformational Approach for Multimodal Web User Interfaces based on UsiXML. In *Proc. of 7<sup>th</sup> Int. Conf. on Multimodal Interfaces ICMI'2005* (Trento, 4-6 October 2005). ACM Press, New York (2005), pp. 259-266.
14. UsiXML v1.8 Reference Manual, Université catholique de Louvain (February 2007).
15. Vanderdonckt, J. Model-Driven Engineering of User Interfaces: Promises, Successes, and Failures. In *Proc. of 5<sup>th</sup> Annual Romanian Conf. on Human-Computer Interaction ROCHI'2008* (Iasi, September 18-19, 2008), S. Buraga, I. Juvina (Eds.). Matrix ROM, Bucarest (2008), pp. 1–10.
16. WfMC, Workflow management coalition workflow standard: Workflow process definition interface – XML process definition language (XPDL). Technical Report WfMC-TC-1025. Workflow Management Coalition, Lighthouse Point (2002).
17. Yuan, F., Li, M., and Wan, Z. SEM2XPDL: Towards SPEM model enactment. In *Software Engineering Research and Practice*, Arabnia, H.R., Reza, H. (Eds.). CSREA Press (2006), pp. 240–245.

# Towards Methodological Guidance for User Interface Development Life Cycle

Francisco Javier Cano Muñoz<sup>1</sup>, Jean Vanderdonckt<sup>2</sup>

<sup>1</sup>Prodevelop S.L., 46001 Valencia, Spain – fjcano@prodevelop.es

<sup>2</sup>Université catholique de Louvain, Louvain School of Management

Louvain Interaction Laboratory, Place des Doyens, 1 – B-1348 Louvain-la-Neuve (Belgium)

jean.vanderdonckt@uclouvain.be – Phone: +32 10 478525

## ABSTRACT

This paper describes how methodological guidance could be provided to user interfaces designers throughout user interface development life cycle supported when a model-driven engineering is involved. For this purpose, a methodologist firstly creates a dashboard model according to a corresponding meta-model in order to define a user interface development path that consists of a series of development tasks (that structure the development path into development actions) and dependencies (that serve as methodological milestones). Once created, a user interface designer enacts a previously defined user interface development path by instantiating and interpreting a dashboard model while being provided with methodological guidance to conduct this development path. This guidance consists of steps, sub-steps, cheat sheets, and methodological actions.

## Author Keywords

Dashboard model, dependency, development path, method enactment, method engineering, methodological guidance, user interface development life cycle.

## General Terms

Design, Experimentation, Human Factors, Verification.

## ACM Classification Keywords

D2.2 [Software Engineering]: Design Tools and Techniques – *Modules and interfaces; user interfaces*. D2.m [Software Engineering]: Miscellaneous – Rapid Prototyping; reusable software. H5.2 [Information interfaces and presentation]: User Interfaces – *Prototyping*.

## INTRODUCTION

When User Interface (UI) designers, modelers, analysts, graphical designers, and developers are given the opportunity to rely on a Model-Driven Engineering (MDE) software environment to produce a UI, they often complain on the lack of methodological guidance throughout the *UI development life cycle* (UIDLC) provided by MDE:

- Although MDE explicitly relies on a structured transformation process, namely involving model-to-model transformation (M2M) and model-to-code compilation (M2C), designers do not easily perceive where some degree of freedom suggests alternative choices in the UIDLC and where some degree of determinism constraints these choices. MDE is often considered as a

straightforward approach, if not sequential, where little or no degree of freedom is offered, even when multiple *development paths* are possible [15].

- Facing the multiplicity of models (e.g., task, domain, abstract UI, concrete UI, context) in a particular development path (e.g., forward engineering, reverse engineering, round-trip engineering), the designer is rarely provided with some guidance on when and how to produce such models [13].
- When a particular step in the UIDLC should be conducted, designers do not determine easily which software should be used for this purpose, especially when different software support the same step, partially or totally. When a particular software is selected, they often feel lost in identifying the right actions to execute in order to achieve the step in the UIDLC [1].
- The multiplicity of development paths conducted among or within various organizations, in particular software development companies [3], increases the feeling of applying a UIDLC that remains not explicitly supported and that requires extensive training to become effective and efficient.
- Although several standardization efforts (e.g., the international standard for describing the method of selecting, implementing and monitoring the software development life cycle is ISO 1220707) and official organizations promote the usage of *process models* in order to increase the productivity of the development life cycle and the quality of the resulting software, they do not often rely on an explicit definition and usage of a method in these process models.

The above observations suggest that MDE is often more driven by the software intended to support it, less by the models involved in the UIDLC, and even less by a method that is explicitly defined to help UI designers. Lao Tch'ai Tche, an old Chinese philosopher, expressed the need for method in the following terms: some consider it noble to have a method; other consider it noble not to have a method; not to have a method is bad; but to stop entirely at any method is worse still; one should at first observe rules severely, then change them in an intelligent way; the aim of possessing method is to seem finally as if one had no method.

Therefore, we believe that a UIDLC according to MDE should rest on three pillars in a balanced way: *models* that capture the various UI abstractions required to produce a UI, a *method* that defines a methodological approach in order to proceed and ensure an appropriate UIDLC, and a *software* support that explicitly supports applying the method.

For this purpose, the remainder of this paper is structured as follows: Section 2 presents a characterization of these three pillars in order to report on some initial pioneering work conducted in the area of UI method engineering with the particular emphasis of methodological support. Section 3 introduces the dashboard model as a mean to define a method that may consist of one or many development paths by defining its semantics and syntax. Section 4 describes how a method could be enacted, i.e. how a development path can now be applied for a particular UI project by interpreting the dashboard model. Section 5 provides a qualitative analysis of the potential benefits of using this dashboard model for method engineering in the UIDLC. Section 6 discusses some avenues of this work and presents some conclusion.

## RELATED WORK

In general in computer science, a Software Development Life Cycle (SDLC) is the structure imposed on the software development by a development method. Synonyms include software development and software process. Similarly, in the field of UI, a UI development life cycle (UIDLC) consists of the development path(s) defined by a UI development method in order to develop a UI (Fig. 1). Representative examples of include: the Rational Unified Process (RUP) or the Microsoft Solution Framework (MSF).

Each development path is recursively decomposed into a variety of development steps that take place during the development path. Each step uses one or several models (e.g., task, domain, and context) and may be supported by some software. All pieces of software, taken together support the development method.

For instance, the development path "Forward engineering" may be decomposed in to a series of development steps: building a task model, building a domain model, building a context model, linking them, producing a UI model from these model, then generate code according to M2C. *Method engineering* [1] is the field of defining such development methods so that a method is submitted to *method configuration* [9] when executed.

The meta-method Method For Method Configuration (MMC) [9] and the Computer-Aided Method Engineering (CAME) tool MC Sandbox [10] have been developed to support method configuration. One integral part of the MMC is the method component construct as a way to achieve effective and efficient decomposition of a method into paths and paths into steps and sub-steps and explain the rationale that exist behind this decomposition. Method

engineering has already been applied to various domains of computer science such as, but not limited to: information systems [1], collaborative applications [12], and complex systems [6]. Typically, method engineering is based on a meta-model [7,8] and could give rise to various adaptations, such as situational method engineering [6] and method engineering coupled to activity theory [10]. In Human-Computer Interaction (HCI), we are not aware of any significant research and development on applying method engineering to the problem of engineering interactive systems. Several HCI development methods do exist and are well defined, such as a task-based development method [16], method-user-centered design [9], activity theory [10], but they are not expressed according to method engineering techniques, so they do not benefit from its potential advantages.

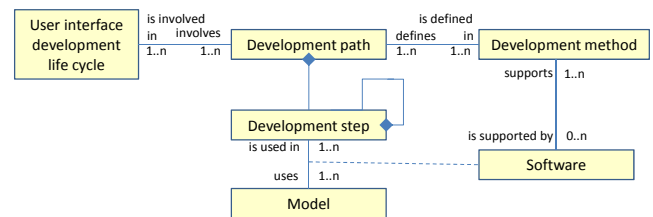


Figure 1. Structure of a UI development life cycle.

Probably the first one to address method engineering in HCI was the MIDAS (Managing Interface Design via Agendas/Scenarios) [11] environment. In this software, a methodologist was able to define a method by its different paths that could be followed and the steps required for achieving each path. MIDAS was able to show at any time when a method is executed, what are the different paths possible (e.g., design alternatives, criteria) by looking at design intentions stored in a library. MIDAS is tailored to the HUMANOID environment [11] and does not rely on a meta-model for defining a method and to execute. But it was a real methodological help.

User Interface Description Languages (UIDLs) do not possess any methodological guidance based on method engineering because they mostly concentrate on the definition and the usage of their corresponding syntaxes and less on the definition of the method [4].

MUICSER [5] provides mechanisms for expressing scenarios with personas based on storyboards. As such, they also used models through storyboard software. Therefore, it could also benefit from the potential advantages of method engineering. A more recent effort used Service Oriented Architecture (SOA) to define and enact a method, but there was no real software for achieving the method engineering.

In conclusion, very few works exist on applying method engineering to HCI, but several existing work could benefit from it.



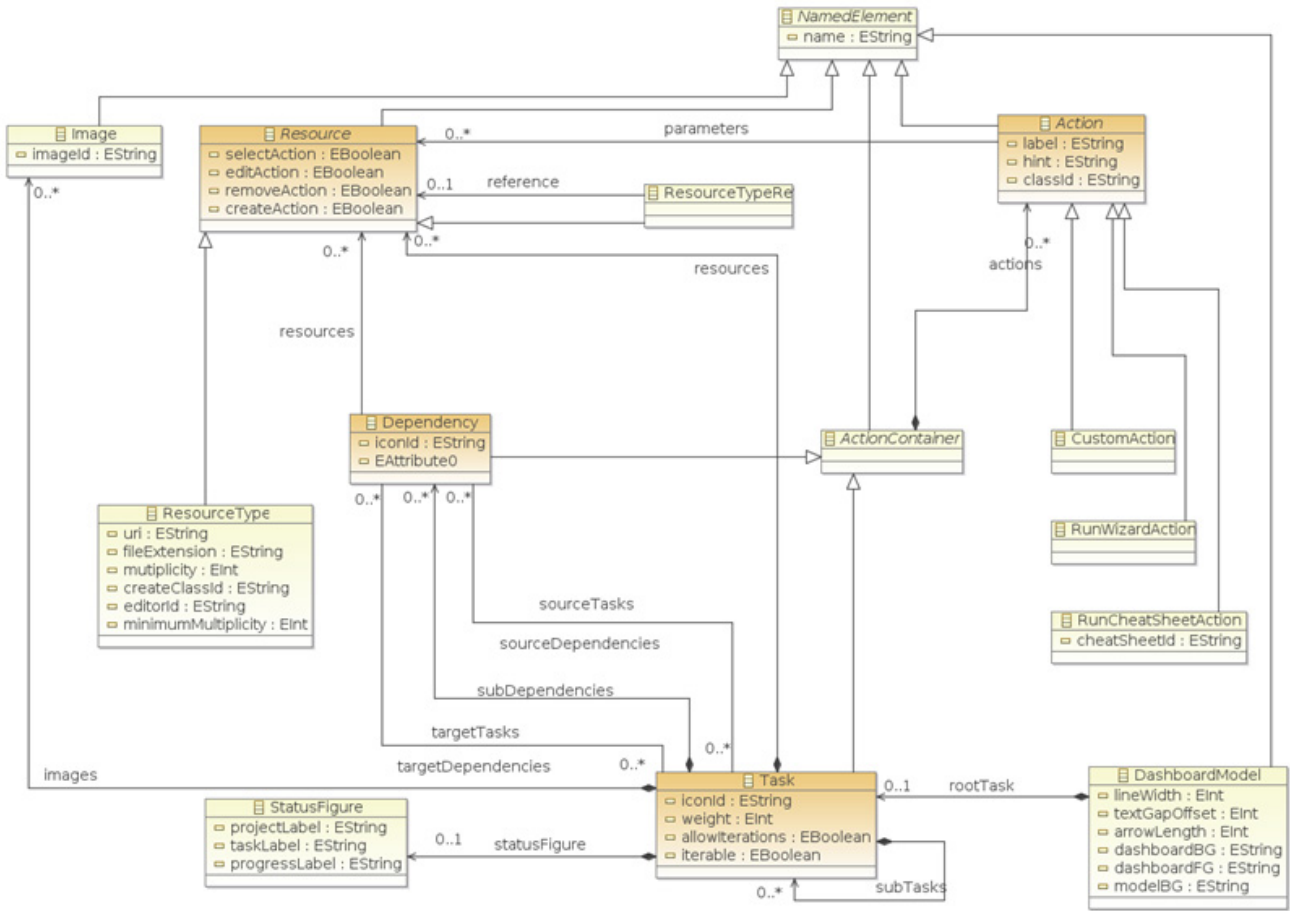


Figure 2. The Moskitt meta-model for a methodological dashboard.

#### A Meta-model for a Methodological Dashboard

To adhere to method engineering principles, a meta-model is defined [8] that addresses its methodological concepts as outlined in Fig. 2. The dashboard is based on a metamodel that allows the description of development steps via their decomposition in Tasks, Resources required in Tasks and Dependencies between Tasks. This Dashboard metamodel has been expressed using Ecore/Eclipse Modelling Framework (EMF) and implemented in the MOSKitt environment [14]. The complete metamodel can be found in this repository URL open to the public at <http://subversion.moskitt.org/gvcase-gvmetrica/dashboard/trunk/es.cv.gvcase.mdt.dashboard/model/>. The main entities, i.e. Task, Resource, Dependency and Action, are structured as follows:

**NamedElement:** consists of a common ancestor for all metamodel elements. With the experience of the definition of several metamodels (more than 10) in the MOSKitt environment we have found very useful to have a common ancestor element that all other elements in the metamodel inherit from. It simplifies several tasks in the following steps in the MDE approach we follow, such as allowing to identify whether any given element belongs to this metamodel by checking its ancestry, and providing several properties we

need in all elements of our metamodel, such as the 'name' property.

- name: EString → this element's name.

**DashboardModel:** represents a complete development path and at the same time is the root element of the metamodel. It holds the visual configuration to be used in the interpreter/enactment view.

- lineWidth: EInt → border elements' width to be used when the dashboard model is shown in the interpreter/enactment view.
- textGapOffset: EInt → gap between text when the dashboard model is shown in the interpreter/enactment view.
- arrowLength: EInt → length of arrows when the dashboard model is shown in the interpreter/enactment view.
- dashboardBG: EString → background color to be used when the dashboard model is shown in the interpreter/enactment view.
- dashboardFG: EString → foreground color to be used when the dashboard model is shown in the interpreter/enactment view.

- **modelBG**: EString → background color to be used in the interpreter/enactment view for tasks and dependencies.
- **RootTask**: Task [0..1] → reference to the whole process that is represented as a task.

**Task**: represents one development step of the development path. A Task will always be bounded by Dependencies, except for the Tasks involving the first and last steps of the process. A Task can produce or consume zero or many Resources. As an ActionContainer, a Task can perform Actions on selected Resources.

- **iconId**: EString → identifier of the icon that represents this Task when seen in the interpreter/enactment view.
- **weight**: EInt → this Task's weight in the development path (Process) in absolute value.
- **allowIterations**: EBoolean → indicates whether this Task allows more than one iteration of it to be created.
- **iterable**: EBoolean → indicates whether this Task can be iterated.
- **subTasks**: Task [0..\*] → Tasks representing the different steps to perform this Task.
- **resources**: Resource [0..\*] → Resources produced or consumed by this Task.
- **sourceDependencies**: Dependency [0..\*] → Dependencies that must be clean before this Task can be started.
- **targetDependencies**: Dependency [0..1] → Dependencies that can be cleaned when this Task is complete.
- **subDependencies** [0..1] → Dependencies that are contained in this Task due to it having more than one internal development sub-steps.
- **statusFigure**: StatusFigure [0..1] → figure to be shown in the interpreter/enactment view regarding the advancement of this Task.
- **Images**: Image [0..\*] → images contained in this Task to be shown in the interpreter/enactment view.

**Dependency**: represents a milestone in the development path, which means that a series of development steps should be achieved before proceeding to the next development step. The Milestone is here introduced as a straightforward mechanism to synchronize different types of development steps, whatever their purpose is. Each Dependency is a step in the development path (Process) that forces the preceding Tasks to synchronize. A Dependency can require zero or more Resources from previous Tasks to be completed. As an ActionContainer, a Dependency can perform one or more Actions on selected Resources.

- **resources**: Resource [0..\*] → Resources this Dependency will produce or consume. Usually but not necessarily these Resources will be ResourceTypeRefs referencing Resources from previous Tasks.

- **sourceTasks**: Task [0..\*] → preceding Tasks that use this Dependency as their Milestone.
- **targetTasks**: Task [0..\*] → Tasks that need this Dependency to be cleaned before they can be performed.

**Resource**: consists of a material or immaterial entity, produced or consumed by a Task or a Dependency of this development path (Process). Resources represent from model definition files to metamodel, document to PDF files.

- **selectAction**: EBoolean → indicates whether the 'select' action is available for this Resource in the ResourceManagement dialog in the interpreter/enactment view.
- **editAction**: EBoolean → indicates whether the 'edit' action is available for this Resource in the ResourceManagement dialog in the interpreter/enactment view.
- **removeAction**: EBoolean → indicates whether the 'remove' action is available for this Resource in the ResourceManagement dialog in the interpreter/enactment view.
- **createAction**: EBoolean → indicates whether the 'create' action is available for this Resource in the ResourceManagement dialog in the interpreter/enactment view.

**ResourceType**: represents the actual type of the Resource element, as in the kind of model file or file document format.

- **uri**: EString → URI pointing to the real resource.
- **fileExtension**: EString → extension of the files this ResourceType represents.
- **multiplicity**: EInt → maximum multiplicity of real resources this Resource can hold.
- **minimumMultiplicity**: EInt → minimum multiplicity of real resources this Resource can hold.
- **createClassId**: EString → identifier of the factory that will allow the creation of this kind of resource via the ResourceManager in the interpreter/enactment view.
- **editorId**: EString → identifier of the editor that allows the edition of this kind of resource via the ResourceManager in the interpreter/enactment view.

**ResourceTypeRef**: represents a reference to a Resource. It is to all effects equal to a Resource except that the ResourceManager in the interpreter/enactment view does not allow its modification in any way.

- **reference**: Resource [0..1] → referenced Resource.

**Action**: represents an action to be performed by the user when enacting the process. An Action can range from launching a transformation to opening a cheatsheet to visiting a web page.

- **label:** EString → human readable label to be shown in the interpreted/enactment view.
- **hint:** Estring → additional information to be given to the Action as a parameter, such as an identifier, a message, etc.
- **classId:** Estring → identifier of the factory that allows the creation and execution of this Action.

**ActionContainer:** represents any element in the metamodel that can hold and perform Actions.

**CustomAction:** represents a custom Action allows the methodologist to specify uncommon Actions with an external specification of the Action.

**RunWizardAction:** expresses a specialized Action that runs the wizard specified by the hint parameter of the Action.

**RunCheatSheetAction:** expresses specialized Action that shows the user a guide or cheatsheet.

- **cheatSheetId:** EString → identifier of the cheatsheet to show.

**StatusFigure:** allows selecting an image to be shown in the interpreter/enactment view which provides the level of progress in the process.

**Image:** represents a decorative image that will be shown in the interpreter/enactment view.

## Method Definition and Enactment

In order to define a UI development method based on one or many UI development paths (e.g., simplified, enhanced forward engineering, forward engineering with loops) as defined in Fig. 1, the person who is responsible for defining such a methodology has to create one Dashboard model based on the meta-model outlined in Fig. 2. A Dashboard model therefore represents the definition of a particular development path, but may also contain several development paths in one model thanks to the concept of milestone.

A milestone consists of a synchronization points between tasks (e.g., development steps) involved in a development path and is attached to a *synchronization condition*. Such a condition governs the contribution of each task to the milestone (e.g., AND, OR, XOR, NOT,  $n$  iterations). Once the synchronization condition is satisfied, the milestone is considered to be achieved and the development path can proceed to the next task (development step).

Fig. 3 depicts in Moskitt how a Dashboard model is created for the development path “Forward Engineering” that consists of the following development steps (that are represented as tasks to achieve to complete the development step):

- *Create Task Model:* this task is aimed at creating a task model that is compliant with the task meta-model, whatever the task meta-model would be. This task manipulates three resources:

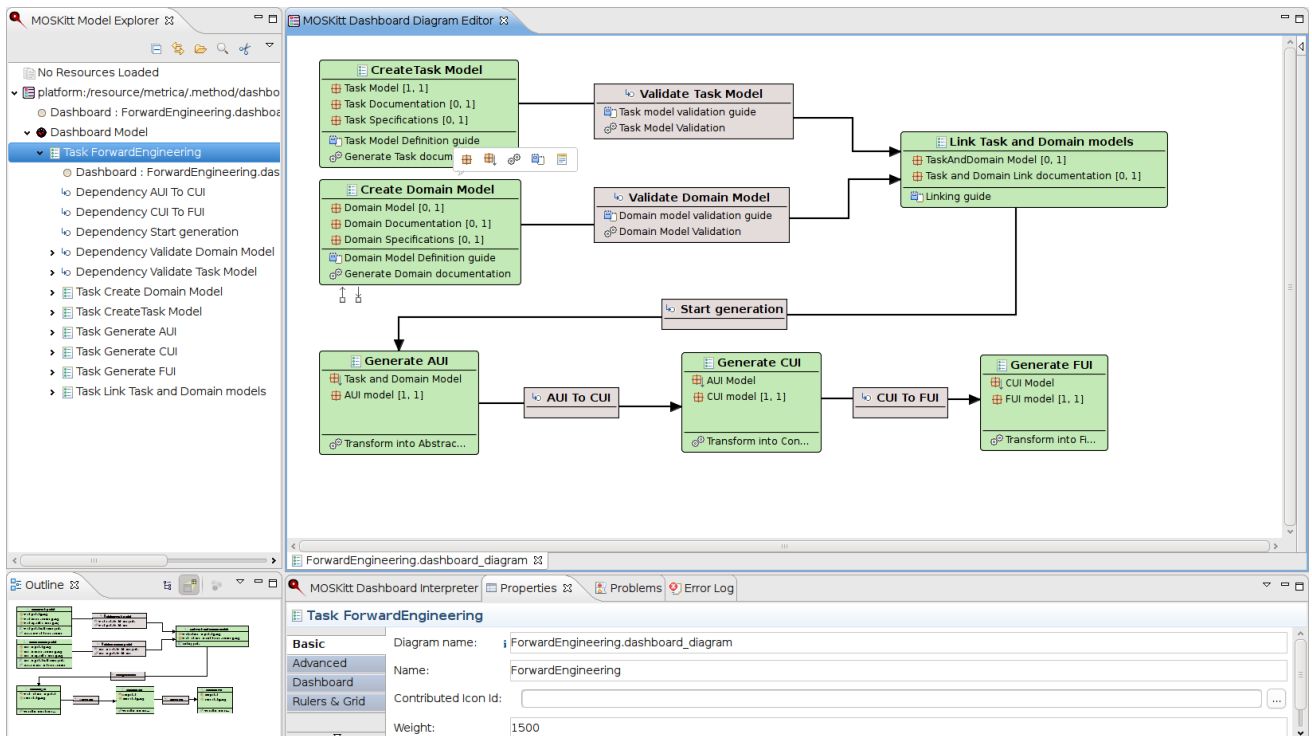
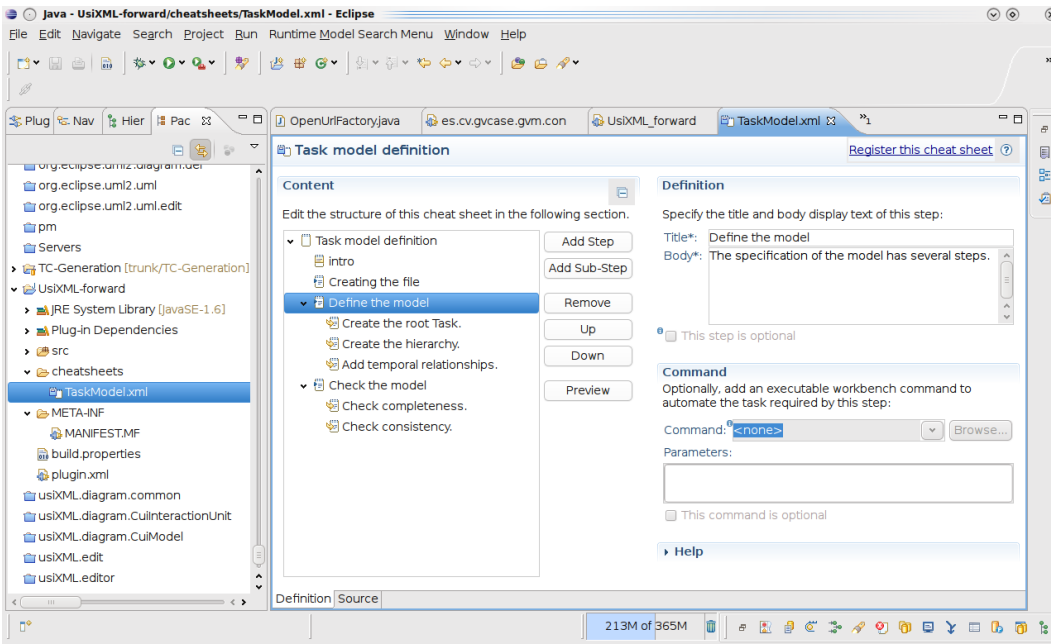


Figure 3. The Dashboard model for the “Forward engineering” development path.



**Figure 4. Cheatsheet for providing methodological guidance on task model definition.**

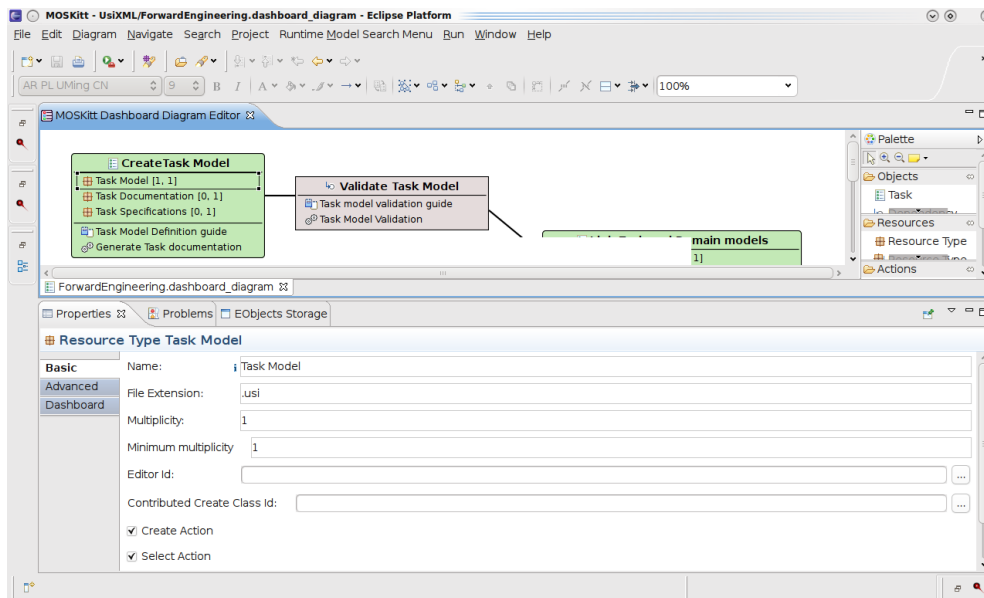
1. One and only one task model that will result from this task.
  2. An optional document containing a documentation of the task modeled.
  3. An optional set of task formal specifications.
- A “task model definition guide” is a cheatsheet provided for giving methodological guidance on how to define a task model. Fig. 4 details some potential development steps and sub-steps for this purpose in a cheatsheet. A *cheatsheet* is hereby referred to as a methodological panel that is provided from the methodologist to the method applier with any rules, heuristics, principles, algorithms, or guidelines that are helpful for achieving the associated task (here, creating a task model that is correct, complete, and consistent). An action “Generate Task Documentation” is added in order to specify a task model would ultimately result from it. For this task, Fig. 5 shows the relevant parameters, including the file extension of the file to be created. In this case, we could specify one file extension (e.g., “.usi” for a UsiXML compliant file [15]) or “.CTT” for a task model expressed according to the ConcurTaskTree notation). In this way, when the method will be enacted, the Dashboard will automatically launch the model editor corresponding to the file extension by default or selected by the method applier (Fig. 6).
- *Validate Task Model*: once the task model has been created, its validity with respect to its corresponding task meta-model is checked by means of Eclipse model checking techniques. Therefore, only one action is trig-

gered: “Task model validation”. Note that this task serves as a milestone: the method applier cannot proceed with the next tasks if the synchronization condition (here the availability of a valid task model) is not satisfied.

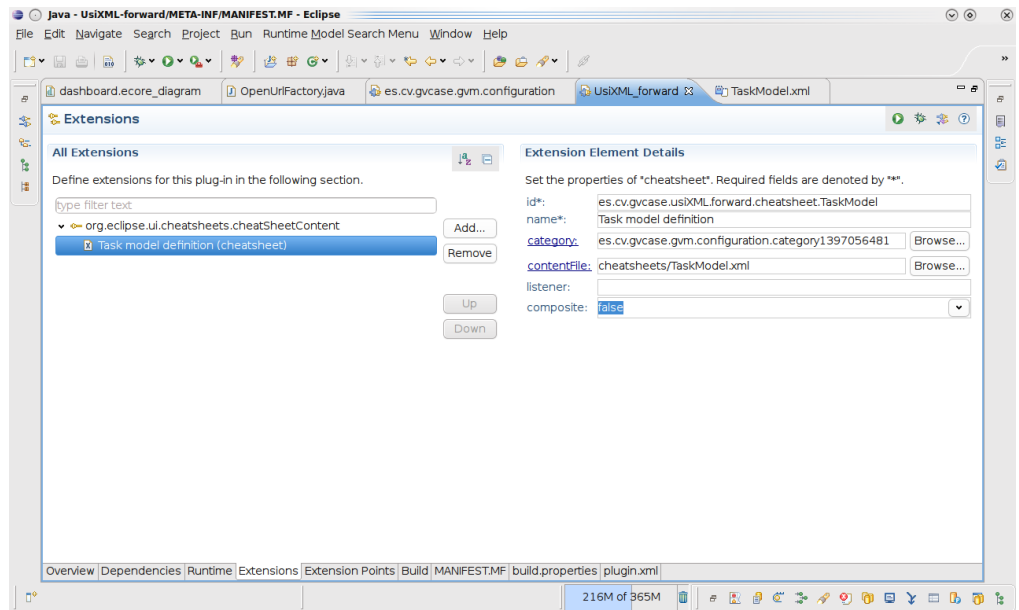
- *Create Domain Model*: this task is aimed at creating a domain model that is compliant with the task meta-model, whatever the task meta-model would be. It contains three resources, one cheatsheet and one action that are similar to those introduced for the task model.
- *Validate Domain Model*: once the domain model has been created, its validity with respect to its corresponding domain meta-model is checked by means of Eclipse model checking techniques.
- *Link Task and Domain models*: this task is aimed at establishing a link from the nodes of a task model to the appropriate nodes of a domain model thanks to the set of mappings accepted between these two models (e.g., a task observes a domain class, a task supports input/output of a set of attributes taken from different classes, a task triggers a method belonging to a class). Note that there is a dependency between this task and the two previous ones in order to ensure that the linking will be applied on two syntactically valid task and domain models.
- *Milestone: start the Abstract UI generation*: when the task model has been linked to a domain model, we have all the elements in order to initiate a generation of an Abstract UI [15]. Again, this serves as a milestone.
- *Generate AUI*: this task is aimed at (semi-)automatically generating an Abstract UI (AUI). For this purpose, an input resource “Task and domain models linked” (coming from the previous milestone) will result into an output resource “AUI model” by means of the action “Transform into AUI”. This action is related to a set of transformation rules that are automatically applied to the input resource in order to obtain the output resource. Again, the embedded transformation engine in Eclipse could be used for this purpose or a custom transformation engine could be specified based on a file extension. In this definition, only one set of transformations

is defined, but several alternative sets of transformation rules could be considered, thus leaving the control to the method applier by selecting at run-time which set to apply. Furthermore, this action is related to a transformation step (here, a M2M), but it could also be attached to an external algorithm that is programmed in a software. When all these alternatives coexist, a cheatsheet could be added to help the method applier in selecting an appropriate technique for ensuring this action (e.g., a transformation or an external algorithm) and parameters that are associated to this actin (e.g., a particular transformation set).

- *Milestone “AUI to CUI”*: this milestone serves as a synchronization point for initiating the next development step through the task required for this purpose.
- *Generate CUI*: this task is similar to the “Generate AUI” except that a CUI is produced instead of a AUI, but with parameters that govern the CUI generation.
- *Milestone “CUI to FUI”*: this milestone serves for initiating the last step.



**Figure 5. The file extension associated to a resource in the Dashboard model.**



**Figure 6. Definition of file extension and associated software, e.g., through a cheatsheet.**

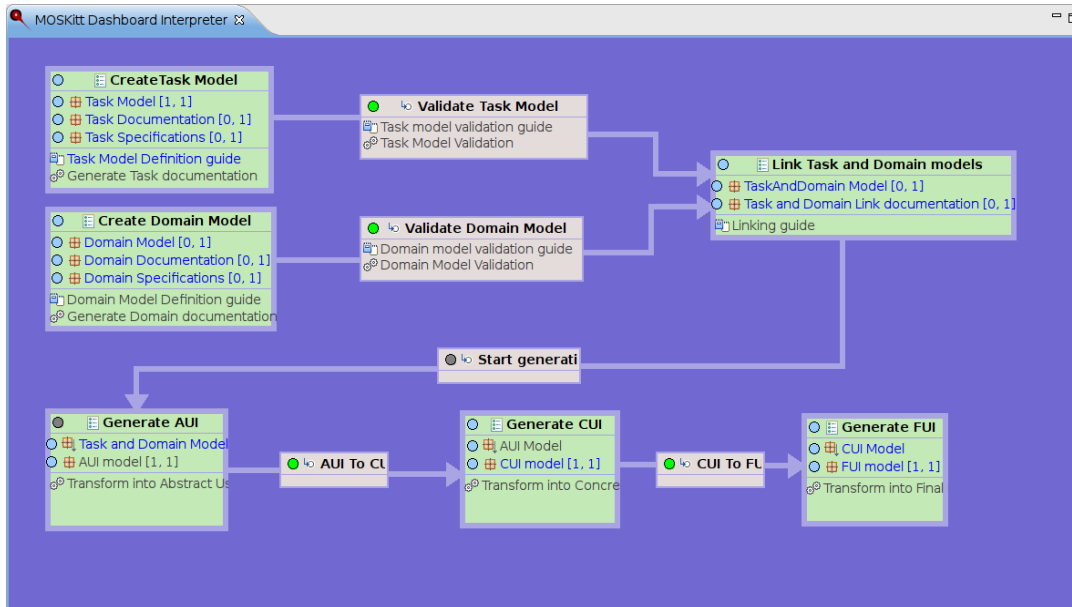
- *Generate FUI*: this task is aimed at transforming the CUI resulting from the previous task into code of the Final UI (FUI) by means of M2C transformation. Again, we may want to specify here that the transformation could be achieved by code generation or by interpretation of the CUI model produced. In the first case, a code generator is executed while a FUI interpreter renders the CUI into a FUI in the second case. Again, one default interpreter could be specified or the method applier can pick another one from a list of potential interpreters or rendering engines.

It is important to state that the **dashboard model is independent of any method, any meta-model and any User Interface Description Language (UIDL)**. It could be used for defining any UIDLC (such as [9,12,16] to name a few), any meta-model of a model involved in such a UIDLC, and any UIDL (see [4] for a list of some representative examples). The only requirement is that each model should be explicitly linked to

its corresponding meta-model in order to check its validity and conformity with respect to the meta-model as it is typically the case in MDA. Transformations gathered in trans-



formation steps should satisfy the same requirement, unless they are executed outside the Eclipse platform. The advantage of this approach is that all models and transformations between are defined by their corresponding meta-models in Eclipse, but forces to define them beforehand. Once one or several development paths of UI development method have been defined in a dashboard model, the method can be *enacted* [2] by instantiating the dashboard model. This instantiation results into a run-time representation of the Dashboard (Fig. 7) that depicts the progression of tasks already achieved, future and pending tasks, all with their associated resources. For instance, if a task requires to output resources to be created, this task will only be considered finished when the corresponding actions will have been able to produce the required resources. The method enactment is then under the responsibility of the person who is in charge of applying the method defined, e.g. an analyst, a designer. In the next section, we review potential benefits brought by the MDA approach under the light of this dashboard approach.



**Figure 7. Definition of file extension and associated software, e.g., through a cheatsheet.**

### Qualitative Evaluation

Usually, potential benefits that can be expected from MDE can be summarized as:

#### 1. Benefits resulting from the existence of a design phase:

- *Reducing the gap between requirements and implementation*: A design phase aims to ensure in advance that the implementation really addresses the customer's and user's requirements. The dashboard largely contributes to this by explicitly defining the output of a development task that should serve as an input for a

next development task, whatever the development steps are.

- *Stakeholder coordination*: Previous planning of the UIDLC enables the stakeholders (e.g., designers, developers, testers) to coordinate their work e.g. by dividing the system into several parts and defining mappings between them.
- *Well-structured systems*: A design phase provides explicit planning of the system architecture and the overall code structure. This facilitates implementation itself as well as maintenance.

#### 2. Benefits resulting from the use of visual abstract models:

- *Planning on adequate level of abstraction*: Modeling languages provide the developer concepts for planning and reasoning about the developed system on an adequate level of abstraction. The Dashboard is based on a meta-model (Fig. 2).
- *Improved communication by visual models*: The visu-

al character of modeling languages can lead to increased usability (understanding, perceiving, exploring, etc.) of design documents for both author and other developers. The Dashboard model is itself entirely visual, which is particularly important for

representing the progression of the method enactment.

- *Validation*: (Semi-)Formal modeling languages enable automatic validation of the design.
- *Documentation*: Models can be used as documentation when maintaining the system.
- *Platform-independence*: Platform-independent models can be reused or at least serve as starting point when implementing the system for a different platform. This includes development platforms like a programming language or component model, as well as deployment platforms like the operating system or target devices. As said, the dashboard based approach is independent of any method, model and UIDL, thus supporting any UIDLC in principle.

3. Benefits resulting from code generation: the benefits associated to this appear when the method is enacted.

- *Enhanced productivity*: Generating code from a given model requires often only a teeny part of time compared to manual mapping into code.
- *Expert knowledge can be put into the code generator*: Expert knowledge – e.g. on code structuring, code optimizations, or platform-specific features – can once be put into the code generator and then be reused by all developers.
- *Reduction of errors*: Automatic mapping prevents from manual errors.

4. Meta goals:

- *Easier creation and maintenance of development support*: the dashboard (rein)forces the method applicator to stick to the initial definition of the method. Therefore, if any deviation with respect to this definition should be recorded, it should be introduced as an exception of the method enactment.
- *Knowledge about creation of modeling languages*: MDE concepts and definitions reflect existing knowledge about modeling, modeling languages, and code generation. The dashboard does not escape from this since it is itself based on a meta-model that could be instantiated at any time.
- *Frameworks and tools*: Tools like Eclipse Modeling Tools provide sophisticated support for all steps in MDE like creating and processing metamodels, creating modeling editors, and defining and executing transformations. The Dashboard is implemented in the Moskitt environment [14] that is itself on top of Eclipse.

5. Maintenance of modeling language and transformations:

- *Systematic and explicit definition of metamodels and transformations facilitates*
- *Maintenance of modeling languages and code generators*: modeling language, associated model-to-model transformations and model-to-code compilations can be maintained at a level of expressiveness that is higher than a traditional programming or markup language.
- *Reuse of metamodels and transformations*: MDE compliant explicit metamodels and transformations can be understood and reused by others.

On the one hand, the method defined in a dashboard can provide substantive and effective knowledge, such as methodological guidance, on how to enact the method.

But on the other hand, once this method is defined, it is almost impossible to break the frontiers imposed by this method. The drawback of this is that any deviation triggers a round-trip engineering problem: the method that was enacted imposes modifying its corresponding dashboard model and propagating the changes in a new enactment of the new method, which is a hard procedure. This could be perceived as a burden by stakeholders who could feel that they are forced to enter in the dashboard everything that is required to properly conduct the method. But this information is intrinsically expressed in a consistent format that could give rise to a library of method definitions.

The dashboard approach has been applied to different real-world case studies, including a large-scale project by the Conselleria de Infraestructuras y Transporte (Valencia, Spain, Figure 8).

## CONCLUSION

In this paper, we presented the dashboard model as a way to support the method engineering of a user interface development life cycle. For this purpose, we first defined what such a development life cycle is and how to structure it according to the principles of method engineering [1,2].

This development life cycle is then expressed in terms of the following concepts: one or several development steps are defined in one single dashboard in order to create one development method, a development (sub-)step becomes a task to be achieved in the dashboard, the models involved in a development step become resources to be created and consumed by a task in the dashboard, the software required to manipulate these models become associated to resources via their associated file extension and/or from a list of potential software (e.g., model editor, model validator, model checker, transformation engine).

The next step of this research will consider the forthcoming ISO 24744 standard on method engineering [2] that defines a set of concepts that support the definition and the enactment of a method based on well-defined concepts along with a graphical notation that combines structural aspects (e.g., how a task is decomposed into sub-tasks) and temporal aspects (e.g., how tasks are related to each other through dependencies and constraints).

## ACKNOWLEDGMENTS

The second author would like to acknowledge of the ITEA2-Call3-2008026 UsiXML (User Interface extensible Markup Language) European project and its support by Région Wallonne DGO6.

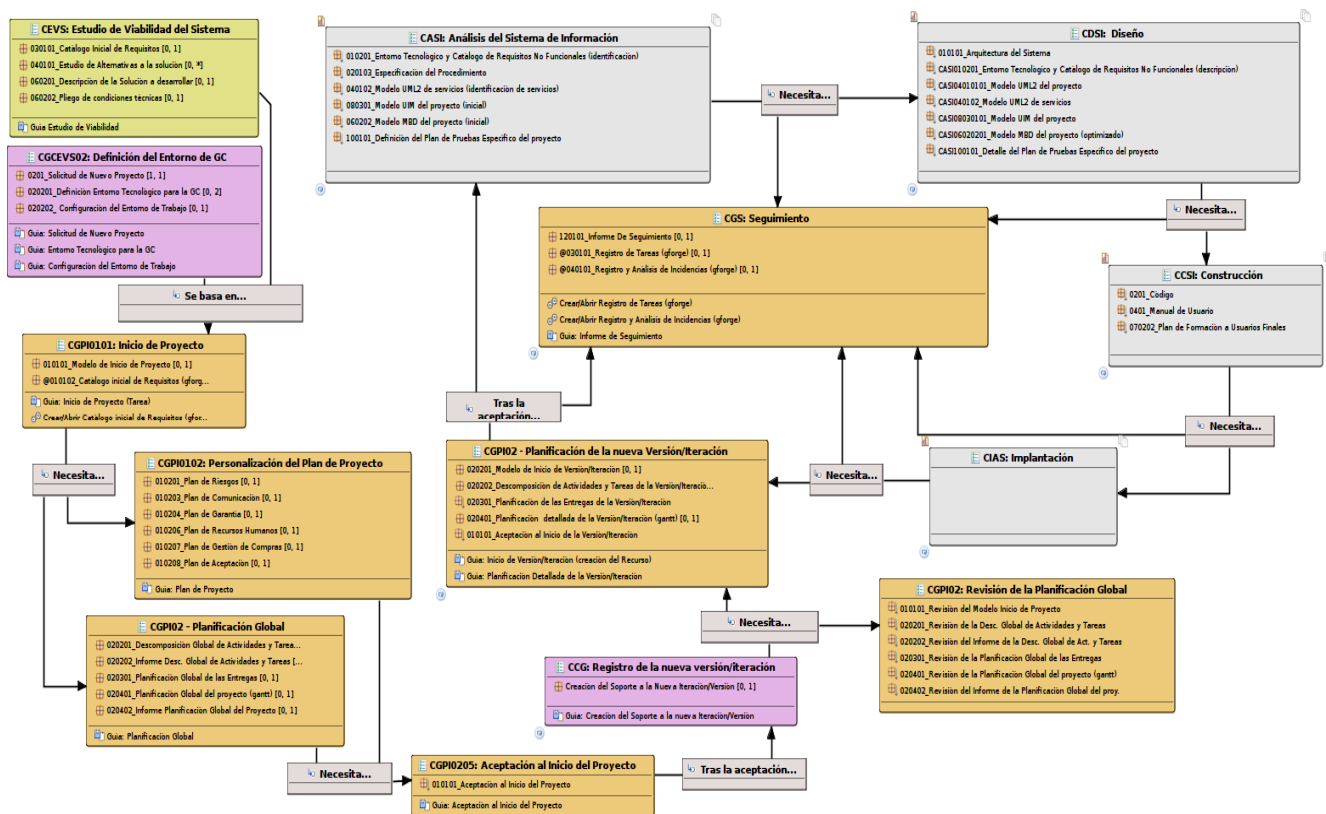


Figure 8. Dashboard model for Spanish Concil.

## REFERENCES

- Göransson, B., Gulliksen, J., Boivie, I. The usability design process - integrating user-centered systems design in the software development process. *Software Process: Improvement and Practice*, 8, 2 (2003), pp. 111-131.
- Brinkkemper, S. Method engineering: engineering of information systems development methods and tools. *Information and Software Technology*, 38, 4 (1996), pp. 275-280.
- Gonzalez-Perez, C. and Henderson-Sellers, B. A work product pool approach to methodology specification and enactment. *Journal of Systems and Software*, 81, 8 (2008), pp. 1288-1305.
- Griffiths, T., Barclay, P.J., Paton, N.W., McKirdy, J., Kennedy, J.B., Gray, P.D., Cooper, R., Goble, C.A., and Pinheiro da Silva, P. Teallach: a model-based user interface development environment for object databases. *Interacting with Computers* 14, 1 (2001), pp. 31-68.
- Guerrero-García, J., González-Calleros, J.M., Vanderdonck, J., and Muñoz-Arteaga, J. A Theoretical Survey of User Interface Description Languages: Preliminary Results. In *Proc. of Joint 4<sup>th</sup> Latin American Conference on Human-Computer Interaction-7th Latin American Web Congress LA-Web/CLIH'2009* (Merida, November 9-11, 2009). E. Chavez, E. Furta-
- do, A. Moran (Eds.). IEEE Computer Society Press, Los Alamitos (2009), pp. 36-43.
- Henderson-Sellers, B. and Ralyté, J. Situational Method Engineering: State-of-the-Art Review. *Journal of Universal Computer Science*, 16, 3 (2010), pp. 424-478.
- Hug, Ch., Front, A., Rieu, D., and Henderson-Sellers, B. A method to build information systems engineering process metamodels. *J. of Systems and Soft.* 82, 10 (2009), pp. 1730-1742.
- Jeusfeld, M.A., Jarke, M., and Mylopoulos, J. Meta-modeling for Method Engineering. The MIT Press, New York (2009).
- Karlsson, F. and Ågerfalk, P.J. Method-User-Centred Method Configuration. In: Ralyté, J., Ågerfalk, P.J., Kraiem, N. (Eds.), *Proc. of Situational Requirements Engineering Processes SREP'05* (Paris, August 29-30, 2005)
- Karlsson, F. and Wistrand, K. Combining method engineering with activity theory: theoretical grounding of the method component concept. *European Journal of Information Systems*, 15 (2006), pp. 82-90.
- Luo, P. A Human-Computer Collaboration Paradigm for Bridging Design Conceptualization and Implementation. In *Proc. of DSV-IS'94* (Carrara, June 8-10,

- 1994). Focus on Computer Graphics Series. Springer (1995), pp. 129–147.
12. Molina, A.I., Redondo, M.A., and Ortega, M. A methodological approach for user interface development of collaborative applications: A case study. *Science of Computer Programming* 74, 9 (July 2009), pp. 754-776.
  13. Sousa, K., Mendonça, H., Vanderdonckt, J.: Towards Method Engineering of Model-Driven User Interface Development. In *Proc. of TAMODIA'2007*. Lecture Notes in Computer Science, vol. 4849. Springer-Verlag, Berlin (2007), pp. 112-125
  14. The Moskitt Environment, ProDevelop, Valencia, 2010. Accessible at <http://www.moskitt.org/eng/proyecto-moskitt/>
  15. Vanderdonckt, J. Model-Driven Engineering of User Interfaces: Promises, Successes, and Failures. In *Proc. of ROCHI'2008* (Iasi, September 18-19, 2008). S. Buraga, I. Juvina (eds.). Matrix ROM, Bucarest, 2008, pp. 1–10.
  16. Wurdel, M., Sinnig, D., Forbrig, P.: Task-Based Development Methodology for Collaborative Environments. In *Proc. of EIS'2008*. LNCS, Vol. 5247. Springer, Berlin (2008), pp. 118-125.

# Improving the Flexibility of Model Transformations in the Model-Based Development of Interactive Systems

Christian Wiehr<sup>1</sup>, Nathalie Aquino<sup>2</sup>, Kai Breiner<sup>1</sup>, Marc Seissler<sup>3</sup>, Gerrit Meixner<sup>4</sup>

<sup>1</sup>University of Kaiserslautern, Software Engineering Research Group,  
Gottlieb-Daimler Str. 42, 67663 Kaiserslautern, Germany,  
{c\_wiehr, Breiner}@cs.uni-kl.de

<sup>2</sup>Centro de Investigación en Métodos de Producción de Software, Universitat Politècnica de València,  
Camino de Vera s/n, 46022 Valencia, Spain,  
naquino@pros.upv.es

<sup>3</sup>University of Kaiserslautern, Center for Human-Machine-Interaction,  
Gottlieb-Daimler Str. 42, 67663 Kaiserslautern, Germany,  
Marc.Seissler@mv.uni-kl.de

<sup>4</sup>German Research Center for Artificial Intelligence (DFKI),  
Trippstadter Str. 122, 67663 Kaiserslautern, Germany,  
Gerrit.Meixner@dfki.de

## ABSTRACT

A problem of approaches for model-based user interface development is related to the lack of flexibility in the kinds and varieties of user interfaces that can be generated. On the other hand, there are several approaches that propose maintaining system models at runtime in order to support runtime adaptations. In these cases, the user interface of a system can be adapted according to the context, on the fly. Flexibility problems are also present in these kinds of approaches. This paper briefly presents an approach for dealing with flexibility at design time. Ideas from this approach have been extended for use at runtime and have been applied to SmartMote, a universal interaction device for industrial environments.

## Author Keywords

Model-based development of interactive systems, MBUID, user interface, design, runtime, model transformation, model mapping.

## General Terms

Design, Experimentation, Human Factors, Verification.

## ACM Classification Keywords

D2.2 [Software Engineering]: Design Tools and Techniques – *Modules and interfaces; user interfaces*. D2.m [Software Engineering]: Miscellaneous – Rapid Prototyping; reusable software. H5.2 [Information interfaces and presentation]: User Interfaces – *Prototyping*.

## INTRODUCTION

As a result of ongoing technological progress, developers are faced with the problem of having to develop user interfaces (UIs) for a plethora of target devices and usage situations, which leads to enormous complexity. Model-based user interface development (MBUID) methodologies prom-

ise to reduce this complexity by leveraging different layers of abstraction, the respective models for expressing aspects of UIs on these levels, and transformation tools for the development and semi-automatic generation processes of UIs [20, 10]. Frameworks, such as the Cameleon Reference Framework (CRF) [9], have been proposed, which define the different abstraction layers and models (e.g., task, dialog, and presentation model) to be used for the systematic, user-centered design of multi-target and context-aware UIs.

Many of today's approaches possess limited flexibility in the kinds and varieties of UIs that can actually be generated by current transformation engines as well as in the capability of coping with UIs customized according to user preferences or context [23]. To overcome these shortcomings and to model context-aware and runtime adaptive UIs, new development methodologies are called for.

During the design phase, the stepwise refinement of the UI on different levels of abstraction enables clear separation of concerns (e.g., adding platform- or modality-specific aspects) and therefore supports the development process. If they are automated, all transformations between the levels need to be executed in order to react to adapted models at runtime. In terms of the flexibility of UIs, this does not only imply annotations within the models but also the constant transformation of the models themselves, which is a challenge in the discipline of modeling.

Another implication of this requirement is that the models need to be constantly available and must be interpreted during runtime, which is not well supported by popular transformation languages yet.

Therefore, new transformation mechanisms are needed that support the explicit specification of model transformations



and manipulations in order to increase the flexibility of today's engineering processes for context-aware UIs.

The remainder of this paper is structured as follows. Section 2 describes related work in the field of transformations in MBUID approaches. In Section 3, we present the Transformation Templates approach, which serves as the initial mapping concept for our transformation approach. In Section 4, we extend our runtime generation approach to improve the flexibility of its transformation process. In Section 5, we discuss results and conclude.

## RELATED WORK

With respect to the CRF, model transformations play an important role for the systematic refinement of the UI specification in the development process [15]. On the first three layers of the CRF, model-to-model transformations are used to transform abstract task models into abstract and concrete UI models. In the development of multi-platform UIs, model-to-text transformations are commonly used to derive the target code from the concrete UI specification for the final target platform.

The overall flexibility of a MBUID approach can be affected by several criteria. For example, its underlying transformational approach can use implicit or explicit transformations. In the first case, the logic of model transformations is fixed and hard-coded in the tools that implement the transformations. An example of these approaches is OO-Method [18]. Some problems of this kind of approaches are the limited diversity in the generated user interfaces during runtime, since the generation process is always the same, and the need to employ manual modifications on the generated code in order to deal with UI requirements that are not supported by the automated transformation process.

In the second case, the logic of model transformations is externalized and can be expressed using transformation models or transformation languages. Examples of transformation languages are Query View Transformation (QVT) [17], the Atlas Transformation Language (ATL) [13], and XSL Transformation (XSLT) [14]. While QVT uses a declarative mapping syntax, ATL relies on a hybrid language concept that supports declarative and imperative mapping rules. XSLT is a widely used standard of the W3C for the transformation of XML documents. XSLT is a functional transformation language, able to transform XML documents into other XML documents, text-based documents, or binary files. For a comparison of different transformation languages, we refer the reader to [21].

Although those transformation languages allow the explicit specification of transformations, they have not been very popular in the development of context-aware and runtime adaptive UIs. In concepts like Dynamic Model-based user Interface Development (DynaMo-AID) [11] and the Multi-Access Service Platform (MASP) [6], model transformation and adaptation during runtime are implemented in the underlying renderers, that is, using an implicit approach. While the MARIA language [19] uses XSLT for

the model transformations, it is unclear how the model adaptation during runtime is specified in this concept.

Other works that have been conducted to develop a context-aware and runtime adaptive user interface include [4] and [24]. The goal of [4] is to preserve the usability of a UI when changes of context, specifically changes of screen size, occur. Different possible UI presentations are designed and finite state machines are used to specify the transitions between them. Unlike in our work, this approach requires all different UIs to be specified at design time. In [24] a layout model is used that constrains and eventually defines the presentation of the UI at runtime. The model contains statements about the orientation and size of the UI elements that generate a constraint system at runtime to calculate the layout of the interface. Like in our work, statements are able to refer to the different UI models available at runtime, however, instead of a constraint system, we base our approach in a repository of mapping rules which are executed at runtime.

The problems discussed above clearly demonstrate that more flexibility is required in MBUID approaches with implicit or explicit transformation logic so that UI developers can easily deal with customized UI requirements.

## FLEXIBILITY AT DESIGN TIME: TRANSFORMATION TEMPLATES

A *Transformation Template* (TT) [2, 3] aims to explicitly specify the structure, layout, and style of a UI according to the preferences and requirements of the end users as well as in line with the different hardware and software computing platforms and environments in which the UI will be used.

A TT is composed of parameters with associated values that parameterize the model-to-model or model-to-code transformations. Figure 1 illustrates the use of a TT when transforming UI models. A model compiler takes the source UI model and a TT as input. The TT provides specifications that determine how to transform the source UI model into the target UI model. The specifications are expressed by means of parameters with values and selectors. Selectors define the set of elements of the source model that are affected by the value of the parameter. The model compiler follows the specifications to generate the target UI model.

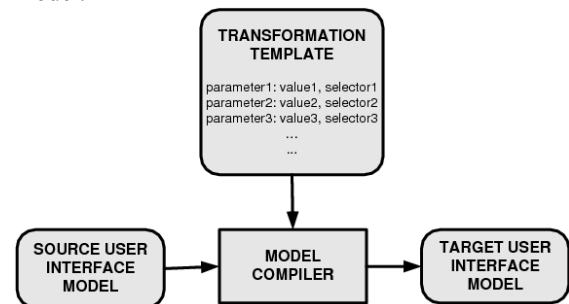
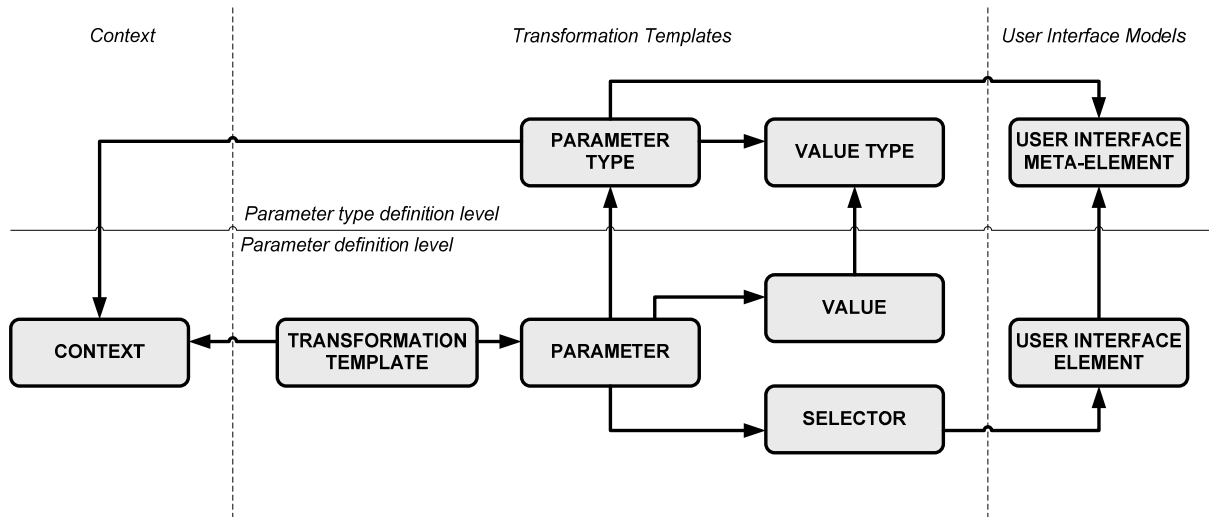


Figure 1. Parameters manipulating model transformations.



**Figure 2. Main concepts of the TTs approach.**

The main concepts that characterize the TT approach are depicted in Figure 2. There are concepts related to context, to UI models, and to the TT themselves.

*Context* refers to the context of use of an interactive system. In accordance with CRF, a context of use is composed of the stereotype of a user who carries out an interactive task with a specific computing platform in a given surrounding environment. Conceptualizing context, we can define TTs for different contexts of use.

A *user interface meta-element* represents, in a generic way, any meta-element of a UI meta-model. A *user interface element* represents an element of a UI model. These generic representations allow the TT approach to be used with different MBUID approaches to specify transformations at design time.

A *parameter type* represents a design or presentation option related to the structure, layout, or style of the UI. Defining a parameter type subsumes specifying the list of UI meta-elements that are affected by it, as well as its value type. The *value type* refers to a specific data type (e.g., integer, URI, color, etc.) or to an enumeration of the possible values that a parameter type can assume. A parameter type, with all or a set of its possible values, can be implemented in different contexts of use. In order to facilitate decision-making regarding these implementations, we propose that each possible value receive an estimation of its importance level and its development cost for different relevant contexts of use. In this way, possible values with a high level of importance and a low development cost can be implemented first in a given context, followed by those with a high level of importance and a high development cost, and so on. Possible values with a low level of importance and a high development cost would not have to be implemented in the corresponding context. Furthermore, for each relevant context of use, usability guidelines can be assigned to each possible value of a parameter type. These guidelines




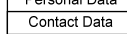
will help user interface designers in choosing one of the possible values by explaining the conditions under which the values should be used.

A *transformation template* gathers a set of parameters for a specific context of use. Each *parameter* of a TT corresponds to a parameter type and has both a value and a selector. A *value* is an instance of a value type. The value of a parameter corresponds to a possible value of the corresponding parameter type. A *selector* delimits the set of UI elements that are affected by the value of a parameter. We have defined different types of selectors that allow the designer to choose a specific UI element; all the UI elements of a certain type; the first or last element contained in a specific type of UI element; or other options.

Table 1 shows an example of the definition of a parameter type named grouping layout. This parameter type is useful for deciding how to present groups of elements. Four different possible values have been defined. The parameter type has been associated to two contexts of use: a desktop platform and a mobile one. For each context of use and each possible value, the importance level and development cost have been estimated. Table 1 also presents a list of usability guidelines for the desktop context and each possible value of the parameter type. These usability guidelines have been proposed from an extraction from [12].

TTs add flexibility in MBUID approaches because they externalize the design knowledge and presentation guidelines and make them customizable according to the characteristics of the project being carried out. TTs can then be reused in other projects with similar characteristics. Furthermore, TTs aim to diversify the kinds of UIs that a MBUID approach can generate. It is also important to note that TTs do not replace any implicit transformation logic or explicit transformation languages; instead, they provide a higher-level tier for UI designers to specify UI transformations.

**Table 1. Grouping layout parameter type.**

Parameter Type				
Name	Possible values enumeration			
	Value	Graphical description		
Grouping layout	group box			
	tabbed dialog box			
	wizard			
	accordion			
	Contexts			
	SW: C# on .NET - HW: laptop or PC		SW: iPhone OS - HW: iPhone	
Possible value	Importance level	Development cost	Importance level	Development cost
group box	high	low	high	low
tabbed dialog box	high	low	medium	medium
wizard	medium	medium	low	high
accordion	low	medium	medium	medium
Possible value	Usability guidelines (for desktop context)			
group box	Visual distinctiveness is important. The total number of groups will be small.			
tabbed dialog box	Visual distinctiveness is important. The total number of groups is not greater than 10.			
wizard	The total number of groups is between 3 and 10. The complexity of the task is significant. The task implies several critical decisions. The cost of errors is high. The task must be done infrequently. The user lacks the experience it takes to complete the task efficiently.			
accordion	Visual distinctiveness is important. The total number of groups is not greater than 10.			

## USING TRANSFORMATION TEMPLATES TO INCREASE FLEXIBILITY AT RUNTIME

One of the most challenging problems of context-aware UIs is that they have to cope with unpredicted changes or changes that cannot be modeled due to the combinatorial explosion of possibilities. In practice, this is often implemented by context manager components in the software, which are able to adapt the models with respect to the usage situation (e.g., tasks will be unavailable or restructured).

In the following subsections, we will briefly describe the core models of our MBUID approach, which deals with the problem mentioned above. This is followed by a description of our underlying mapping model, and a feasibility study.

### Core Models for the Design of Runtime Adaptive Systems

While we use several models to generate a fully functional UI [7], three core models can be identified for specifying the UI (as depicted in Figure 3). The Useware Markup Language (useML) [16] is used to structure the user's tasks. Based on this model, a modality-independent dialog model is derived, which provides a behavioral view of the UI. For describing the dialog behavior of the UI, the Dialog

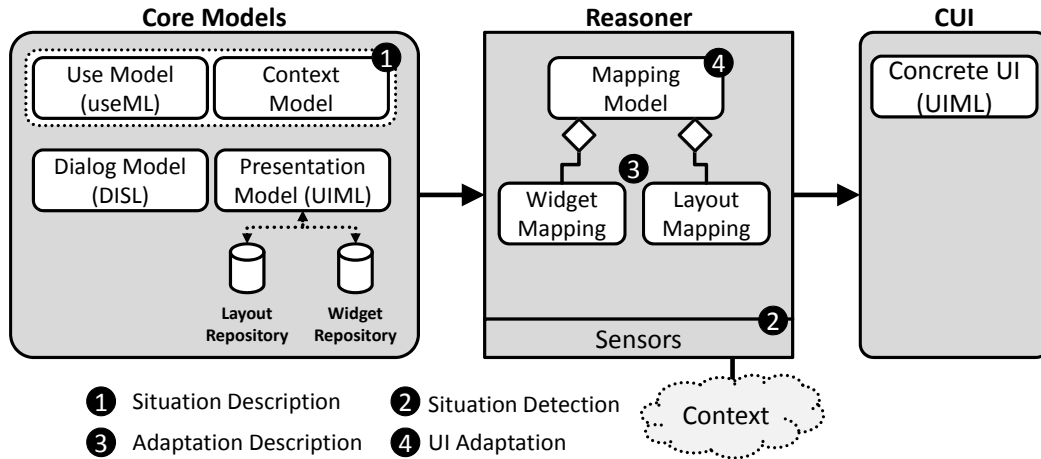
and Interface Specification Language (DISL) [22, 5] is employed. The User Interface Markup Language (UIML) [1] is used to define how the content is presented to the user in terms of concrete interaction objects and their layout [16]. Consequently, this presentation model can then be directly rendered and displayed to the user.

As tasks and their representation as interaction objects are central components of the models, they have to be linked across all levels of abstraction in order to combine their information. This is necessary due to the need for combined information from these models during the transformation process to keep the presented UI consistent with the underlying models.

### Mapping Model

TTs have been extended to refer to dynamic, runtime model data, so the UI can be automatically generated and react to adjusted models.

In addition to the models presented in the previous section, a contextual model (see Figure 3) was integrated in order to provide access to static context information (such as information about the user or environment that may have a direct influence on the interaction).



**Figure 3. Reasoning process in our MBUID approach for runtime adaptive systems.**

Mapping rules are provided as an extension to TTs to be able to refer to model data in addition to the fixed values that are available in the original TTs. To support the reasoning process in terms of reuse and performance, a repository containing frequently occurring widgets and widget configurations can be used. Using the layout mapping model, the structure of the UI (in terms of containers) is defined, with a hierarchical structure of UI containers and widgets, and using the widget mapping model, the mappings connecting abstract interaction objects to concrete widgets are described (see Figure 3). The mapping rules within the mapping models describe which UI template should be used under defined conditions. These conditions can be defined freely using all available model information. To avoid conflicting rules, priorities can be assigned to each rule. As an example, rules can be defined that map “select” actions by default to radio buttons, map them to a set of buttons if the environment requires touch input, or map them to a combo box if little space is available in the UI.

Both the containers and the widgets are specified using UIML, meaning that a concrete design is defined for the components, and are available from the widget repository. Containers are defined here as a type of UI elements for grouping or separating other elements, like a frame containing multiple radio buttons. Each container can contain other containers or widgets. This way, the UI can be assembled like a model kit. To provide reusable components, the UIML templates feature parameters to customize their presentation (e.g. size, color, captions, etc.). The repository can be easily extended by adding new UI components.

#### **Feasibility Study**

In order to demonstrate the feasibility of our approach, we developed a functional prototype as an extension to the SmartMote approach [8]. The models have been implemented and are connected using the Mapping Models to assemble the UI at runtime.

A short description of the assembly process will now be given. Due to its hierarchical structure, the layout mapping model is evaluated recursively. In this process, containers describe the abstract layout of the elements in the current phase of refinement. Placeholders for further refinement will be integrated as slots within the containers, which can either be refined by a widget or by other containers (see Figure 4). Multiple options can be specified for a slot to provide the required flexibility in the UI generation process. For each selected component, the corresponding UIML template is fetched and inserted into the slot of its parent component. The parameters of the templates are also set according to the mapping rules, using either fixed values or references to model data.

Thus, all of the transformation specification previously coded in the generator software can be formalized using these mappings, which guarantee by definition that the generation will be successful. Instead of using a chain of model-to-model transformations to create the UI, different kinds of information from all core models is necessary and used at the same time. By retaining the specialization of the models and reducing the complexity, this improves the performance of the reasoning process.

#### **DISCUSSION AND CONCLUSION**

In this paper, we presented a new mapping concept for improving the flexibility of an approach for developing context-aware UIs on the basis of UI models according to the levels of abstraction in the CRF.

For the development of such UIs, the underlying models have to be automatically manipulated during runtime. The TT approach can serve as the underlying concept, but it had to be extended. An extended TT concept was developed that supports the interlinking and mapping of different models during runtime. Mapping rules can use model values as input and manipulate the UI generation process during runtime. To test the feasibility of this concept, a first prototype was developed.

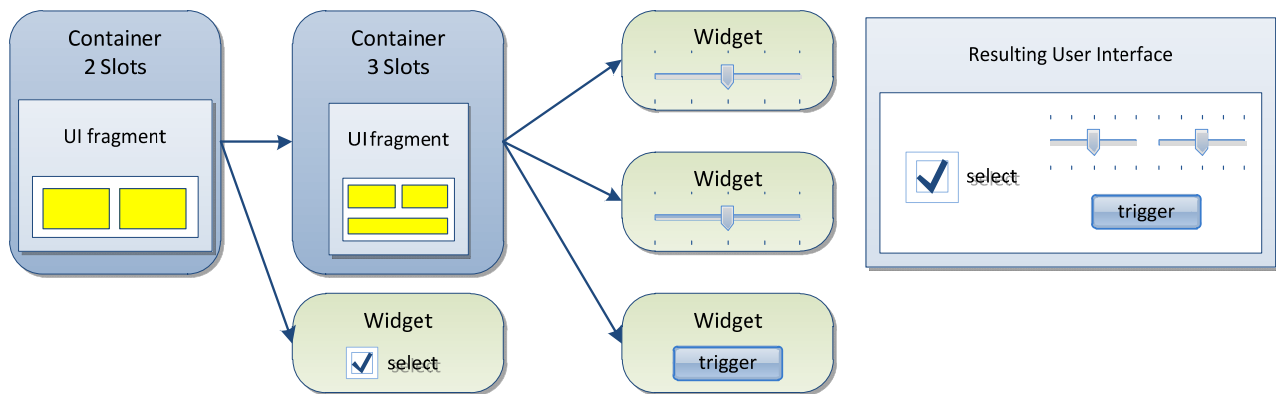


Figure 4. Relationships of container, slots, and widgets based on an example.

#### ACKNOWLEDGMENTS

This work has been developed with the support of MICINN under the project PROS-Req (TIN2010-19130-C02-02) co-financed with ERDF, GVA under the project ORCA (PROMETEO/2009/015) and the BFPI/2008/209 grant. We also acknowledge the support of the ITEA2 Call 3 UsiXML project under reference 20080026, financed by the MITYC under the project TSI-020400-2011-20.

Parts of the presented work are result of the GaBi project funded by the German Research Foundation (DFG) which is part of the AmSys research focus at the University of Kaiserslautern funded by the Research Initiative Rhineland-Palatinate.

#### REFERENCES

1. Abrams, M., Phanouriou, C., Batongbacal, A.L., Williams, S.M., and Shuster, J.E. UIML: An Appliance-Independent XML User Interface Language 31 (1999), pp. 1695-1708.
2. Aquino, N., Vanderdonckt, J., and Pastor, O. Transformation templates: adding flexibility to model-driven engineering of user interfaces. In *Proceedings of the 25th ACM symposium on applied computing, SAC'2010* (Sierre, March 2010). ACM Press, New York (2010), pp. 1195-1202.
3. Aquino, N., Vanderdonckt, J., Valverde, F., and Pastor, O. (2009) Using profiles to support model transformations in the model-driven development of user interfaces. In *Proc. of 7th Int. Conf. on Computer-Aided Design of User Interfaces CADUI'2008* (Albacete, June 2008). V. López Jaquero, Montero Simarro F, Molina Masso JP, Vanderdonckt J (eds). Springer, Berlin, pp. 35-46
4. Benoît Collignon, Jean Vanderdonckt, and Gaëlle Calvary. 2008. Model-Driven Engineering of Multi-target Plastic User Interfaces. In *Proceedings of the Fourth International Conference on Autonomic and Autonomous Systems (ICAS '08)*. IEEE Computer Society, Washington, DC, USA, 7-14.
5. Bleul, S., Schäfer, R., Müller, W.: Multimodal Dialog Description for Mobile Devices. *Gehalten auf der Workshop on XML-based User Interface Description Languages at AVI 2004*, Gallipoli, Italy (2004).
6. Blumendorf, M., Feuerstack, S., Albayrak, S. Multi-modal user interfaces for smart environments: the multi-access service platform. *Proceedings of the working conference on Advanced visual interfaces*. pp. 478-479 ACM, Napoli, Italy (2008).
7. Breiner, K., et al., 2009. Towards automatically interfacing application services integrated in a automated model-based user interface generation process. In *Proceedings of MDDAUI'09*. IUI '09, Sanibel Island, Florida.
8. Breiner, K., et al., 2011. Automatic adaptation of user workflows within model-based user interface generation during runtime on the example of the SmartMote. In *Proceedings of the 15th International Conference on Human-Computer Interaction (HCII 2010)*, Orlando, FL.
9. Calvary G, et. al. (2003) A unifying reference framework for multi-target user interfaces. *Interact Comput* 15(3):289-308.
10. Cantera Fonseca, J. M., González Calleros, J. M., Meixner, G., Paternó, F., Pullmann, J., Raggett, D., Schwabe, D., Vanderdonckt, J.: *Model-Based UI XG Final Report*, W3C Incubator Group Report, <http://www.w3.org/2005/Incubator/model-based-ui/XGR-mbui-20100504>, (2010).
11. Clerckx, T., Luyten, K., Coninx, K. DynaMo-AID: a Design Process and a Runtime Architecture for Dynamic Model-Based User Interface Development. In *The 9th IFIP Working Conference on Engineering for Human-Computer Interaction Jointly with the 11th International Workshop on Design, Specification and Verification of Interactive Systems*. pp. 11-13 Springer-Verlag (2004).
12. Galitz, W. O. *The essential guide to user interface design: an introduction to GUI design principles and techniques*. Wiley, New York, (2002).
13. Jouault, F., Kurtev, I. Transforming models with ATL. *Proc. of MoDELS 2005 Workshops*, vol. 3844 of Lec-



- ture Notes in Computer Science, Springer, Berlin. (2006), 128-138.
14. Kay, M. XSLT 2.0 and XPath 2.0 Programmer's Reference. John Wiley & Sons, 4th edition, (2008).
  15. Limbourg, Q., et al., UsiXML: a Language Supporting Multi-Path Development of User Interfaces, Proc. of 9th IFIP Working Conference on Engineering for Human-Computer Interaction. Lecture Notes in Computer Science, Vol. 3425, Springer, Berlin, (2005), 200-220.
  16. Meixner, G., Seissler, M., Breiner, K. Model-Driven Ueware Engineering. In Hußmann, H., Meixner, G., Zühlke, D. (eds.): Model-Driven Development of Advanced User Interfaces. 1-26 Springer, Heidelberg (2011).
  17. OMG, Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT), <http://www.omg.org/spec/QVT/index.htm>, last visit: July 2011.
  18. Pastor, O., and Molina, J. C. *Model-Driven Architecture in Practice: A Software Production Environment Based on Conceptual Modeling*. Springer-Verlag, 2007.
  19. Paternó, F., Santoro, C., and Spano, L.D. MARIA: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. ACM Trans. Comput.-Hum. Interact. 16, 1-30 (2009).
  20. Puerta, A.R. und Eisenstein, J. Towards a General Computational Framework for Model-Based Interface Development Systems. IUI '99, (1999), 171-178.
  21. Schaefer, R., A survey on transformation tools for model based user interface development. Proceedings of the 12th International conference on Human-Computer Interaction (HCI). Berlin, Heidelberg, Springer, (2007), 1178-1187.
  22. Schaefer, R.: Model-Based Development of Multimodal and Multi-Device User Interfaces in Context-Aware Environments. PhD-thesis, Aachen: Shaker Verlag, (2007).
  23. Vanderdonckt, J. Model-Driven Engineering of User Interfaces: Promises, Successes, and Failures. In S. Bura-ga and I. Juvina, Eds., *Proc. of 5th Annual Romanian Conf. on Human-Computer Interaction ROCHI'2008*, (Iasi, 18-19 September 2008), pp. 1–10. Matrix ROM, Bucarest, 2008.
  24. Veit Schwartze, Marco Blumendorf, and Sahin Albayrak. 2010. Adjustable context adaptations for user interfaces at runtime. In Proceedings of the International Conference on Advanced Visual Interfaces (AVI '10), Giuseppe Santucci (Ed.). ACM, New York, NY, USA, 321-324.

# Towards Evolutionary Design of Graphical User Interfaces

Josefina Guerrero-García<sup>1,2</sup>, Juan Manuel González-Calleros<sup>1,2</sup>, Jean Vanderdonckt<sup>3</sup>

<sup>1</sup>Universidad Autónoma de Puebla, Facultad de Ciencias Computacionales  
Ciudad Universitaria, PC. 72592, Puebla (México)

<sup>2</sup>R&D Unit, Estrategia 360 (México)  
{jguerrero, juan.gonzalez}@cs.buap.mx

<sup>3</sup>Louvain Interaction Laboratory, Louvain School of Management, Université catholique de Louvain  
Place des Doyens, 1 – B-1348 Louvain-la-Neuve (Belgium)  
Jean.vanderdonckt@uclouvain.be

## ABSTRACT

In this paper we argue that user interface design should evolve from iterative to evolutionary design in order to support the user interface development life cycle in a more flexible way. Evolutionary design consists of considering any input that informs to the lifecycle at any level of abstraction and its propagation through inferior and superior levels (vertical engineering) as well as the same level (horizontal engineering). This lifecycle is particularly appropriate when requirements are incomplete, partially unknown, to be discovered progressively. We exemplify this lifecycle by a method for developing user interfaces of workflow information systems. The method involves several models (i.e., task, process, workflow, domain, context of use) and steps. The method applies model-driven engineering to derive concrete user interfaces from a workflow model imported into a workflow management system in order to run the workflow. Instead of completing each model step by step, any model element is either derived from early requirements or collected in the appropriate model before being propagated in the subsequent steps. When more requirements are elicited, any new element is added at the appropriate level, consolidated with the already existing elements, and propagated to the subsequent levels.

## Author Keywords

Model Driven Engineering, UsiXML, UIDL, User Interfaces, Evolutionary design, Workflow systems.

## General Terms

Design, Experimentation, Human Factors, Verification.

## ACM Classification Keywords

D2.2 [Software Engineering]: Design Tools and Techniques – *Modules and interfaces; user interfaces*. D2.m [Software Engineering]: Miscellaneous – Rapid Prototyping; reusable software. H5.2 [Information interfaces and presentation]: User Interfaces – *Prototyping*.

## INTRODUCTION

In Software Engineering (SE), *evolutionary prototyping* is a software development lifecycle (SDLC) model in which a software prototype is progressively created for supporting demonstration to customer and for elicitation of system requirements elaboration [11]. Evolutionary prototyping

model includes four main phases: 1) definition of the basic requirements, 2) creating the working prototype, 3) verification of the working prototype, and 4) improvement of the requirements elicited.

Evolutionary prototyping model allows creating working software prototypes faster and may be applicable to projects where: system requirements early are not known in advance, new software needs to be created, and developers are not confident enough in existing software development experience.

Evolutionary design extends evolutionary prototyping in the sense that a system prototype, once it has been sufficiently refined and validated, may go further in the software development life cycle: from early to advanced design, if model-driven engineering is used as a method for the user interface (UI) development life cycle.

The remainder of this paper is structured as follows: the next section introduces the concept of evolutionary design, its motivation, and its definition. Next, the full implementation of this concept is applied on a software tool, and exemplified through two case studies. An experiment conducted to determine what the superior factors of evolutionary design over a traditional SDLC are. Last, we provide some related work and our conclusions.

## EVOLUTIONARY DESIGN CONCEPT

Evolutionary design consists of considering any input that informs to the lifecycle at any level of abstraction and its propagation through inferior and superior levels (vertical engineering) as well as the same level (horizontal engineering). A propagation is defined by an operation type (i.e., creating, deleting, modifying), a source and a target (e.g., a model entity, a model attribute, a model relationship), and a direction (i.e., forward, backward or bidirectional). For instance, creating a class in one model may propagate the creation of another class in another model. If there is continuous feedback between the evolving design representations, each design activity can benefit from information derived in the other steps. For example, user interface design benefits from task analysis; problems in the task analysis can in turn be revealed during user interface design, allowing benefit to be derived in both directions [1]. We give a

taxonomy of these propagations as model-to-model (M2M) transformations in HCI. A taxonomy of model transformations [16] showed that is important to consider horizontal versus vertical transformations. A horizontal transformation is a transformation where the source and target models reside at the same abstraction level.

A typical example is refactoring. A vertical transformation is a transformation where the source and target models reside at different abstraction levels. A typical example is refinement, where a specification is gradually refined into a full-fledged implementation, by means of successive refinement steps that add more concrete details. The remainder of this paper will focus only on creating operations as a systematic example for discussion propagations. Other operation types are similar.

### UI DESIGN METHOD FOR WORKFLOW

In this section, we briefly outline the steps of a method for designing GUIs of workflow information systems (WfIS). Since this paper is aimed at addressing the problem of propagation in evolutionary design, this outline is explained here only with a level of details that is useful for the rest of the discussion and for the sake of evolutionary design. For more details, please refer to [8]. We then provide a series of propagations that have been implemented in FlowiXML, a software that support evolutionary design based on the aforementioned method. The method is composed on the following major steps: Workflow information system requirements, design and implementation.

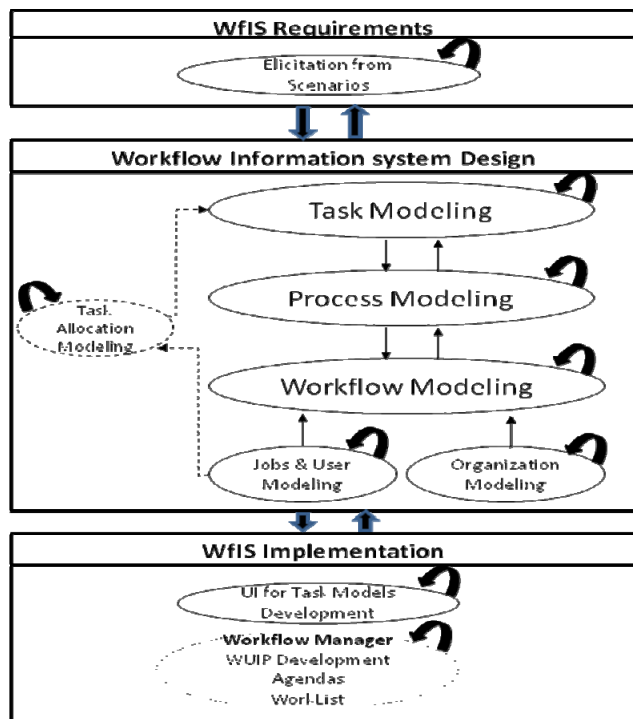


Figure 1. Methodology to develop workflow information systems

In Figure 1, forward and backward arrows denote the propagation of information from one model to another. For instance, a new task model must make available a task for a process model and vice versa, a new task in a process model might be detailed with a task model. Jobs, user stereotypes and organizational modeling just affect the workflow model. Then the workflow model makes them available for process modeling and task modeling. This particular aspect of concepts propagation was significantly useful for the software tools that support FlowiXML methodology [10].

“The process is iterative; once the implementation is done the system can be refined starting from the requirements or the design”. The modeling activities also are iterative. It is important to mention, the evaluation of the final implemented system is not in the scope of this paper. As many methods exist for this purpose and depending of the context of use they are used, we just assumed that in some way the system will be tested and if there is a need to iterate then there is a point back into the requirements definition and the design of the workflow information systems.

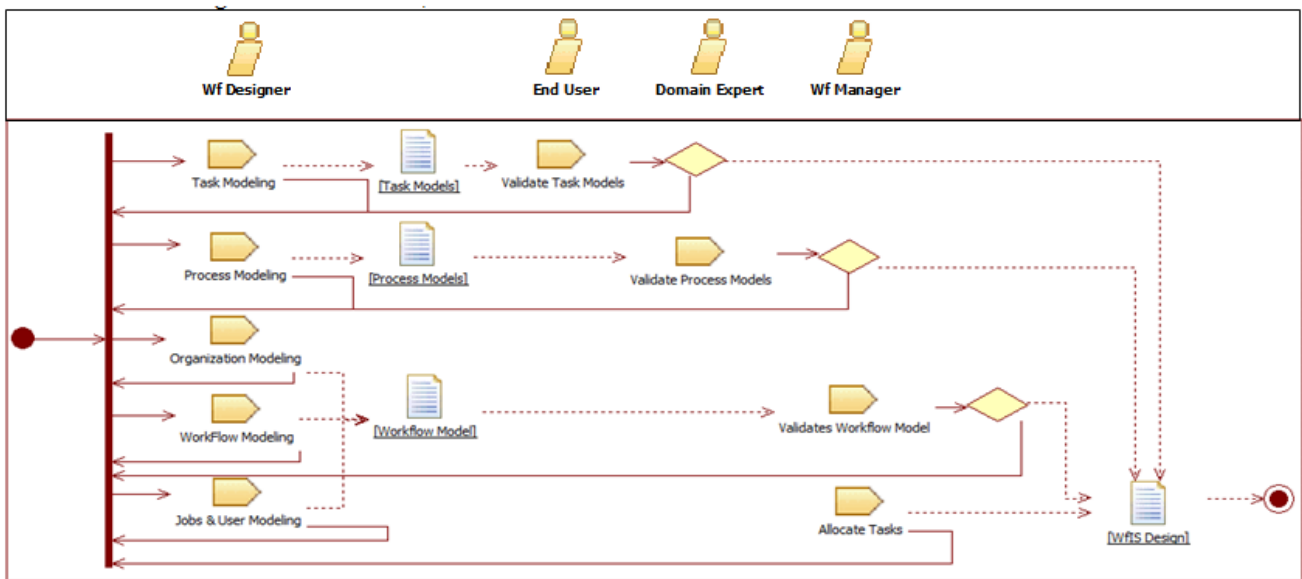
### Identification Criteria

One important and recurrent element that is of our interest is the concept of “task”. There are fundamental works that define workflow, processes, and tasks [23]. We established [5] a set of criteria to identify the concepts of: task, process and workflow. We looked at four dimensions surrounding the task execution (i.e., time, space, resources, and information). Any variation of any of these four dimensions, taken alone or combined, thus generates a potential identification of a new concept. At first everything is classified as tasks. A task could be part of a process model or a task model. Existing knowledge on task identification criteria is again relevant to make such separation. Table 1 shows part of these identification criteria.

	Time	Space (location)	User Stereotype
Workflow	Series of time periods	Different locations	Different groups of resources
Process	Series of time periods	Same location	One resource or a group of resources
Task	Same time period	Same location	Same resource

Table 2. Identification criteria.

A task model is composed of tasks performed by the same resource, in the same location and in the same time period. The reader must not be confused by the fact that during the execution of the task one of these properties might be changed. For instance, the task “insert client information” could be suspended during the execution of the task, for instance, while answering the phone, consequently when the user resumes the execution of the task the time period changes, it is no longer the same. However, this does not affect the nature of the task and its time periodicity; these kind of external events are part of the task life cycle.



**Figure 2. Evolutionary design of graphical user interfaces activity diagram.**

Location and change in user is something evident to identify but the time periodicity might be tricky to discover. If the designer confronts difficulties we propose the use of a timeline to gather tasks just to be sure that they are executed in the same time series. A process model is composed of tasks performed by one resource, or one or more group of resources; the location is the same; the time periodicity changes. Finally a workflow is composed of processes performed by one or more group of resources; located in different organizational units (within the same or different organizations); and the time periodicity of the processes changes.

### EVOLUTIONARY DESIGN OF GUI

Model-based user interface development environments (MB-IDEs) [22] seek to describe the functionality of a user interface using a collection of declarative models. The WfIS design is composed of several models (workflow, process and task) and several actors are involved (Wf designer, end-user, domain expert and Wf manager). The first activity corresponds to the consolidation of the concepts identified in the elicitation of the scenario. In [13] we have investigated three different techniques for eliciting model elements from fragments found in a textual scenario in order to support activities of scenario-based design. The design process is depicted in Figure 2, the drawings used corresponds to the Software Process Engineering Meta-model (SPEM) notation, promoted by the OMG.

The system design is an activity that can start from any model (Figure 2) except for the task allocation because it needs tasks and resources already defined. The design of a workflow permits designers to identify concepts freely and to start to detail based on their preferences. One designer may prefer to get into details of task modeling before describing a process model. Once the task models are ready,

then it can model the processes that then are arranged to represent the workflow. Another designer might have a better understanding of the problem with the workflow model (more abstract view of the problem) and then start to refine by adding process models and finished with task models. There is no constraint on the starting and end point for modeling, just to be sure keep the traceability of the concepts that are shared in different models (task model is part of process and a process model is part of a workflow model). To support traceability, the transformation language or tool needs to provide mechanisms to maintain an explicit link between the source and target models of a model transformation [15].

The end-user is the responsible of validating task models. Note by end-user we understand the person who actually is in charge of performing this task, i.e., the most qualified to say something about it. The process is validated by the domain expert. This is due to the fact that a process requires a higher level of understanding of the problem. In this view it could be any person in the organization that through the requirements analysis has been identified to be most familiar with the whole process modeled.

Finally the workflow model is validated by the Wf manager, who is the person or group of persons with an understanding of the whole workflow when processes are grouped. It is also, the Wf manager who is in charge of allocating tasks in a process to resources. The final result is the Workflow Information Systems (WIS) design. All these design activities are also accompanied with guidance for the modeling activities.

Progressing from one model to another involves transforming the model and mapping pieces of information contained in the source model onto the target model [3].

## SYSTEM IMPLEMENTATION

The implementation of the system refers to the UI generation of tasks. There is a plethora of user interface description languages that are widely used, with different goals and different strengths. A review of XML user interface description languages was produced [9] that compares a significant selection of various languages addressing different goals, such as multi-platform user interfaces, device-independence, and content delivery.

For instance, the global DISL structure consists of an optional head element for Meta information and a collection of templates and interfaces from which one interface is considered to be active at one time. Interfaces are used to describe the dialog structure, style, and behavior, whereas templates only describe structure and style in order to be reusable by other dialog components.

RIML semantics is enhanced to cover pagination and layout directives in case pagination needs to be done, in this sense it was possible to specify how to display a sequence of elements of the UI. RIML is device independent and can be mapped into a XHTML specification according to the target device.

No concepts or task models are explicitly used in See-scoaXML. The entry point of this forward engineering approach is therefore located at the level of Abstract UIs.

SunML presents a reduced set of elements that seems to be not enough, but the composition of widgets is used to specify more complex widgets. In TeresaXML the whole process is defined for design time and not for run-time. At the AUI level, the tool provides designers with some assistance in refining the specifications for the different computing platforms considered. The AUI is described in terms of Abstract Interaction Objects (AIOs) that are in turn transformed into Concrete Interaction Objects (CIOs) once a specific target has been selected.

UIML is a meta-language, is modality-independent, target platform-independent and target language-independent. The specification of a UI is done through a toolkit vocabulary that specifies a set of classes of parts and properties of the classes. Different groups of people can define different vocabularies: one group might define a vocabulary whose classes have a 1-to-1 correspondence to UI widgets in a particular language, whereas another group might define a vocabulary whose classes match abstractions used by a UI designer. UsiXML is structured according to different levels of abstraction relying on a transformational approach to move among them. It ensures the independence of modality thanks to the AUI level. It is supported by a collection of tools that allow processing its format.

WSXL is a Web services centric component model for interactive Web applications. WSXL applications consist of one or more data and presentation components, together with a controller component which binds them together and specifies their interrelated behavior.

XICL is a language to UI development by specifying its structure and behavior in an abstract level than using only DHTML. It also promotes reuse and extensibility of user interface components. The developer can create new and more abstract UI components. XIML supports design, operation, organization, and evaluation functions; it is able to relate the abstract and concrete data elements of an interface; and it enables knowledge-based systems to exploit the captured data.

Even that the model describe above can be incorporated to UIML, XICL or XIML languages, we have considered that UsiXML is the suitable language that could accommodate workflow concepts in a flexible way because its documentation is available including its meta-models and deep analysis can be done, it has a unique underlying abstract formalism represented under the form of a graph-based syntax, UsiXML allows reusing parts of previously specified UIs in order to develop new applications, It is supported by a collection of tools that allow processing its format, it allows cross-toolkit development of interactive application thanks to its common UI description format.

We rely on UI generation from task models techniques [14,17,18] for deriving UI. Some techniques apply task models, among other models, as a source to develop UIs, thus we model the allocation patterns with task models to derive Workflow User Interface Patterns (WUIPs). Finally we identify potential information relevant for the implementation of the workflow manager with the UI flow. After having defining the UIs involved in the workflow, we need now to link all the UIs: the one for the workflow manager and the ones for the workflow tasks. This will be achieved thanks to the user interface flow.

During the execution of work, information passes from one resource to another as tasks are finished or delegated; in FlowiXML we use an agenda assigned to each resource to manage the tasks that are allocated/offered to him. The manager uses the work list to view and manage tasks that are assigned to each resource. By linking UIs we expect to solve the problem of synchronizing the communication among them. We introduce some rules that can be applied to facilitate the modeling of the UIs flow that is relevant for the implementation of a WfIS.

### Software Support

The software support allows modeling the general WfIS defining workflow, process and task models, organizational units, jobs and resources involved, and allocation of task to resources. The software was developed based on the conceptual modeling presented in [8]. It is the result of constant improvements based on informal evaluations, interviews with the users after modeling a WfIS. The family of software tools is shown in Figure 3; we will explain them along with the description of some case studies. A video on YouTube illustrates the usage of the software tools (<http://www.youtube.com/user/FlowiXML>).

## APPLYING THE METHODOLOGY ON REAL LIFE CASE STUDIES

We have worked with several case studies applying our methodology. As the method does not force any sequence of steps, all case studies followed different development paths. This shows the flexibility of the proposed method.

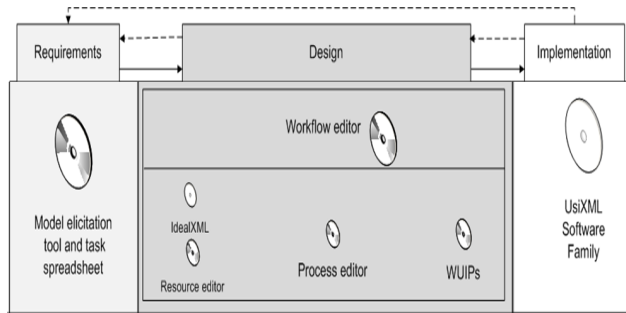


Figure 3. Software tool.

To gather the case studies we asked some students to propose a real life case study with at least twenty complex tasks developed in four different organizational units, involving sufficiently diversity of resources. The students, under our supervision, developed the case studies by completing a report including: introduction to the problem; scenario where the problem takes place; working hypotheses; task identification; task modeling; organizational modelling; jobs and user modeling; process modeling; workflow modelling; UI definition; workflow simulation and implementation; workflow analysis.

Students were free to select IdealXML or CTTE as the tool for task modeling. We will use one case study to exemplify the methodology and the evolutionary design, the reader could find the other case studies published in FlowiXML website (<http://www.usixml.org/index.php?mod=pages&id=40>).

The first task is about of how to extract WfIS concepts from textual scenarios describing the problem to be solved. The problem related to a library management, the whole process that makes new books and existing books available to borrow. A series of tasks have been identified: Insert book, correct notice, find mistakes, publish notice, insert notices, find book, among others. Once all tasks have been selected, grouped and described, a task model is constructed accordingly to its order.

Tasks are gathered in their corresponding organizational units and the selection and definition of jobs and users for the corresponding tasks are graphically selected (Figure 4). The next step consists on the definition of a process which indicates which tasks must be performed and in what order. Thus answering the question: what to do? After having identified tasks that are part of a process, then they have to be related to each other by means of process operators.

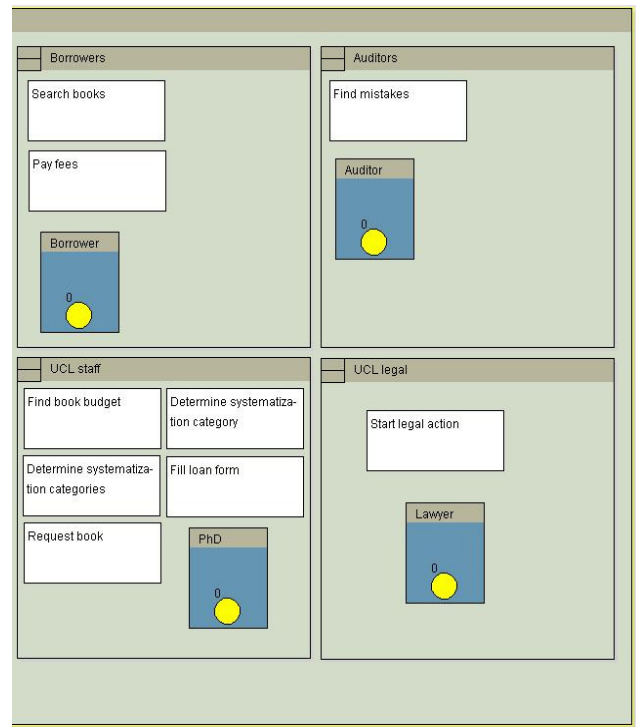


Figure 4. Organizational modeling and resource allocation.

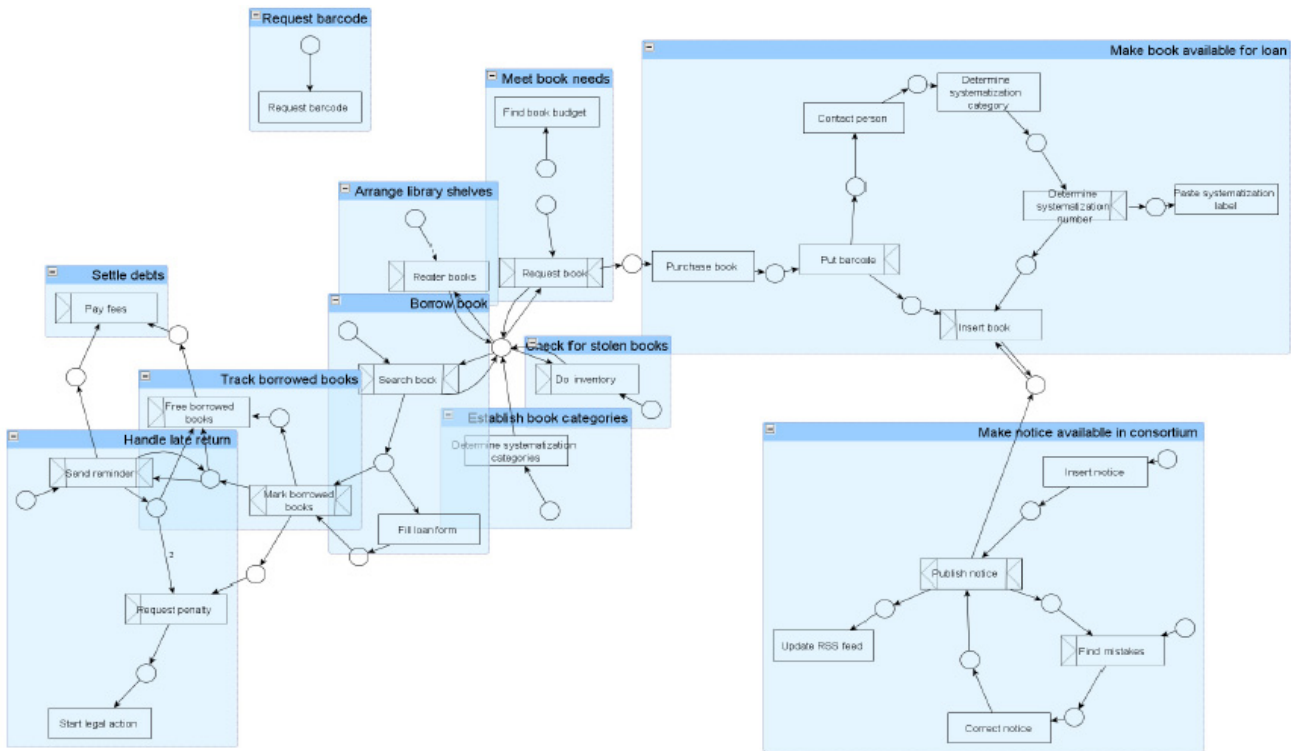
Three concepts are relevant for a process model: place (process state), transition (task), and process operators, this is known as a WF-net [23]. In Figure 5 the workflow of the case study is represented.

Task models do not impose any particular implementation so that user tasks can be better analyzed without implementation constraints. This kind of analysis is made possible because user tasks are considered from the point of view of the users need for the application and not how to represent the user activity with a particular system. For instance, let's consider the *search notice* task; the notice for a given book (if exists) can be searched using different parameters such as: ISBN, editor, keywords, title, authors, publication year; then the user validates its search information and launch the search; the system convey the results of the search; finally the user selects the code of the desired book. Figure 6 shows the task modeling using the CTT graphical notation [17].

We rely on UI generation from task models techniques [14,17,18] for deriving UI from task models, thus the whole GUI is developed thanks to this methodology. For instance the GUI for the Check in task is shown in figure 7.

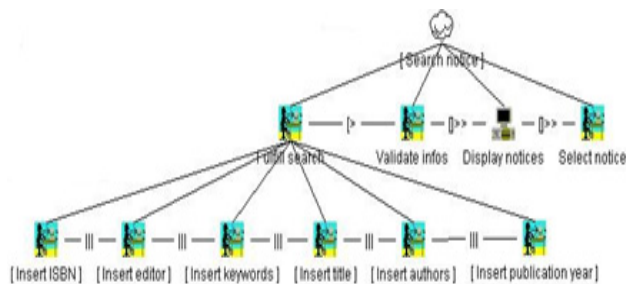
Finally, it is worth to say that the workflow modeling allows the users to identify portions of the workflow where some optimization could be introduced.





**Figure 5. Workflow modeling for the library management system.**

In our example, the current workflow could be improved by adding some patterns to optimize the workflow by minimizing the number of possible freezes. Moreover we added some flexibility to the workflow by allowing several resources to have the same job where it is not actually the case (such as for the librarian job which has only one resource associated currently).



**Figure 6. Task model.**

Another case study is SPORT-KIT project. SPORT-KIT is designed to be a training system like the Wii Fit® but more evolved and designed for elderly people. A battery of tests is run automatically with the users' interactions to know his/her wellness and physical capabilities so that we can train them effectively to improve them. Also, SPORT-KIT involves a tactile interface, a board with pressure sensor and a webcam. Following the identification criteria, we identify 28 tasks: Start SPORT-KIT, Start test, Personal test, Ask size, Ask waist measurement, Ask weight Compute BMI, Start EUROQUOL Test, 5 questions of the test,

Scale test, Store EUROQUOL result, Start STAND test, STAND test 2 feet, STAND test 1 feet left, STAND test 1 feet right, STAND test 1 feet eye closed, Store STAND test result, Start STEP Test, STEP Test, Store STEP Test result, Start FORCE Test, FORCE Test, Store FORCE Test result, Start SOUPLENESS Test, SOUPLENESS Test, Store SOUPLENESS Test result, Compare weight, Compute training program. Some of them are shown in Table 2. As we can observe, the goal of the SPORT-KIT system is to compute a training program.

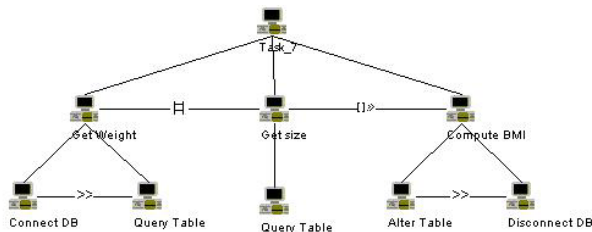
**Figure 7. GUI for the Check in task.**

We consider only the project **SPORT-KIT**, because the subject is deep enough with this single aspect of the final product. Here is the list of organizational units involved in this part:

- *Project leader*. Gives the interface and architecture to programmers, controls the UI and interaction.
- *Programmers*. Implement the solution; work with foundation server to share code amongst the project progression.
- *Testers*. Test the software as it's now, track bugs, give input on usability of the HMI.
- *Final testers*. Test the final software, give feedbacks and input to project leader and programmers.
- *Physical trainer*. Makes the training program, chooses the tests and helps to implement; reports to project leader.

ID	Task name	Definition	Nature
1	Start SPORT-KIT	User starts the program.	Interactive
2	Start Test	No previous test was done by the user	Automatic
3	Personal Test	User introduce information about himself	Automatic
4	Ask Size	User introduce his size in meter	Interactive
5	Ask waist measurement	User introduce his waist measurement, if he doesn't know clothes size is proposed	Interactive
6	Ask weight	User introduce his weight	Interactive
7	Compute BMI	Compute the body mass index and store it in the database	Automatic
8	Start EUROQUOL Test	Explain to user what he has to do with this test: just touch the best answer	Automatic
9	5 questions of the test	Ask the question about health status of the person	Interactive
10	Scale Test	The user has to evaluate his health status on a scale from 0 to 100	Interactive

**Table 3. Some identified tasks.**



**Figure 8. Compute BMI task model.**

For each task we can have a task model following the CTT notation [17]. For instance, Figure 8 shows the subtasks needed in order to compute the body mass index and store it in the database (task 7). After, we can identify which jobs are in charge of a particular task; in most of the task the principal actor is the user, the helper collaborate in some of them with the user, and the physician participates just in 5 tasks. User participates in the following tasks: Start Sport, Start Test, Personal Test, Ask Size, Ask waist measurement, Ask weight, Compute BMI, Start EUROQUOL Test, 5 Questions of the test, Scale Test, Store EUROQUOL result, Start STAND test, STAND test 2 feet, STAND test 1 feet left, STAND test 1 feet right, STAND test 1 feet eye closed, Start STEP Test, STEP Test, Start FORCE Test, FORCE Test, and SOUPLENESS Test. Helper participates in the following tasks: Start STAND test, STAND test 2 feet, STAND test 1 feet left, STAND test 1 feet right, STAND test 1 feet eye closed, STEP Test, and Start SOUPLENESS Test. Physician participates in the following tasks: Store STAND test result, Store STEP Test result, Store FORCE Test result, Store SOUPLENESS Test result, and Compare weight.

Next step in this case study is producing the process model. The identification criterion was the ability to regroup tasks in a consistent sub-process of the whole process. Also, consistency was automatically checked by the software we used, so we cannot break the rules.

The frontiers of the processes are defined as follows:

Process name	Tasks
Cligne sur SPORT-KIT	1, 2
Test <b>papier</b>	3 → 10
Test <b>pression</b>	21, 22
Test <b>back scratch</b>	24, 25
Calcul IMC	7
Test <b>step</b>	18, 19
Test <b>équilibre</b>	12 → 16
Sauve résultats dans la BD	11, 17, 20, 23, 26, 27

The complete workflow that uses the previously identified processes is shown in Figure 10, we use Petri net to model the workflow.

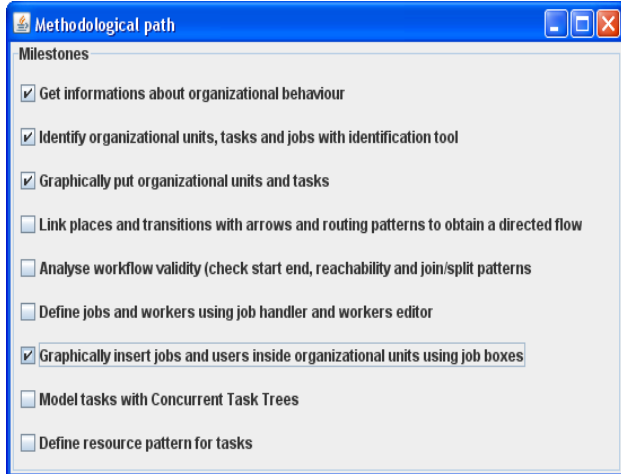
Finally, UIs for the case study are derived using the model-driven approach of UsiXML, a set of *transformation rules* were applied [see [www.usixml.org](http://www.usixml.org) for more information]. It is not the scope of this paper to address UI development but relies on existing work. UsiXML model-driven UI development uses as input a task model, enriched with our meta-description, and transform it into a user interface.

### Importance of the Evolutionary Design

Adding/modifying a task, user or object attribute in the textual scenario results into creating/modifying an attribute in the corresponding model. Similarly adding/modifying a new task on a workflow model affects the task modeling, as

another task must find its place in an existing task model or a new task model must be designed.

The propagation of these information impacts the different layers of the development process, without information traceability the evolutionary design would not be possible. In Figure 9 the methodological path editor is shown, where the different properties to use during the development process.



**Figure 9. Methodological path editor.**

For instance, Milestones are used to get a methodological path reminder. The workflow editor user may model the workflow using many different step orderings. It is a practical way to note what you still have to do, telling you where you are (after a holiday break for example).

## RELATED WORK

The multidisciplinary nature of system design can lead to a lot of different works in the literature which will allow developers to validate the quality of the system development in an efficient manner. In [20] some concepts for evolutionary systems and some of the technology investigations that are felt to support those concepts are presented. It discusses the evolutionary design of complex software which objective is to provide economic methods for systems to keep up with changer requirements over their lifetimes by providing a strong information base for evolution, enabling analysis of impacts of intended changes, enabling design and implementation of more adaptable systems.

The work of Chong Lee [2] draws from extreme programming (XP), an agile software development process, and claims-centric scenario-based design (SBD), a usability engineering process. Extreme programming, one of the most widely practiced agile methodologies, eschews large upfront requirements and design processes in favor of an incremental, evolutionary process. In [1] we can see how different user interface design artifacts can be linked to expose their common elements, facilitating the communication on which coevolutionary design is based.

The Clock architecture language [6] was designed to support the evolutionary design of architectures for interactive, multi-user systems. In [24] authors discuss an approach for linking GUI specifications to more abstract dialogue models and supporting an evolutionary design process, which is supported by patterns. Additionally, a proposal is presented of how to keep connections between concrete user interface, abstract user interface and a task model. Based on evolutionary character of software systems, Rich and Waters [19] have developed an interactive computer aided-design tool for software engineering; it draws a distinction between algorithms and systems, centering its attention on support for the system designer. In [4] authors have argued that task and architecture models can be semi-automatically linked.

They have developed a computer-based tool, *Adligo*, which generates links between the UAN as a task model and the Clock architecture as an architecture model. To support co-evolutionary software development the linkage process between the models can be carried out at any stage in the development cycle; none of the design artifacts has even to be complete. Krabbel *et al.* [12] adopted a variation of object-orientedness for constructing application software as a product and combined it with an evolutionary development strategy based on prototyping. The approach focuses on a close relationship between the tasks and concepts of the application domain and the software model. In [21] an approach to developing systems which can be summarised as ‘analyse top-down, design middle-out, and build bottom-up’ is presented. The novelty of the approach lies in its use of task analysis to define an appropriate domain for the system and then the use of a working prototype to grow a system from the bottom up.

Model-based user interface development environments show promise for improving the productivity of user interface developers and possibly for improving the quality of developed interfaces. There are many MB-UIDEs that follow a formalized method, but their supporting tools do not provide facilities to change the sequence of the method activities, thus restricting the possibilities to adapt the method. The TEALLACH design process [7] aims to support the flexibility for the designer lacking in existing environments by providing a variety of routes in the process; from one entry point, the designer/developer can select any model to design independently or associate with other models.

## CONCLUSION AND FUTURE WORK

Many existing software engineering processes are unable to account for continuous requirements and system changes requested throughout the development process. Evolutionary design has emerged to address this shortcoming. In this paper, a methodology to support evolutionary user interface development life cycle in a more flexible way was introduced. The methodology supports the evolutionary lifecycle including any level of abstraction to start the design.

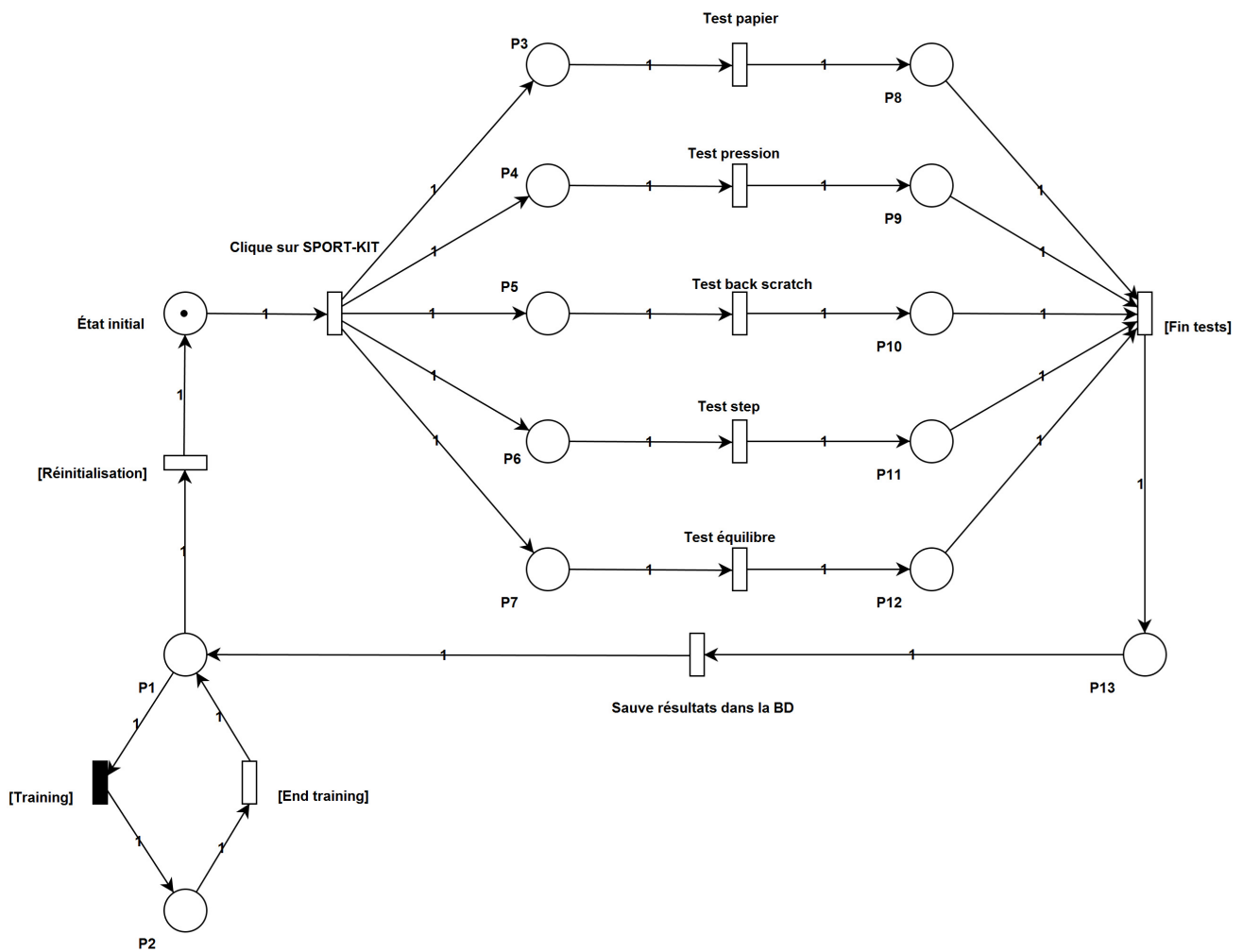


Figure 10. Partial view of the workflow for the SPORT-KIT project.

It is widely accepted that evolutionary design is a natural and preferred design process. A workflow editor was developed to support the method proposed in this paper. A collection of case studies is gathered to validate the methodology. As future work we plan to consider to extent our work to support the development of Post-wimp UIs. Also we will apply the methodology to different context of use, such as: education, training, medicine.

#### ACKNOWLEDGMENTS

We acknowledge the support of the ITEA2 Call 3 UsiXML (User Interface extensible Markup Language – <http://www.usixml.org>) European project under reference #2008026 and its support by Région Wallonne DGO6, and the support of the Repatriation program by CONACYT. We thank all UIDL reviewers for their fruitful feedback that contribute in the enhancement of this paper.

#### REFERENCES

1. Brown, J., Graham, N., and Wright, T. The Vista Environment for the Coevolutionary Design of User Interfaces. In *Proc. of CHI'98*. ACM Press, New York

- (1998), pp. 376–383.
2. Chong Lee, J. Embracing agile development of usable software systems. In *Proc. of Extended Abstracts of CHI'2006*. ACM Press, New York (2006), pp. 1767–1770.
3. Clerckx, T., Luyten, K., and Coninx, K. The Mapping Problem Back and Forth: Customizing Dynamic Models while preserving Consistency. In *Proc. of TAMODIA'-2004*. ACM Press, New York (2004), pp. 33–42.
4. Elnaffar, S. and Graham, N.C. Semi-Automated Linking of User Interface Design Artifacts. In *Proc. of CADUI'-99*. Kluwer Academic Pub. (1999), pp. 127–138.
5. Garcia, J.G., Vanderdonckt, J., and Lemaigre, Ch. Identification Criteria in Task Modeling. In *Proc. of I<sup>st</sup> IFIP TC 13 Human-Computer Interaction Symp. HCIS'2008*. IFIP, Vol. 272, Springer, Boston (2008), pp. 7–20.
6. Graham, T.C.N. and Urnes, T. Linguistic support for the evolutionary design of software architectures. In *Proc. of ICSE'96*, IEEE Computer Society, Los Alamitos (May 1996), pp. 418–427.

7. Griffiths, T., Barclay, P.J., Paton, N.W., McKirdy, J., Kennedy, J., Gray, P.D., Cooper, R., Goble, C.A., and Pinheiro, P. Teallach: a Model-based User Interface Development Environment for Object Databases. *Interacting with Computers* 14, 1 (December 2001), pp. 31–68.
8. Guerrero García, J., Lemaigre, Ch., González Calleros, J.M., and Vanderdonckt, J. Towards a Model-Based User Interface Development for Workflow Information Systems. *Int. J. of Universal Comp. Science* 14, 19 (Dec. 2008), pp. 3236–3249.
9. Guerrero-García, J., González-Calleros, J.M., Vanderdonckt, J., and Muñoz-Arteaga, J. A Theoretical Survey of User Interface Description Languages: Preliminary Results. In *Proc. of LA-Web/CLIHIC'2009* (Merida, November 9–11, 2009), IEEE Computer Society Press, Los Alamitos, 2009, pp. 36–43.
10. Guerrero García, J., A Methodology for Developing User Interfaces to Workflow Information Systems. PhD. Thesis, UCL (2010).
11. Hekmatpour, S. Experience with evolutionary prototyping in a large software project. *ACM SIGSOFT Software Engineering Notes* 12, 1 (Jan. 1987), pp. 38–41.
12. Krabbel, A., Wetzel, I., and Züllighoven, H. On the inevitable intertwining of analysis and design: developing systems for complex cooperations. In *Proc. of DIS'97*. ACM Press, New York (1997), pp. 205–213.
13. Lemaigre, Ch., Garcia, J.G., and Vanderdonckt, J. Interface Model Elicitation from Textual Scenarios. In *Proc. of 1<sup>st</sup> IFIP TC 13 Human-Computer Interaction Symposium HCIS'2008*. IFIP, Vol. 272, Springer, Boston (2008) pp. 53–66.
14. Limbourg, Q. and Vanderdonckt, J. Addressing the Mapping Problem in User Interface Design with UsiXML. In *Proc. of TAMODIA'2004*, ACM Press, New York (2004), pp. 155–163.
15. Loia, V., Staiano, A., Tagliaferri, R., and Sessa, S. An evolutionary hybrid approach to the design of a decision support system. In *Proc. of SAC'00*, Vol. 1, ACM Press, New York (March 2000), pp. 524–528.
16. Mens, T. and Van Gorp, P. A Taxonomy of Model Transformation. *Electronic Notes in Theoretical Computer Science* 152 (2006), pp. 125–142.
17. Paternò, F. *Model-Based Design and Evaluation of Interactive Applications*. Springer-Verlag, London (1999).
18. Puerta, A.R. and Eisenstein, J. Towards a general computational framework for model-based interface development systems. *Knowledge-Based Systems* 12, 8 (1999), pp. 433–442.
19. Rich, Ch. and Waters, R.C. Computer aided evolutionary design for software engineering. *ACM SIGART Bulletin* 76 (April 1981), pp. 14–15.
20. Salasin, J. and Shrobe, H. Evolutionary design of complex software (EDCS). *ACM SIGSOFT Software Engineering Notes* 20, 5 (December 1995), pp. 18–22.
21. Seaton, P. and Stewart, T. Evolving task oriented systems. In *Proc. of CHI'92*, ACM Press (1992), pp. 463–469.
22. Szekely, P., Retrospective and Challenges for Model-Based Interface Development. In *Proc. of DSVIS'96*, F. Bodart and J. Vanderdonckt (Eds.), Springer-Verlag, 1996, pp. 1–27.
23. van der Aalst, W. & van Hee, K., *Workflow Management: Models, Methods, and Systems*. THE MIT Press, Cambridge. 2002.
24. Wolff, A., Forbrig, P., Dittmar, A., and Reichart, D. Linking GUI elements to tasks: supporting an evolutionary design process. In *Proc. of TAMODIA'2005*. Springer, Berlin (2005), pp. 27–34.

# Challenges for a Task Modeling Tool Supporting a Task-based Approach to User Interface Design

Costin Pribeanu

National institute for Research and Development in Informatics – ICI Bucharest  
Bd. Maresal Averescu Nr. 8-10, Bucharest, Romania  
+40 (0)213160736 - pribeanu@ici.ro

## ABSTRACT

In a task-based approach to user interface development, the task model is given a leading role among other models. Task modeling is a key activity in the design of a usable user interface. Task patterns that are based on both task and domain models are typical interaction structures that are capturing operations performed onto domain objects. This paper is discussing some key requirements for a task modeling tool aiming to support model transformations in UsiXML: full computer-aided task decomposition, model validation, model simulation, operations with task patterns, and simultaneous access at task and domain model elements based on mapping rules.

## Author Keywords

UsiXML, model transformation, task patterns.

## General Terms

Design, Experimentation, Human Factors, Verification.

## ACM Classification Keywords

D2.2 [Software Engineering]: Design Tools and Techniques – *Modules and interfaces; user interfaces*. D2.m [Software Engineering]: Miscellaneous – Rapid Prototyping; reusable software. H5.2 [Information interfaces and presentation]: User Interfaces – *Prototyping*.

## INTRODUCTION

Models are used to capture design knowledge needed for the construction of the future user interface (UI). Main concepts abstracted into these models refer to users, tasks, application domain, presentation and dialog. In a transformational approach to UI design the task and domain models are the main source models for the derivation of presentation and dialog models at various levels of abstraction [2], [3], [6].

Model-based approaches which are giving the task model a leading role among the other models are also referred to as task-based approaches. There are several model-based approaches that are using the information from both the task and domain models in order to exploit task patterns associated with typical operations performed onto domain objects [10], [14]. A limitation of these approaches is the fact that they are exploiting only the last layer in the task models.

A well known task modeling notation is CTT (Concur Task Tree) which was widely used in various interactive environments such as CTTE (CTT Environment) and Teresa

[8]. As pointed out by Luyten et al, CTT served as inspiration for several tools aiming at supporting task modeling in UsiXML [7]. CTTE enables the designer to create task trees and to specify task properties such as task type, frequency, and estimated execution time. There are two important CTTE features that represent *a first key requirement for a task modeling tool: the model validation and the model simulation*.

The goal of the UsiXML project is to develop an innovative model driven language supporting the “ $\mu 7$ ” concept: multi-device, multi-user, multi-cultural / language, multi-organization, multi-context, multi-modality and multi-platform. UsiXML will define a flexible methodological framework that accommodates various development paths as found in organizations and that can be tailored to their specific needs [5].

This paper is aiming to discuss some key requirements for a task modeling tool aiming to support the model transformations in UsiXML: full computer-aided task decomposition, operations with task patterns, and simultaneous access at domain and task model based on mapping rules.

In order to explain our approach we will use as example a software assistant for formative usability evaluation that was previously developed in a task-based approach. We will focus on the relationships between objects and analyze the categories of tasks which are afforded by each type of relationship. Then we will look for mappings between the domain model and the task model with the aim of covering the whole task hierarchy.

The rest of this paper is organized as follows. In section 2, we will briefly describe our task modelling framework and some general mappings between domain and task models. We will also present our example. In the next section we will discuss the difficulties associated with a transformational task-based approach and some limitations of the existing task modeling tools. The paper ends with conclusion in section 4.

## APPROACH

### The layered task model

In our approach we distinguish between three decomposition layers, which are relevant in the task modeling for user interface design [11]:

- A *functional task layer* that results from mapping application functions onto user tasks. Each function cor-



responds to a business goal, which is accomplished by carrying on one or several user tasks.

- A *planning task layer* that results from the decomposition of functional tasks up to the level of unit tasks, having a clear relevance for the user [4]. This layer shows how users are planning task performance by decomposing a task in sub-tasks and giving an ordering preference for each of them.
- An *operational task layer* that results from the decomposition of unit tasks up to the level of basic tasks, showing how a unit task will be actually carried on by using various interaction techniques [13].

In terms of a pattern language philosophy [1], domain and task models are the main forces driving the model-based design of a UI. In a previous work, we identified several task-domain mappings that are useful for the model-based derivation of the user interface [10]:

- *Unit tasks* – domain objects: operations performed onto domain objects, such as create new, edit, display or delete, are modeled as unit tasks in the task model. Unit tasks could have a simple operational structure or could be nested in a goal hierarchy.
- *Basic tasks* – domain object attributes / available commands: operations performed onto domain object attributes are mapped onto information control basic tasks while available commands on the target platform are mapped onto function control basic tasks.

In the same work, we presented a software tool exploiting these mappings between task and domain models in order to produce the operational task layer. A *second key requirement for a task modeling tool is to provide support for computer-aided task decomposition in a transformational approach*. This means to cover also the decomposition at functional and planning levels by exploiting the mappings between the task and domain models, as shown in Table 1.

Task model	Task goals	Domain Model
Functional layer	Class and object management	Classes and objects
Planning layer	Add, edit, display, delete objects and relationships	Objects and relationships
Operational layer	Edit or display object attributes	Object attributes
Operational layer	Commands (transactions, navigation)	Objects, relationships and attributes

**Table 1. The task-domain mappings.**

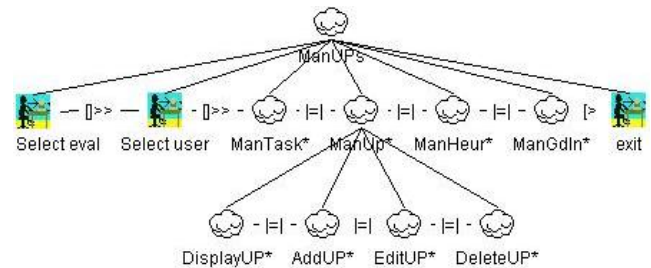
Basic tasks for function control have as operational goals to start, confirm or cancel a transaction (e.g. add, edit, ok, cancel) over domain model elements or to navigate between interaction units.

### Example

We will use as example a software assistant for formative usability evaluation that was developed in a task-based approach [12]. The purpose of formative usability evaluation is to identify and fix usability problems as early as possible in the development life cycle. There are two main categories of methods frequently used: inspection-based evaluation (heuristic evaluation, guideline-based evaluation) and user testing. In each case, a heuristic and / or usability guideline is used to document the usability problem. Usability problems are recorded for each evaluation task.

The main goal of the application is the management of usability problems (UP).

The functional layer shows the business goals that should be accomplished with the target application. Each function is mapped onto a user task. In our example, the tasks are accomplishing the management of four classes of objects: evaluation tasks, usability problems, heuristics and guidelines. The task decomposition at functional level is presented in Figure 1 using the CTT notation.



**Figure 1. The task ManUP – functional layer.**

Usability problems are the central class in this application. Heuristics are used to document a usability problem. Guidelines are used to detail a heuristic. The heuristics are displayed in a list box. When a heuristic is selected, its definition is displayed in a text box below. Several heuristics could be associated to a usability problem.

The user could consult the guidelines detailing a heuristic in a separate window (by pressing the ShowGuidelines button). In Figure 1, the interaction unit for editing a usability problem is presented.

Ideally, it should be possible to create the user interface illustrated in Figure 1 by applying transformation rules to the task and domain models.

Tasks on the first decomposition level in Figure 1 are interactively specified in CTTE since there is a simple one-to-one mapping from application functions to user tasks. The abstract (complex) tasks are iterative (infinitely) and linked with the “|=” temporal operator (could be performed in any order).

**Figure 2. The user interface for editing a usability problem.**

The next decomposition level could be obtained automatically, by performing a task to task transformation. This task pattern has four types of operations for each class: display, add, edit and delete. The four tasks are also iterative (infinitely) and linked with t ' [= ] temporal operator (could be performed in any order) like their parent tasks.

When simulating the task model, we experienced difficulties since CTTE supports both task id and task name, but is using the task name in simulation. So we had to rename all similar tasks in order to get unique names (e.g. AddT, AddUP, AddH, AddGdIn), which resulted in an extra work and reduced the specification readability. Therefore, *a third key requirement for a task modeling tool is to automatically assign a task id, to use it for internal validation and model simulation and to display the task name (which should not be unique).*

This requirement is important since it makes possible to define task patterns and to perform operations with them easier.

## TRANSFORMATION ISSUES

### The development path

The flexibility principle of UsiXML methodology makes it possible to support different development paths. In our approach, the concrete interface model is produced by applying transformation rules from task and domain models. The rationale for this approach is explained below.

According to Norman, changing the artifact is changing the nature of the task [9]. The change of platform is not only affecting the operational task layer but also the functional and operational layers. The task requirements are different for desktop computers and mobile devices. Although is possible to migrate from a desktop computer to a mobile device, this works well for some tasks that are likely to be performed in a particular context of use. For example, consulting the heuristics – guidelines hierarchy could migrate

to a mobile device. Otherwise, each platform is supporting specific interaction metaphors which are shaping the overall user interface design.

Another problem is the change of modality. Some application functions could support the change of graphic modality with voice. However, for a relatively large application is difficult to ensure an acceptable level of usability without radical changes in the task model, at least at planning level. For example, respecting accessibility guidelines only ensures that a disabled user could access the content. In order to make it usable, the interaction spaces should be tailored to the limitations of human working memory. In other words, the user interface should be structured differently for a blind user, in smaller interaction units.

The choice of what to show into an interaction unit is an important design decision that sometimes is taken beforehand. Many designers put forward an interaction concept or a metaphor that provide with a better user experience. This means that the choice of structuring the user interface in interaction units could be a subjective human decision. In many cases, criteria like familiarity, consistency (with other applications or real life artifacts) prevail.

The goal hierarchy in the task model, the relationships between objects and the mappings between task and domain models could positively influence the design decision on structuring the user interface in a usable way but we consider that these criteria should be used at the concrete interface level.

In all these cases (change of platform, change of modality, user interface structure) the task model is also changing because of the navigation tasks. When the user interface is split into several interaction units, the number of navigation tasks increases. Simulating the task model helps in structuring the interface since it makes obvious the navigation tasks that are not needed.

For example, the list of heuristics could be available in the same window or in a separate window opened when the user wants to associate a heuristic to a usability problem. Since each usability problem is documented by at least one heuristic, the best is to have the list of heuristics permanently displayed in the editing window, like in Figure 1. From a methodological point of view, we advocate for a complete task specification at operational level (including navigational tasks) based on task and domain mappings as a prerequisite for the user interface derivation.

### Task & domain to task transformations

We consider that transformation rules should be embodied in a set of tools that are producing UsiXML specifications.

In order to make a UsiXML specification manageable with a reasonable effort, several functions are required for such a tool: opening a model, finding a model element, inserting the specification of a model element into the model, and combining small models into larger models.

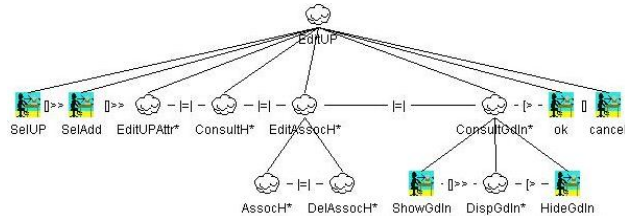
Transformations from task and domain to task are producing the planning and operational layers in the task model. Actually, these transformation rules are producing task patterns that could be further exploited in similar applications.

The transformations producing the planning layer are exploiting the semantic relationships between objects. In our example, editing a usability problem is a complex task with several sub-goals: editing the attributes of the usability problem, displaying info about a heuristic, editing the associated heuristics, and displaying the guidelines associated with a heuristic.

Depending on the use of semantic relationships, the tasks on the second decomposition level in Figure 2 could have a specific degree of complexity. For example, the task “EditUP” has a hierarchical structure and is further decomposed in several sub-tasks corresponding to the aforementioned sub-goals.

In Figure 3 the planning layer for the task EditUP is presented. There are three relationships between objects used in this task. The first is the relationship between the evaluation tasks and usability problems (aggregation) which is visible (the task id is displayed). The second is the relationship between a usability problem and the associated heuristics (1...\*) which is visible. The second is the relationship between a heuristic and the associated guidelines which could be made visible by pressing the “Show guidelines” button.

When editing a usability problem, the user can change the UP attributes and the associated heuristics but can only consult the heuristics and associated guidelines. As it could be seen in Figure 3, the use of domain objects and their relationship are structuring the task model at planning layer.



**Figure 3. The task EditUP – planning layer.**

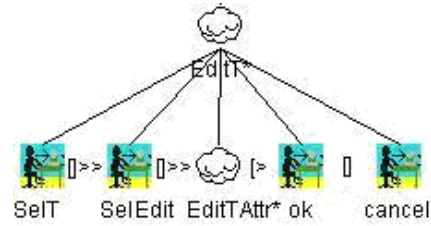
A similar task pattern applies for adding a usability problem. The only difference is the task for selecting the usability problem which is no longer needed. In this respect, *a fourth key requirement for the task modeling tool is to provide with operations over a task patterns collection.*

The transformations rules are using attributes of the task in the source task model (such as canonical task type and task nature) and the information available in the domain model. In our approach, the source task is the parent task (which is decomposed).

*A fifth key requirement for a task modeling tool is to provide a way to specify a link between each task and the corresponding domain model element.*

In this respect, the corresponding element for each task could be an object class, a relationship, an object attribute or none (for navigational tasks). This mapping will be further exploited for task decomposition, concrete user interface derivation, and writing the code for event processing. Ideally, the task modeling tool should provide a model validation that extends the CTTE feature (based on temporal operators) with checking the consistency of mappings between task and domain elements.

If no relationship is involved, a simpler task pattern is needed for editing a domain object. The task decomposition for editing the evaluation tasks (EditT) is presented in Figure 4.



**Figure 4. The task EditT – planning layer.**

<p>(a) class management</p>	<p>The user is provided with typical operations performed onto the objects of a given class. Tasks are repetitive and could be performed in any order</p>
<p>(b) edit complex</p>	<p>The pattern applies for complex editing task that have several sub-goals: change of attributes, change of relationships, displaying of other objects.</p>
<p>(c) edit object attrib-</p>	<p>The user selects the object and then the edit command. A similar pattern applies for adding an object (in this case the preliminary object selection is not needed).</p>
<p>(d) delete object</p>	<p>The user selects the object and then the command. The object attributes are displayed together with a shield message asking to confirm / cancel the deletion.</p>
<p>(e) display object</p>	<p>The user selects the object and then the command. The object attributes are displayed until the user confirms the visualization</p>

**Table 2. Detailed mapping rules.**

The unit tasks “EditUPAttr” and “EditTAttr” follow a similar task pattern at operational level. The transformation rules are exploiting the information in the domain model by creating a basic task for each object attribute that is visible in the interface.

Since there are 4 classes of objects in this application a pattern-based transformational approach is very useful and saves a lot of specification work.

## CONCLUSION

In this paper, we investigated some task patterns at functional and planning layer that extend and refine the set of patterns identified in a previous work. In Table 2, a set of mapping rules is presented that are based on the notation described in [10]. Each mapping rule is expressed as a task pattern having a prefixed part, a sequence of unit tasks and a post fixed part.

We mentioned several key requirements for a task modeling tool supporting a transformational approach to user interface design. The purpose of such a tool is to provide with a pattern language for task modeling including pattern definition based on task and domain mappings, and operations with task patterns over the UsiXML task model.

We discussed several aspects influencing a transformational approach to user interface design. The relationships between domain objects and their centrality for the user tasks give a sort of directness of the user interface structure. There are also other aspects that are shaping a user interface, like the platform and modality for which is primarily developed as well as the context specific requirements.

Using a task modeling tool supporting the aforementioned requirements makes possible to produce more than a task model. In this respect we consider useful to include all the relevant information from the domain model in order to get an extended task+domain model. This extended task+domain model is the source model for the transformations from task and domain to the concrete user interface. As such, it could serve as a sort of abstract user interface that support model checking (including task – domain mappings), model simulation and reasoning about the user interface usability.

## ACKNOWLEDGEMENT

This work was carried on in the framework of the Eureka Cluster ITEA2 European project UsiXML (08026) funded under the PNCDI II Innovation Project 294E.

## REFERENCES

1. Alexander, C. *The Timeless Way of Building*. Oxford University Press, New York (1979).
2. Aquino, N., Vanderdonckt, J., Valverde, F., Pastor, O. Using Profiles to Support Model Transformations in Model-Driven User Interfaces Development. In *Proc. of CADUI'2008* (Albacete, 11-13 June 2008). Springer, Berlin (2008)
3. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J. A unifying reference framework for multi-target user interfaces. *Interacting with Computers* 15, 3 (2003), pp. 289-308.
4. Card, S.K., Moran, T.P., and Newell, A. *The Psychology of Human-Computer Interaction*, Lawrence Erlbaum Associates, Mahwah (1993).
5. Faure, D., Vanderdonckt, J. (Eds.). *Proc. of 1st Int. Workshop on User Interface Extensible Markup Language UsiXML'2010* (Berlin, 20 June 2010). Thales Research and Technology France, Paris (2010). ISBN 978-2-9536757-0-2.
6. Limbourg, Q., and Vanderdonckt, J. Multipath Transformational Development of User Interfaces with Graph Transformations. In *Human-Centered Software Engineering*, Human-Computer Interaction Series, Springer London, (2009).
7. Luyten, K., Haesen, M., Degrandtsart, S., Demeyer, S., Oestrowski, D., and Coninx, K. On stories, models and notations: Storyboard creation as an entry point for model-based interface development with UsiXML. In *Proc. of UsiXML'2010*. D. Faure, J. Vanderdonckt (Eds.), Thales Research, Paris (2010), pp. 1-9.
8. Paternò, F., and Santoro, C. One Model, Many Interfaces. In *Proc. of CADUI'2002*, Kluwer Academics, Dordrecht (2002), pp. 143-154.
9. Norman, D.A. Cognitive artifacts. In *Designing Interaction - Psychology at the Human-Computer Interface*. J.M.Carroll (Ed.), Cambridge University Press, Cambridge (1991).
10. Pribeanu, C. Tool support for handling mapping rules from domain to task models. In *Proc. of TAMODIA'2006*. K. Coninx, K., Luyten, K. Schneider (Eds.). Springer, Berlin (2007), pp. 16-23.
11. Pribeanu, C. Task Modeling for User Interface Design – A Layered Approach. *International Journal of Information Technology* 3,2 (2007), pp. 86-90.
12. Pribeanu, C. A usability assistant for the heuristic evaluation of interactive systems. *Studies in Informatics and Control* 18, 4 (2009), pp. 355-362.
13. Pribeanu, C., and Vanderdonckt, J. Exploring Design Heuristics for User Interface Derivation from Task and Domain Models. In *Proc. of CADUI'2002*. Kluwer, Dordrecht (2002), pp. 103-110.
14. Tran, V., Kolp, M., Vanderdonckt, J., Wautelet, Y., and Faulkner, S. Agent based user interface generation from combined task, context and domain models. In *Proc. of TAMODIA 2009*. Springer, Berlin (2010), pp. 146-162.

# A Model for Dealing with Usability in a Holistic Model Driven Development Method

Jose Ignacio Panach, Óscar Pastor, Nathalie Aquino

Centro de Investigación en Métodos de Producción de Software

Universitat Politècnica de València

Camino de Vera s/n, 46022 Valencia, Spain

{jpanach, opastor, naquino}@pros.upv.es

## ABSTRACT

Currently, the importance of developing usable software is widely known. For this reason, there are many usability recommendations related to system functionality (called functional usability features). If these functional usability features are not considered from the very early steps of the software development, they require many changes in the system architecture. However, the inclusion of usability features from the early steps in a traditional software development process increases the analyst's workload, who must consider not only features of the business logic but also usability features. In the Software Engineering community, holistic MDD methods are a solution to reducing the analysts' workload since analysts can focus all their efforts on the conceptual model (problem space), relegating the architecture design and the implementation code (solution space) to automatic transformations. However, in general, MDD methods do not provide primitives for representing usability features. In this paper, we propose what we call a Usability Model that gathers conceptual primitives to represent functional usability features abstractly enough to be included in any holistic MDD method.

## Author Keywords

Model-Driven Development, usability, conceptual model.

## General Terms

Design, Experimentation, Human Factors, Verification.

## ACM Classification Keywords

D2.2 [Software Engineering]: Design Tools and Techniques – *Modules and interfaces; user interfaces*. D2.m [Software Engineering]: Miscellaneous – Rapid Prototyping; reusable software. H5.2 [Information interfaces and presentation]: User Interfaces – *Prototyping*.

## INTRODUCTION

According to ISO 9126-1 [10], usability is a key issue in obtaining good user acceptance of the software [6]. Some authors have divided usability recommendations into two groups [11]: recommendations that only affect the interface presentation (e.g., a label meaning), and recommendations that affect the system functionality (e.g. a cancel function). This second type is called functional usability features, and it is the most difficult type to include in the software [11], since these features affect the system interface as well as the system architecture. For example, the feature Cancel aims to cancel the execution of a service. The implementation of this usability feature is not a single button in the in-

terface; on the contrary, this feature also affects data persistency and functionality.

There are several authors in the Software Engineering community that have identified functional usability features and have proposed methods to include them in software development [8]. All these works propose including functional usability features from the early steps of the software development process, since they involve many changes in the system architecture if they are considered only when interfaces are designed. However, these approaches have some disadvantages:

- **Cost/benefit ratio:** The analyst must deal with usability from the requirements capture step until the implementation throughout the entire development process. This increases the analyst's effort and the cost/benefit ratio is not always favourable for features that are difficult to implement [11].
- **Changeable requirements:** Usability requirements (like other system requirements) are continuously evolving [12] and the adaptation to new requirements can involve a lot of rework in the system architecture.
- **Dependency on the implementation language:** The architecture design depends on the language used in the implementation and on the target platform.

Our research work is based on the idea that the Model-Driven Development (MDD) method is a suitable solution for reducing all these disadvantages [21][20]. MDD proposes that the analyst must focus all their efforts on building a conceptual model that represents all the system features (a holistic conceptual model) [17]. What we mean by "holistic" is that the conceptual model must include all the relevant systems perspectives: class structure, functionality and interaction. While class structure and functional view are supported in almost all the existing MDD approaches, interaction view is usually designed once the architecture has been finished. Our work focuses on modelling interaction adequately, at the same level of data and functionality, to provide a full system description in the conceptual model.

These complementary perspectives must provide the corresponding conceptual primitives, which are modelling elements having the capability of abstractly representing a feature of the system. Examples of conceptual modelling elements for the class structure view are classes of a class dia-



gram, attributes, and services. Examples of conceptual modelling elements for the functional view include service pre/post conditions, valid states and transitions. In this paper, our intention is to focus on conceptual modelling elements for the interaction view, and in particular, for functional usability features. The holistic conceptual model can then be seen as the input for a model compiler that can generate the software application automatically (or semi-automatically, depending on the model compiler capacity). Providing such a MDD-based software production environment, the problems of existing approaches that deal with functional usability features can find an adequate solution.

There are currently several MDD methods which model full functional systems, such as WebRatio [1], AndroMDA [2], and OO-Method [19] among others. However, none of them can model most functional usability features in their conceptual model. These features must be manually implemented, inheriting all the disadvantages of manual development mentioned above. This paper aims to extend existing MDD methods with conceptual primitives that represent functional usability features well known in the Human Computer Interaction community [11]. All these primitives are gathered in what we call the Usability Model.

The paper is structured as follows. Section 2 presents the state of the art. Section 3 describes the functional usability features used in our proposal. Section 4 explains our proposed Usability Model with its primitives. Section 5 describes how to include the Usability Model in a holistic MDD method using OO-Method as example. Finally, section 6 presents some conclusions and future work.

## STATE OF THE ART

If we look for existing proposals that deal with usability in MDD, we notice that, currently, there are not many works in the literature. Two examples of authors that propose considering usability in MDD methods are Taleb [24] and Gull [9]. The main disadvantage of both proposals is that these authors do not specify the usability traceability among the different development steps. Moreover, a specific notation to represent usability features in each step does not exist.

There are also works that propose integrating HCI techniques in MDD, such as Wang [27]. Wang proposes a user-centred design where the users play an important role in modelling the interface. This work focuses only on usability features related to the interface display, not to the functionality. In contrast, Sottet [22] is an author that deals with usability considering functional usability features. This author investigates MDD mappings for embedding both usability description and control. For Sottet, a user interface is a graph of models, and usability is described and controlled in the mappings between these models. The main disadvantage of Sottet's proposal is that the analyst must specify the transformation rules for each system and this is not trivial.

Other proposals use existing models to represent usability features, such as Sousa [23]. Sousa has defined an activity-based strategy to represent usability goals. The main disadvantage of this proposal is that we cannot model how usability features are related to the system functionality. Other authors that represent usability in existing models are Tao [25] and Brajnik [4], who propose modelling usability by means of state transition diagrams. However, state transition diagrams are only able to represent interactions, so they cannot represent all the usability features.

There are also some works [7][15] related to measuring the system usability in MDD conceptual models. Fernandez [7] proposes a model to evaluate system usability from conceptual models. Molina [15] proposes measuring usability attributes focused on navigational models. The main shortcomings of these proposals is that many usability attributes are subjective, and therefore cannot be measured automatically without taking into account the user.

For instance, attributes related to the attractiveness sub-characteristic [10] cannot be measured by means of conceptual models. Therefore, the result of early usability evaluation is a sort of prediction.

After studying related works, we conclude that existing proposals for dealing with functional usability features in MDD present some problems when we want to include them in a real software development process. First, few works have a specific notation to represent functional usability features in a model, and existing notations do not cover the model of all the existing features. Second, it is not clear how to include usability features throughout the whole software development process since existing proposals do not specify the traceability among models.

## BACKGROUND: PROPERTIES OF FUNCTIONAL USABILITY FEATURES

In previous works, we have extracted the properties that are needed to configure a set of usability features [18] defined by Juristo [11] and called FUFs (Functional Usability Features). We chose these usability features from all the existing ones since FUFs are specific to management information systems, the target systems of our work.

Moreover, FUFs have templates to capture usability requirements that are very useful for identifying features' properties. In the FUF definition, each FUF has a main objective that can be specialized into more detailed goals named mechanisms. A detailed explanation of properties derived from FUFs is out of scope of this paper, but can be consulted in [13]. Next, we summarize the process we followed to extract these properties:

1. We defined **Use Ways (UW)** using the usability mechanisms description. Each usability mechanism can achieve its goal through different means. We called each such mean Use Way (UW). Each UW has a specific target to achieve as part of the overall goal of the mechanism. For example, the usability mecha-



nism called *System Status Feedback* aims to inform the user about the internal system state. We identified that this goal can be achieved in at least three use ways: (1) *Inform about the success or the failure of an execution* (UW\_SSF1): it informs if an action execution finished successfully; (2) *Show the information stored in the system* (UW\_SSF2): it shows information of the system before the user triggers an action; (3) *Show the state of relevant actions* (UW\_SSF3): it indicates which actions cannot be triggered in the current system state.

2. We defined **properties** for each UW. We called the different UW configuration options to satisfy usability requirements as Properties. For example, we identified that the *Inform about the success or the failure of an execution* Use Way is composed of two properties: (1) *Service selection*; (2) *Message visualization*. By means of *Service selection* the analyst can specify the service that will inform about the success or failure; by means of *Message visualization*, the analyst can specify how the message will be displayed (format, position, icon, etc.).

In previous works [18], we defined Use Ways and properties but their inclusion in an MDD method affected its conceptual model. This paper aims to define a generic approach that does not affect the existing conceptual model. This new approach is based on a Usability Model explained in the next section.

#### A USABILITY MODEL

This section presents the conceptual primitives needed to represent Use Ways extracted from FUFs. Since currently there is no a standard notation to represent usability features, we have used a set of primitives very similar to UML notation [26]. We use graphical elements already defined in UML but we have extended these elements with some textual descriptions. The new conceptual primitives are gathered in a model called Usability Model.

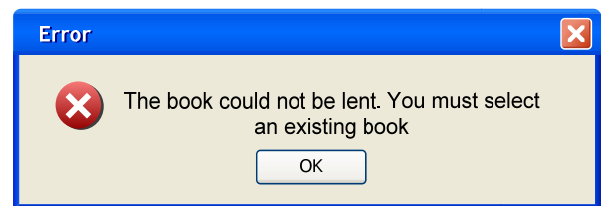
Next, we detail primitives to represent the set of 22 Use Ways [13] and their properties in a Usability Model. As an illustrative example to introduce these primitives, we use a system to manage a library. In the example, we aim to improve the system usability by means of the three Use Ways described above: UW\_SSF1, UW\_SSF2 and UW\_SSF3. We have focused our example in these three elements because the primitives needed to represent all their properties are enough to represent any other property of the remaining 19 Use Ways.

Table 1 shows the properties of the three Use Ways used in the example and their values. These properties must be specified by the analyst in order to develop a usable system. Next, we present how these three Use Ways improve the usability of the system to manage a library.

Use Way	Property	Value
UW_SSF1	Service selection	All the services
	Message visualization	Display the failure message textually
UW_SSF2	Dynamic information	Number of loans and number of penalties
	Static information	The labels “Loans” and “Penalties”
	Message visualization	Textually with Arial, black. Size 10
UW_SSF3	Service selection	Lend book
	Condition to disable	When the member is penalized
	Descriptive text	“The member is currently penalized”

**Table 1. Properties of UW\_SSF1, UW\_SSF2 and UW\_SSF3.**

1. UW\_SSF1 (*Inform about the success or the failure of an execution*): Each service of the system must inform whether or not the execution finished successfully with a textual message. For example, if the system does not support this Use Way, when the librarian tries to lend an inexistent book, she/he will not notice about the reason of the system failure. However, including UW\_SSF1, when the librarian tries to lend an inexistent book, the system displays an interface similar to Fig 1, detailing information about the reason of the mistake. According to Table 1, when the services specified in the property *Service selection* fail in the execution, an error message is displayed such as *Message visualization* indicates.



**Figure 1. Error derived from UW\_SSF1.**

2. UW\_SSF2 (*Show the information stored in the system*): When the librarian queries the list of members, she/he can choose a member and navigate towards the list of her/his current loans and towards the list of her/his penalties (buttons Loans and Penalties in Fig 2). The librarian would like to see how many loans and penalties the member has without performing the navigation. This functionality is supported with UW\_SSF2, which provides dynamic aliases for the navigation buttons by means of the property *Dynamic information*. Fig 3 shows the navigations buttons after including UW\_SSF2 in the

system. According to Table 1, navigation buttons aliases are composed of *Static information* (Loans and Penalties) together with *Dynamic information* (the number of loans and penalties for the selected member), and they are displayed such as *Message visualization* indicates.

id_Mem	Name	Surname	Age	Address
1	Mark	Smith	18	Liberty 45
2	James	Salt	37	King 23
3	Mary	Forrest	73	Street 34
4	Tony	Jones	30	Saint James 89
5	Robert	Samuel	23	Frank 29

**Figure 2. List of members with navigations towards Loans and Penalties.**



**Figure 3. Navigation buttons with dynamic aliases.**

- UW\_SSF3 (*Show the state of relevant actions*): If this Use Way is not included in the system, when the user tries to execute a service that cannot be triggered in the current system state, the execution results in a failure. For example, when the librarian executes the service to lend a book to a member that is currently penalized, the execution will finish unsuccessfully. Including UW\_SSF3, the button to lend a book is disabled if the member is penalized, avoiding the librarian to make a mistake. Fig 4 shows an example of an interface that has disabled the service to create a new loan, since the selected member is currently penalized. According to Table 1, if the *Service selection* (Lend book) has a *Condition to disable* (when the member is penalized) that becomes true, the service is disabled and a *Descriptive text* ("The member is currently penalized") is displayed.

**Figure 4. Disabling the button to execute an action when it cannot be triggered.**

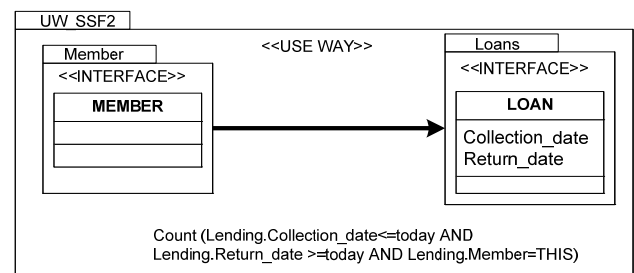
The main goal of our work is to demonstrate that these aspects related to usability improvement can and should be included in the conceptual schemas that are used in MDD environments, what is often just ignored. We need thus in-

corporate the corresponding set of conceptual primitives for that purpose. The primitives that compose the Usability Model are grouped into two levels: **packages** and **elemental primitives**. Packages are primitives that contain a set of other primitives (packages or elemental primitives). Elemental primitives constitute the building blocks from which packages are constructed. There are two types of packages in our Usability Model:

- First, for each Use Way, the analyst must define a package that groups all the primitives that define the Use Way. Each Use Way is represented by means of an element similar to a UML package whose name is the name of the **Use Way** with the label *Use Way*. Fig 5 shows an example to represent UW\_SSF2.
- Second, the analyst must define inside each package Use Way, the interfaces involved in the Use Way definition. Each interface groups the main interactive operations that the user can perform with the system. We propose defining interfaces by means of an element similar to a UML package with the label *Interface*. Fig 5 shows an example of two interfaces: *Member* and *Loans*.

Once we have defined the packages, the next step in our proposal is to define elemental primitives inside them:

- First, the analyst must define **navigations** in each Use Way with a property to navigate among several interfaces. These Navigations determine the target interfaces that can be reached from a source interface. For example, in Fig 2 the librarian can navigate towards the list of loans and penalties for a selected member. We propose specifying these navigations by means of an arrow with a source and a target. Fig 5 represents the primitives to define the navigation from member to loans displayed in Fig 2.



**Figure 5. Primitives to model UW\_SSF2 in the library management system.**

- Second, the analyst must specify attributes and services used in the properties of the Use Way. An attribute is an element used to ask the user for data or to query stored data. A service is an element that represents an action that can be executed by the user. Attributes and services are related to a class; therefore we propose modelling them according to the UML notation used to represent classes. Fig 5 shows the attributes of the class *Loan* used in the definition of the dynamic alias formula (explained in the next step).

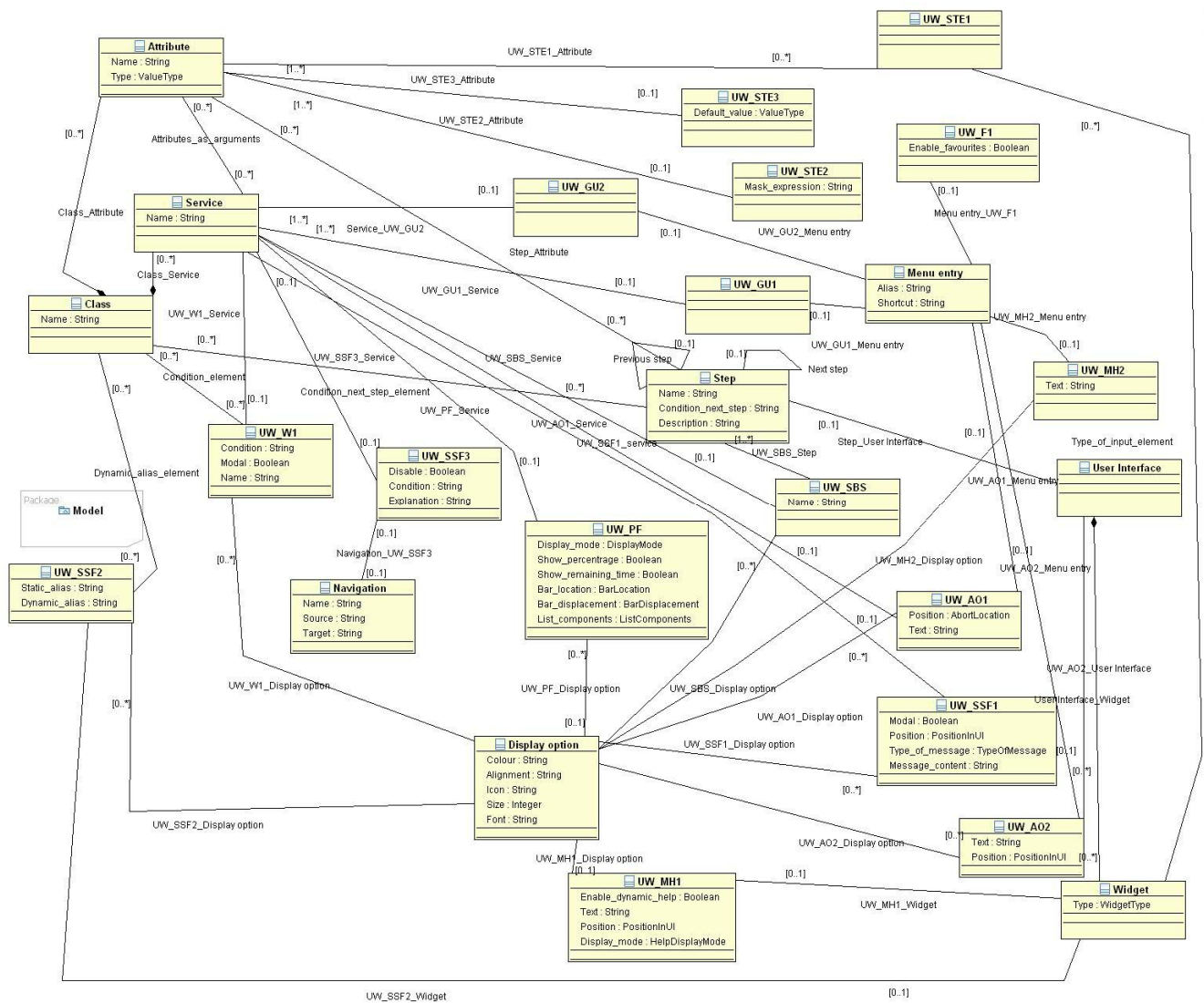


Figure 6. The Metamodel of the Usability Model.

- Third, the analyst must define **formulas** needed in Use Ways with properties that use conditions or dynamic information. These formulas are represented textually. For example, the formula to describe the dynamic alias used in the navigation towards Loans in Fig 2 has been textually defined in Fig 5.

```
<window id="1" name="Error_window" width="400" height="158">
  <outputText id="M1" name="Error_message" isVisible="true" isEnabled="true" isBold="true"
    textColor="#000000" value="The book could not be lent. You must select an existing book"/>
  <imageComponent id="I1" name="Error_icon" />
  <button id="OK1" name="Ok_Button" />
</window>
```

Table 2. Example of UsiXML to represent how the error window will be displayed.

- Finally, the analyst must specify how the interface will be displayed to the user. We have called this primitive display, and it is defined textually using the UsiXML notation [13] (User Interface eXtensible Markup Language), an XML-based markup language for defining user interfaces. Table 2 shows a piece of code of UsiXML to express how the window of Fig.1 is displayed.

These elements compose the suite of conceptual primitives used in the Usability Model. They are enough to represent any of the 22 Use Ways extracted from Juristo's FUFs. A description of the primitives needed to represent each Use Way is detailed in [15].

We have defined a usability metamodel to specify the properties of the Use Ways and how these properties are related to system functionality. The aim of the metamodel is to identify the elements needed to represent all the properties. All these elements can be represented with the conceptual primitives described above. The usability meta-

model is drawn in Fig. 6 (and also downloadable from [14]), where each Use Way is represented with a class with the prefix UW. Attributes and relationships represent properties of the Use Ways. For example, the class UW\_SSF2 that represents the Use Way *Show the information stored in the system*, represents the properties *Static information* and *Dynamic information* by means of two attributes. The property *Message visualization* is represented with the relation to the class *Display option*, which is a class to represent how the visual elements will be displayed in the interface. It is important to note that in the metamodel we can see how the properties are involved in the functional and interaction features.

For example, in UW\_SSF2, the property *Dynamic information* depends on the information stored in the system (which is a functional feature), therefore, we need to relate the class that represents UW\_SSF2 to the class that represents the system persistency, called Class in the metamodel. Moreover, there is a relationship with the class Widget, since the navigation alias (linking *Static information* and *Dynamic information*) is displayed in a button (which is an interaction feature).

Depending on the MDD method where we would like to include usability features, some of the primitives that compose the Usability Model can already be supported by the existing conceptual model of the MDD method. For example, if the MDD method has a model to represent the class structure, the primitives Attributes and Services are already supported. Next section explains how the Usability Model can be included in an existing MDD-based approach without affecting the models. In a first step, we propose extracting the information represented in existing primitives by means of model-to-model transformations. In a second step, the analyst models unsupported primitives in the Usability Model.

## THE USABILITY MODEL IN AN INDUSTRIAL MDD METHOD: A LAB CASE

### The Usability Model in a Holistic Method

This section explains how to integrate the Usability Model into a holistic MDD method without changing its existing conceptual model. As we commented above, by holistic we mean that all the relevant systems views (static, dynamic,

interaction) are properly incorporated into the used modeling strategy. Fig 7 represents graphically a summary of the process. In the example, there are two existing models in the conceptual model of the MDD method: one model to represent the system persistency and another model to represent the interaction. In this example, we have depicted only two models, but the number of models that the MDD method uses depends exclusively on the chosen MDD method. Moreover, the existing MDD method can support code generation from the conceptual model by means of a model compiler. The level of automation of this process also depends on the MDD method chosen. Some methods are automatic (generate full functional systems), and others semi-automatic (some manual implementation).

Our proposal to include the Usability Model in a holistic MDD method consists of three steps:

1. *Derivation of conceptual primitives defined in the existing conceptual model:* The notation of the Usability Model includes functionality, persistency, navigation and interaction elements that can be defined in other models of the MDD method (depending on the expressiveness of the MDD method chosen). For example, in Fig 7, the classes, attributes, and services needed in the Usability Model have been previously defined in the Class Model. Elements that have been previously defined in the existing conceptual model do not need to be defined again in the Usability Model. In this first step, we automatically extract primitives defined in other models and include them in the Usability Model (model-to-model transformations).
2. *Modelling unsupported conceptual primitives:* Once supported primitives have been automatically derived, the analyst must manually specify properties that are not supported by existing models.

We propose performing these transformations with ATL [3], which is a language to specify transformations using a source metamodel and a target metamodel. In the example of Fig 7, the source metamodels are the metamodel of the Class Model and the metamodel of the Task Model, while the target metamodel is the metamodel of the Usability Model.

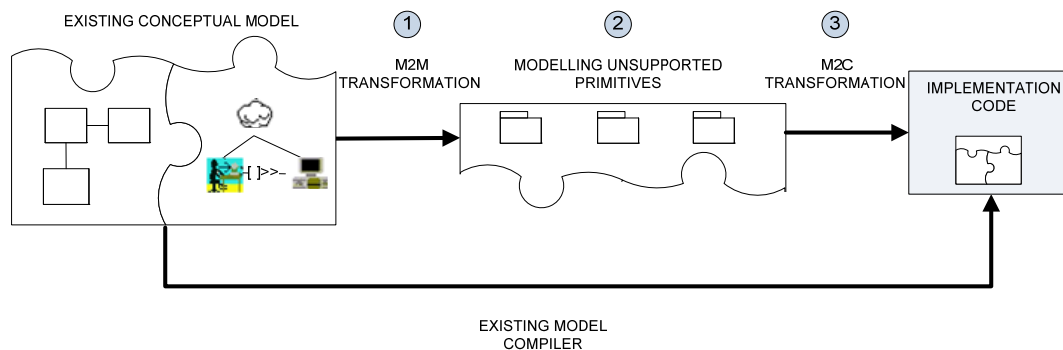


Figure 7. An overview of the process to integrate the Usability Model in a holistic MDD method.



3. *Code generation*: Once the Usability Model has been fully defined, we can generate code from this model by means of automatic Xpand transformations [28] (model-to-code). The code generated from the Usability Model can be combined with the code generated by the model compiler, which generates code from the existing conceptual model. In the end, the final code includes all the elements specified in the existing conceptual model and in the Usability Model.

It is important to mention that ATL and Xpand transformations must be defined once only for a specific MDD method since these transformations based on metamodels are valid for developing any system. Therefore, steps 1 and 3 can be executed automatically by means of transformation templates.

#### A Lab Case with OO-Method

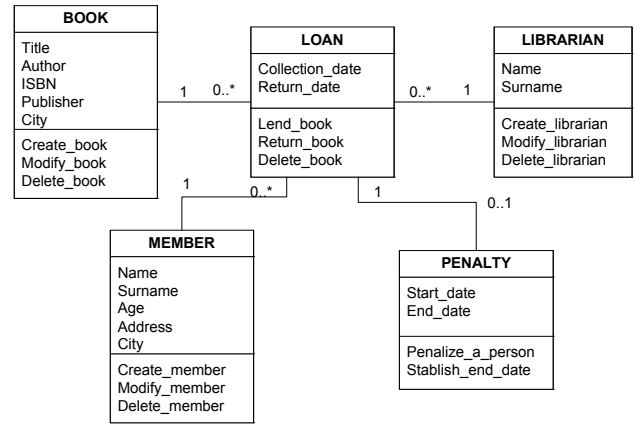
In order to demonstrate that our proposal works for a real software development process, we have used OO-Method [19]. OO-Method has been successfully implemented in industry with a tool called OLIVANOVA [5], which can generate full functional systems automatically from a conceptual model. This is the reason why we have chosen OO-Method as proof of concept for our proposal. The OO-Method conceptual model is composed of four complementary models:

1. **Object Model**: Specifies the system structure in terms of classes of objects and their relations. It is modelled as an extended UML [26] class diagram.
2. **Dynamic Model**: Represents the valid sequence of events for an object.
3. **Functional Model**: Specifies how events change object states.
4. **Presentation Model**: Represents the interaction between the system and the user [16]. This model represents the interface by means of Interaction Units. Moreover, this model represents Elementary Patterns that will be displayed inside the interfaces, such as masks, filters, or navigations, among others.

With regard to the system used in the lab case, we use the system to manage a library. This example is simple enough to facilitate the understanding of our proposal. Fig 8 shows the OO-Method Object Model of the system.

The other three models that compose the conceptual model of OO-Method (Dynamic, Functional and Presentation) are not displayed for space reasons. Next, we explain how to model UW\_SSF1 (*Inform about the success or the failure of an execution*), UW\_SSF2 (*Show the information stored in the system*) and UW\_SSF3 (*Show the state of relevant actions*) for developing the library management system in OO-Method.

The first step of our proposal consists of extracting information from the existing primitives.



**Figure 8. Object Model of the system to manage a library.**

From the list of properties of the three Use Ways (Table 1), we can extract from the OO-Method's conceptual model the followings:

- **UW\_SSF1**: *Service selection* can be derived from the Object Model.
- **UW\_SSF2**: *Static information* of navigations can be derived from the Presentation Model.
- **UW\_SSF3**: *Service selection* and *Condition to disable* can be derived from the services of the Object Model and their preconditions. A precondition in the Object Model is a condition that must be satisfied to execute a service.

In order to derive these properties from existing OO-Method models, we have used ATL transformations. The source metamodels are the four metamodels that define the four conceptual models of OO-Method (object, dynamic, functional and presentation); and the target metamodel is the metamodel of the Usability Model. Next, we show the ATL transformation rule that generates a first version of the Usability Model to represent UW\_SSF1 when the service execution finishes with a failure.

From the two properties, we can only derive *Service selection* from the Object Model since OO-Method does not have any model to represent *Message visualization* (the Presentation Model does not have primitives to represent this property). The ATL rule is simple in order to be as illustrative as possible and avoid technical terms. We have defined similar rules to extract supported primitives of UW\_SSF2 and UW\_SSF3.

```
rule Service2UWSSF1Failure{
  from
    b: Object!Service
  to
    e: Usability!Service (Name <- b.Name) }
```

Once we have extracted the properties supported by the OO-Method's conceptual model, the second step of our proposal is to complete the Usability Model with unsupported conceptual primitives. Next, we detail how to complete each Use Way in the Usability Model. Primitives that are automatically extracted from existing models are drawn on grey background in the figures. Primitives added manually are drawn on white background.

For UW\_SSF1, the property *Service selection* has been extracted from the Object Model automatically (in the first step). If the users want to visualize failure messages for the service Lend\_book like Fig 1, we must model the property *Message visualization* with the values shown in Table 2. Fig 9 shows the Usability Model for representing UW\_SSF1. The property *Service selection* is represented with the primitive Service and the property *Message visualization* with the primitive Display. The primitives Use Way and Interface are generated from scratch in the ATL transformation.

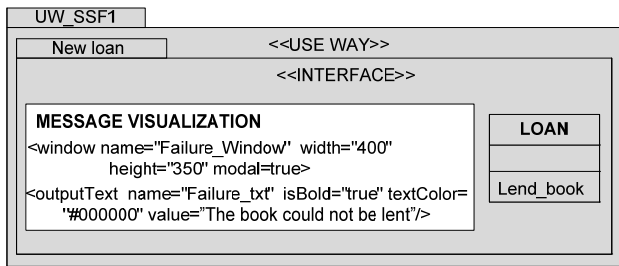


Figure 9. Model to represent UW\_SSF1.

The second usability feature that must be included in the library system is to provide dynamic labels in navigation

buttons to display how many loans and penalties a selected member has (UW\_SSF2). As we have commented above, navigations, their static aliases and interfaces can be extracted from the OO-Method Presentation Model (in the first step). In the second step, the analyst must specify the properties *Dynamic information* and *Message visualization* since these elements are not supported by the OO-Method Presentation Model. Fig 10 shows the Usability Model for UW\_SSF2. The property *Static information* is represented with the primitive Navigation and Interface, *Dynamic information* is represented with the primitives Formula and Attributes, and *Message visualization* is represented with the primitive Display. The primitive Use Way is generated from scratch in the ATL transformation.

The third usability feature to include is for disabling the service Lend\_book when the member currently has a penalty (UW\_SSF3). The properties *Service selection* and *Condition to disable* have been extracted from preconditions of the Object Model (in the first step). The aim of the preconditions is to trigger an error if the user tries to execute a service when a condition is not satisfied. Therefore, the definition of these preconditions can be used to know when to disable the service in order to avoid a user mistake. In this second step, the analyst must only specify the descriptive text that will be shown when the service is disabled (property *Descriptive text*). Fig 11 shows the model to disable Lend\_book when the member currently has a penalty. The property *Service selection* is represented with the primitive Service, *Condition to disable* with the primitives Formula and Attributes and *Descriptive text* with the primitive Display.

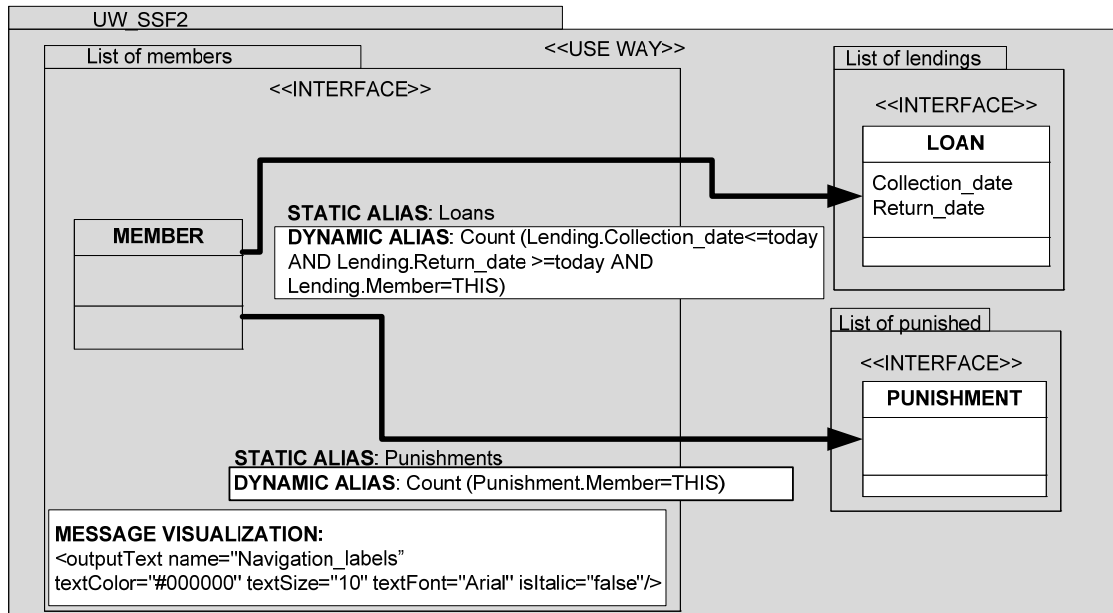
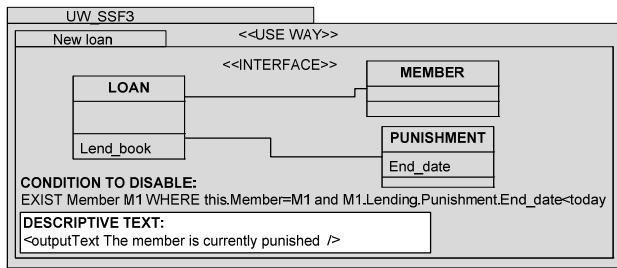


Figure 10. Model to represent UW\_SSF2.





**Figure 11. Model to represent UW\_SSF3 to disable Lend\_book.**

Finally, in the third step, the Usability Model must be transformed into code that implements all the characteristics represented in it. This transformation is performed with Xpand [28]. The code derived from the Usability Model must be included in the code generated with the OO-Method model compiler. Below, we show a small chunk of Xpand code used in the transformation from UW\_SSF1 into Java code.

```
«DEFINE javaClass FOR Class»
    «FILE Class.name+".java"» public class
«name» {
    «FOREACH service1 AS s»
        DisplayOption show
        public void «s.name»(
            «FOREACH attribute1 AS a»
                «a.type» «a.name»
            «ENDFOREACH»
        «IF s.uwSsf11 != null» If ("error")
        show.display(«s.uwSsf11.modal», «s.uwSsf11.position»,
        «s.uwSsf11.typeOfMessage»,
        «s.uwSsf11.messageContent»);
        «ENDIF»
    «ENDFOREACH»
«ENDFILE» «ENDDDEFINE»
```

Fig 1, Fig 3 and Fig 4 show screenshots of the system to manage a library developed with OLIVANOVA after including UW\_SSF1, UW\_SSF2 and UW\_SSF3 respectively. Therefore, we can state that our proposal can be successfully applied to an industrial MDD method.

## CONCLUSION

The contribution of this paper is the definition of a Usability Model to deal with usability features in a holistic MDD method. We have defined conceptual primitives to represent usability features defined by Juristo for management information systems and we have gathered them in a Usability Model. It is important to note that there are many other non-functional usability features that are out

of scope of this paper, such as, understandability or attractiveness. Moreover, systems of other areas such as multimedia applications or virtual reality systems are out of scope too. The main advantages of our proposal with regard to existing proposals to deal with usability in MDD are: (1) The Usability Model can represent most functional usability features for a management information system (we can ensure that it supports all the FUFs defined by Juristo); (2) The notation used in the Usability Model has an unambiguous syntax and semantics, which allows transformations to be performed; (3) The Usability Model can be used in any MDD method (we have used OO-Method as example).

We have learned some lessons applying the proposal to OO-Method: First, the difficulty of writing ATL and Xpand transformations depends exclusively on the MDD method chosen. OO-Method generates the whole system, but MDD methods with less powerful model compilers need more effort to define transformations. However, it is important to mention that these transformations are defined only once and can be used indefinitely in every software development. Second, the existence of a Usability Model does not ensure that generated systems are usable. The analyst must follow usability guidelines to combine the primitives properly. As future work, we plan to define metrics to measure the usability of the system based on the conceptual primitives of the Usability Model. Moreover, we plan to measure the effort required to implement this approach in an MDD method. This measure will be done considering analysts who know previously FUFs and analyst who do not know them yet.

## ACKNOWLEDGMENTS

We gratefully acknowledge the support of the ITEA2 Call 3 UsiXML project (20080026) and financed by the MI-TYC under the project TSI-020400-2011-20; the MICINN under the project PROS-Req (TIN2010-19130-C02-02) co-financed with ERDF; the Generalitat Valenciana under the project ORCA (PROMETEO/2009/015).

## REFERENCES

1. Acerbis, R., Bongio, A., Brambilla, M., and Butti, S.: WebRatio 5: An Eclipse-Based CASE Tool for Engineering Web Applications. LNCS, vol. 4607. Springer, Berlin (2007), pp. 501-505.
2. AndroMDA, <http://www.andromda.org/>.
3. ATL: <http://www.eclipse.org/atl/>
4. Brajnik, G.: Is the UML appropriate for Interaction Design? Università di Udine (2010) 6.
5. CARE Technologies S.A. <http://www.care-t.com>
6. Davis, F.D. User acceptance of information technology: system characteristics, user perceptions and behavioral impacts. *Int. Journal Man-Machine Studies* 38 (1993), pp. 475-487.
7. Fernández, A., Abrahao, S., and Insfran, E. A Web

- Usability Evaluation Process for Model-Driven Web Development. In *Proc. of 23<sup>rd</sup> Int. Conf. on Advanced Information Systems Engineering CAiSE'2011*. Springer, London (2011), pp. 108-122
8. Folmer, E. and Bosch, J. Architecting for usability: A Survey. *Journal of Systems and Software* 70, 1 (2004) pp. 61-78.
  9. Gull, H., Azam, F., and Iqbal, S.Z. Design of Novel Usability Driven Software Process Model. *Int. Journal of Computer Science and Information Security* 8 (2010) pp. 46-53.
  10. ISO/IEC 9126-1, Software engineering - Product quality - 1: Quality model (2001).
  11. Juristo, N., Moreno, A.M., and Sánchez, M.I. Analysing the impact of usability on software design. *Journal of Systems and Software* 80 (2007), pp. 1506-1516.
  12. Lawrence, B., Wiegers, K., Ebert, C.: The top risk of requirements engineering. *IEEE Software* 18 (2001), pp. 62-63.
  13. Limbourg, Q. and Vanderdonckt, J. UsiXML: A User Interface Description Language Supporting Multiple Levels of Independence. In *Engineering Advanced Web Applications*, M. Matera, S. Comai, S. (Eds.). Rinton Press, Paramus (2004), pp. 325-338.
  14. List of Use Ways and Properties, <http://hci.dsic.upv.es/UsabilityModel/UseWaysList.html>
  15. Molina, F. and Toval, A. Integrating usability requirements that can be evaluated in design time into Model Driven Engineering of Web Information Systems. *Advances in Engineering Software* 40 (2009), pp. 1306-1317.
  16. Molina, P.J., Meliá, S., and Pastor, Ó. JUST-UI: A User Interface Specification Model. In *Proc. of 4<sup>th</sup> Int. Conf. on Computer-Aided Design of User Interfaces CADUI'2002* (Valenciennes, June 2002). Kluwer Academics, Dordrecht (2002), pp. 63-74.
  17. Olive, A. Conceptual Schema-Centric Development: A Grand Challenge for Information Systems Research. In *Proc. of the 16th Int. Conf. on Advanced Information Systems Engineering*. LNCS, vol. 3520, Springer-Verlag, Berlin (2005), pp. 1-15.
  18. Panach, J.I., España, S., Moreno, A., and Pastor, Ó. Dealing with Usability in Model Transformation Technologies. In *Proc. of ER'2008*. LNCS, vol. 5231. Springer, Berlin (2008), pp. 498-511.
  19. Pastor, O. and Molina, J.C. *Model-Driven Architecture in Practice*. Springer, Berlin (2007).
  20. Selic, B.: The Pragmatics of Model-Driven Development. *IEEE software* 20 (2003) 19-25
  21. Sendall, S., Kozaczynski, W.: Model Transformation: The Heart and Soul of Model-Driven Software Development. *IEEE Software* 20 (2003) 42-45.
  22. Sottet, J.-S., Calvary, G., Coutaz, J., and Favre, J.-M. A Model-Driven Engineering Approach for the Usability of Plastic User Interfaces. In *Proc. of Engineering Interactive Systems EIS'2007* (2007), pp. 22-24.
  23. Sousa, K., Mendonça, H., and Vanderdonckt, J. Towards Method Engineering of Model-Driven User Interface Development. In *Proc. of TAMODIA'2007*. LNCS, vol. 4849. Springer, Berlin (2007), pp. 112-125.
  24. Taleb, M., Seffah, A., and Abran, A. Investigating Model-Driven Architecture for Web-based Interactive Systems. *Int. Journal on Human-Computer Interaction* 2 (2010).
  25. Tao, Y. An Adaptive Approach to Obtaining Usability Information for Early Usability Evaluation. *IMECS* (2007), pp. 1066-1070.
  26. UML: <http://www.uml.org/>
  27. Wang, X., Shi, Y.: UMDD: User Model Driven Software Development. In *Proc. of IEEE/IFIP Int. Conference on Embedded and Ubiquitous Computing*, (Shanghai, 2008).
  28. XPAND, <http://www.eclipse.org/modeling/m2t/?project=xpand>

# Proposal of a Usability-Driven Design Process for Model-Based User Interfaces

Eric Montecalvo, Alain Vagner, Guillaume Gronier

Public Research Centre Henri Tudor

29, av. J.F. Kennedy L-1855 Luxembourg-Kirchberg

eric.montecalvo@tudor.lu, alain.vagner@tudor.lu, guillaume.gronier@tudor.lu

## ABSTRACT

Nowadays, user interface design is becoming more and more complex, particularly with the proliferation of new mobile devices, available modalities, context-awareness capabilities and so on. However, in this context, the quality of User Interfaces (UIs) must be preserved. To cope with this situation and to meet the needs of UI designers to reason at a higher level of abstraction during the design step, different solutions have been proposed, such as the use of models describing the aspects to consider when designing UIs (e.g. user tasks, interaction model, context model etc.) and User Interface Description Languages (UIDL). UsiXML (USer Interface eXtensible Markup Language) is one of them and defines a set of relevant models to support the design of UIs at different levels of abstraction. Moreover, it is compliant with a Model-Driven Engineering (MDE) approach. The usability of the UI is an important part of the quality of interactive systems, and we have previously proposed a generic method, named GENIUS and inspired by the user-centred design approach, to try to improve the quality of UI within a model-driven interface approach. This paper is focused on the first two steps of the method, which are the modelling of interactive systems and transformation between models. During these steps, we propose a means to take into account usability criteria in a pragmatic way.

## Author Keywords

Ergonomics, graphs, Model-Driven Engineering, models, ontologies, transformations, usability, user interfaces.

## General Terms

Design, Experimentation, Human Factors, Verification.

## ACM Classification Keywords

D2.2 [Software Engineering]: Design Tools and Techniques – *Modules and interfaces; user interfaces*. D2.m [Software Engineering]: Miscellaneous – Rapid Prototyping; reusable software. H5.2 [Information interfaces and presentation]: User Interfaces – *Prototyping*.

## INTRODUCTION

The purpose of Model-Based User Interfaces (MBUI) is to offer the designer several high-level models to design and analyse an interactive system rather than starting to implement it upfront. Thus, the focus is given to the important aspects of the interactive system such as task analysis, context of use, graphical interactions, etc.

The Cameleon Reference Framework [2] (Figure 1) pro-

vides a consensus on the types of UI models used for the different levels of abstractions and to support the changes of the context of use. It defines the tasks and concepts model, the Abstract User Interface model (AUI), the Concrete User Interface model (CUI) and the Final User Interface (FUI). UI designers can use these models as a means to design the UI following a MDE approach. The UsiXML models, implementing the Cameleon framework, are used in the modelling steps of the GENIUS method. This paper is dedicated to proposing a way, based on ontologies, for the integration of usability inputs throughout the design steps. The GENIUS method is developed within the Luxembourgish competitive research project GENIUS (model driven Generation of ErgoNomic User interfaces).

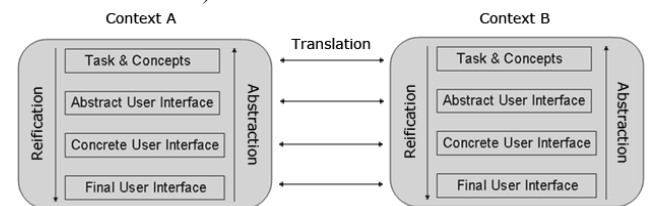


Figure 1. Cameleon Framework (simplified) and Transformations.

The first section of this paper provides a brief overview of the GENIUS method [7] and the ontological approach. The next section introduces how ontologies and related tools could be used as a way to integrate usability inputs. The last section deals with some examples about how to use this approach to help UI designers in the modelling step of the interactive system.

## THE GENIUS METHOD AND THE MODEL-BASED USER INTERFACES DESIGN PROCESS USING ONTOLOGIES

Previous works [4][18] investigate the use of MDE for embedding both the description and control of usability criteria in models. They also offer a general definition of transformations and mapping between models, which are involved in the modelling steps, and identify them to be of key importance for reasoning on usability. In order to try to reach a better usability within a MDE approach, the GENIUS method (Figure 2) attempts to identify where and how to take into account usability inputs and the usability validation process. In the GENIUS method, one of the main development paths considered is forward engineering, going from the high-level abstraction to the lower-level abstraction of the interactive system.

### The GENIUS Method

The method proposed is based on an iterative process made from four successive major steps. This iterative process is inspired by the User-Centred Design (UCD) method [9], since the involvement of the users is a key concept. The first step is related to the modelling (e.g. user tasks, abstract UIs, concrete UIs etc.). The second step consists of generating UIs based on the modelling, taking into account usability inputs. The third step allows multiple users to use and test the produced UIs and automate the feedback channel with semantic logs of the use of UIs. Finally, the fourth step consists in analysing the feedbacks to enhance or adapt the designed models in the first step.

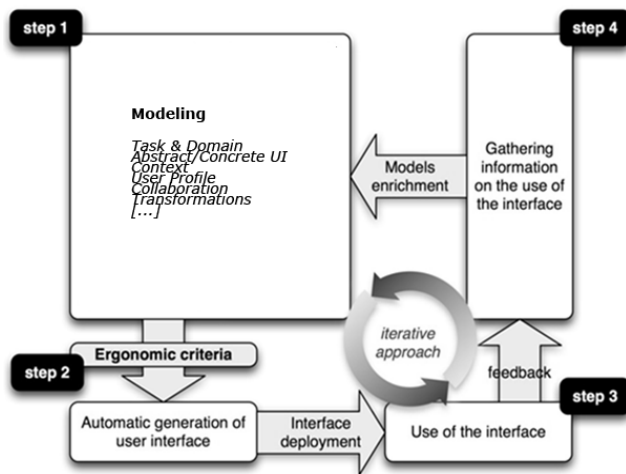


Figure 2. The GENIUS method.

In this paper, the focus is given on the two first steps, wherein the different stages defined in the MDE approach are performed and where the usability models are used in order to help designers having a predictive viewpoint on usability during the modelling of an interactive system.

### What are ontologies?

An ontology mainly consists of the specification of a conceptualisation [8]. Ontologies may be used for modelling and metamodeling. Several languages exist to describe ontologies, like DAML, OIL, KIF and OWL. In the rest of this paper, we focus on semantic web technologies and thus use Web Ontology Language (OWL), as it is the standard to define ontologies on the Web. As OWL provides us with important flexibility and expressiveness degrees, it is possible to express languages such as Unified Modelling Language (UML) in OWL [15]. Since metamodeling, models transformations and productive models are native features of the semantic web technologies considered, it allows us to start the deployment of our MDE approach.

### Benefits of ontologies

Ontologies allow the creation of links between different and distributed models at runtime. These links are generic

and are considered as additional statements. Anyone has the possibility to extend, annotate and redefine existing models on the Web. In this frame, the semantic Web promotes the “open world assumption” for dealing with knowledge incompleteness: The truth value of a statement is independent of the fact that it is known. So, it is a very different situation than in data silos, in which the knowledge coherence is maintained at a local level and is based on the “closed world assumption”: Any statement that is not known to be true is false. “Open world assumption” and links between models are real enablers for a distributed inference on several models expressed with semantic web technologies.

For example, a designer can choose to extend standard UsiXML models described using ontologies by using private models that express his preferences and experience. These models add knowledge that will be taken into account by inference engines and model transformations. The links between models could also be established by the system as a means to trace the origin of concepts after transformations.

Moreover, in the semantic web ecosystem, the tooling is hectic and stable, such as ontology design tools (e.g. Protégé [22]), knowledge bases (e.g., Virtuoso [23]), inference engines (e.g., Hermit [24]), querying system (e.g. Protocol and RDF Query Language – Description Logic [SPARQL-DL] [25]), etc. These different components are used as a technological framework supporting the needs of our project.

### Transformations within ontologies

Schaefer provides us with a large review of transformation tools for MBUI [16], such as graph transformations, ATL, TXL, 4DML, UIML Peers, XSLT, GAC and RDL/TT. The comparison criteria between these tools takes into account the relevant features within the MBUI design context. In these comparisons, graph transformations appear to be associated to UsiXML since it is specifically designed for the multi-path development of UIs and it is one of the first approaches which have been proven to be MDA-compliant [19]. Moreover, in order to classify the different model transformations, an extensive taxonomy has been proposed by Czarnecki and Helsén [3]. The concepts of graph transformation support different kinds of transformations such as abstraction, reification and translation between models (and traceability). These transformations are particularly interesting in regard to the defined method and goals of the GENIUS project.

Transformation rules can be expressed as graph patterns. By taking advantage of the potential of the ontology reasoners and querying systems, we try to detect potential usability problems in a predictive way and to associate suggestions directly in the modelling and transformations steps. A transformation is performed by a set of transformation functions using mappings properties to provide

the designer with a means for selecting the most appropriate transformation between models. We make the assumption that it seems to be an easy way for both describing and controlling usability criteria during the design time.

### Ontologies and Model-Based User Interfaces

Following a declarative approach, the transformations of models can be expressed as graph transformations, because models can be designed with an underlying graph structure. In this way, the expressions of transformations and mappings can be used for graph rewriting (addition, modification or deletion of elements of graphs, and so the elements of models.)

Generally in the conventional model-driven approach to UI generation, transformation rules are integrated within tools and consequently the links between models can be quite difficult to control. Ideally, a designer should be able to alter or redefine these rules and choose between different possible propositions of transformations. To address these issues, one of the current directions for the GENIUS project is to use ontology-based tools in order to express models, rules and inferences.

### USABILITY KNOWLEDGE INTEGRATION AND USES

Human-Computer Interactions are part of our everyday life, from computers to mobile phones, and in numerous artefacts. There is a proliferation of devices, acting in more and more different contexts and providing specific interactions (e.g., touchscreen). Also, with the cloud computing trend, more and more services are available from different connected mobile devices. This propagation of electronic devices tends to complicate interaction paradigms of these interactive systems that lead global usability issues.

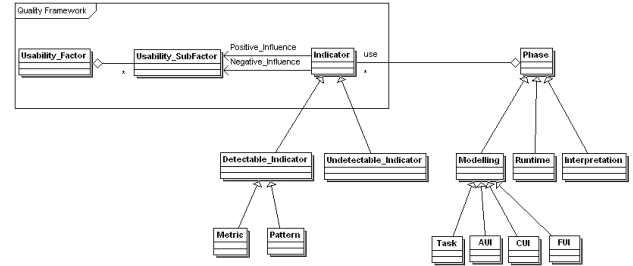
#### Usability resources

Since the beginning of work in psychology about usability, numerous methods were developed in order to correct usability problems and to design usable applications. We are focusing here on usability heuristics, which consist of a set of guidelines or rules for the a priori evaluation of user interfaces. In the literature, we can find several usability heuristics specifications, the more well-known being Nielsen's [20] and Bastien & Scapin [21] heuristics. More recent works propose to formalise these usability heuristics into a unified model [17]. A recent study on the calculable metrics of a Concur Task Tree (CTT) model is also presented in [14].

#### Usability metamodel definition

We saw that there are lots of references about quality models, usability criteria or guidelines in the literature. Most of them propose an organisation of usability knowledge according to a set of criteria and sometimes associated metrics. In our experimentation, we have defined a simple metamodel (Figure 3) to keep a compatibility with these different sources. The purpose was not to

define a comprehensive metamodel, but to allow the instantiation of main elements coming from these sources and considered as essential in our approach. So, to begin with, we have integrated the key elements in common in our metamodel.



**Figure 3. A Basic Usability Metamodel (UML notation).**

To simplify the navigation of the UI designer within the viewpoint on usability during the modelling step, only two hierarchical levels of usability concept were kept: The factors, which are the main generic characteristics of the usability concept (e.g. effectiveness, efficiency, learnability etc.); and subfactors, which are the measurable usability criteria and can be linked to one or more factors.

An indicator is a generic element giving information on a subfactor. The information is about the influence of the indicator on each subfactor and can be either a positive or negative influence as proposed by Jameson [10][11].

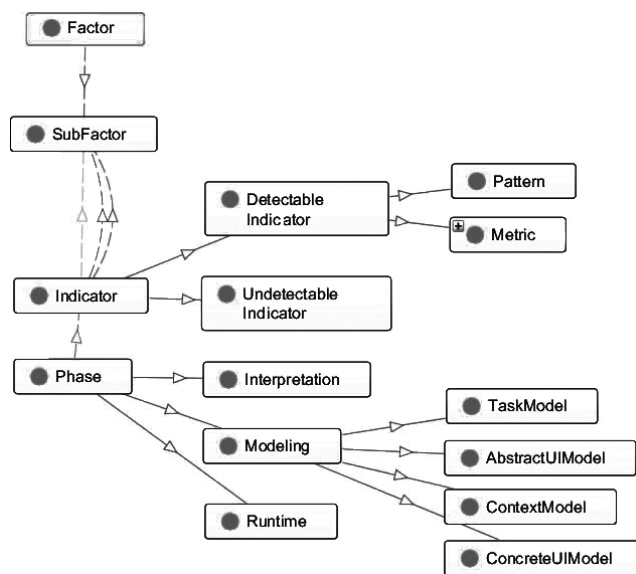
An indicator can be a metric (composed of values, ranges, thresholds etc.) calculated from the current state of modelling, or a generic pattern of structured elements, which should be matched also during the modelling step. The patterns may be used to propose a transformation of the matched elements, directly as a graph rewriting in the designed models of the interactive system.

We also consider the notion of phase to specify the step wherein the indicator can be obtained (e.g. Platform-Independent Model (PIM) / Platform-specific model (PSM) [5], at the runtime on the use by end users, etc.). This notion is mainly used to classify the indicators within specific steps of the GENIUS method and it is used by the system implemented to determine if an indicator has to be calculated for a given step.

As mentioned before, our experimentation is based on ontologies to represent all concepts of models, so we have translated this metamodel from the UML notation to an OWL representation (Figure 4) in order to use the capabilities of querying systems, reasoners and other associated tools.

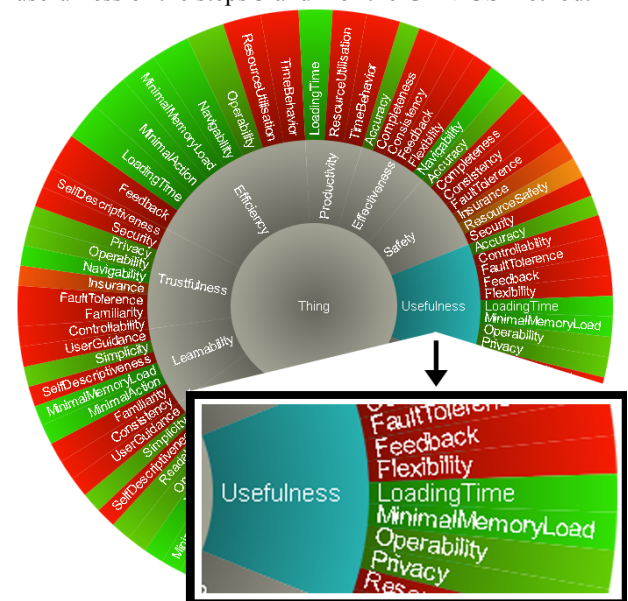
At this time, to exploit a usability model, we have chosen to use an instantiation of Quality in Use Integrated Measurement (QUIM) model defined in [17], which consists of a consolidated model of usability factors and the associat-

In the next section we present several aspects of this process, most particularly the usability overview of the UI and the implementation of usability metrics, patterns and transformations.



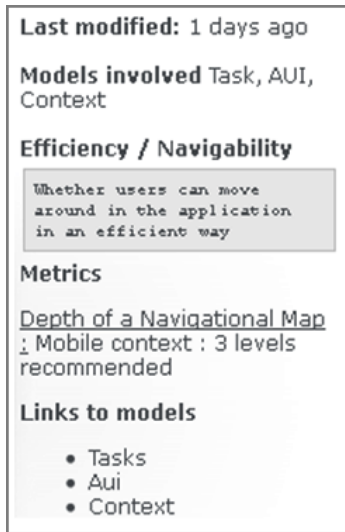
Here is an example of visualisation implemented for the

Associated with a colour code, it allows a quick overview of the usability state according the measured metrics and matched patterns. The colour code (from red to green or from dark to light for a monochrome version of this paper) used for the sunburst items is relative to the ratio of positive influence indicators over the negative influence indicators associated with each subfactor. Thus for a given criterion, the number of indicators having a positive influence is the sum of reached metrics, the number of positive patterns matched, and the number of suggested transformations which are accepted by the designer. The number of indicators having a negative influence is the sum of unreached metrics, the number of negative patterns matched, and the number of suggested transformations which are not accepted. Of course this representation and the calculation of the ratio may be personalised (e.g. coefficient relative to given priorities, etc.) and cannot be considered as a final validation of the real usability level of the designed interactive system. This justifies the usefulness of the steps 3 and 4 of the GENIUS method.



**Figure 5a. Usability viewpoint visualisation (left side).**





**Figure 5b. Usability viewpoint visualisation (right side).**

So, the sunburst is built with data coming from the triple store [23], a knowledge database. Next, the statements are translated to a JSON representation, in the expected format for the advanced JavaScript visualisation toolkit used, TheJit [26].

The right panel (Figure 5b) shows information relative to the selected subfactor. It contains the relevant information that can be useful for the designer in making choices during the modelling and activate the suggested propositions of transformation. This panel contains the list of current designed models involved in the selected subfactor (the usability criterion), the definition of the factors and sub-factors, the list of calculated metrics and matched patterns, their description and the links to each part of the designed interactive system.

Basically, the underlying idea is to provide the designer (who may have different profiles like developer, UX designer etc.) with an easy way to visualise, find and understand how the usability viewpoint is measured. Thus, during the modelling step, all the changes of models provide dynamic feedback on the positive and negative influences of indicators among usability factors. Moreover, this panel provides him with recommendations to correct the part of models linked to matched pattern thanks to the suggested associated transformations.

#### Measurable metrics

First of all, thanks to semantic web technologies, the annotations allow us to bind any kind of information for all elements of models. These annotations can be useful in defining usability indicators. For instance, each user task could be decorated with additional information in order to supply the end users with relevant indications to understand and complete the tasks (e.g. help, input mask, understandable error messages etc.). The coverage rate of these annotations within models can be measured by on-

tology-based tools and be used as a usability indicator.

The traceability information about the links between transformed elements can also be stored by using annotations. This adds semantic to the logs of user interactions, which means that rather than having a basic log of events, it is possible to analyse them with the initial associated elements of models, from concrete models (e.g. CUI, FUI) to abstract models (e.g. task model) and get the effectiveness and efficiency measures during user testing.

#### Matched patterns

As previously mentioned, the patterns are used to detect specific structured elements which have either a positive or negative influence on the usability criteria.

Here is an example of a pattern improving the system feedback of the designed UI, by suggesting the addition of a time indicator in AUI/CUI for specific computer tasks. In this case the computer tasks are loading and saving tasks. One of the usability criterion positively influenced is the “Feedback” criterion, whose description coming from QUIM is “Responsiveness of the software product to user inputs or events in a meaningful way.”

Thus, the suggestion can be expressed as: “Add visual indicator components in AUI and so CUI models, for each loading or saving computer tasks.” To refine this suggestion, the proposed transformations can depend on the waiting time, which can be estimated or come from logs: a) from 0 second to 0.3 seconds, no time indicator; b) from 0.3 seconds to 2 seconds, add a spinner loader; c) Above 2 seconds, add a progress bar. Here is an example of a priori defined influences for each variant.

Variants	Positive influence	Negative influence
a	Likeability	<i>In this case no feedback is needed</i>
b	Feedback	Readability
c	Feedback Accuracy	Attractiveness

**Table 4. Variants and usability influences.**

An example of an associated SPARQL query retrieving all loading and saving tasks (task\_x) with duration greater than a given duration (duration\_x) is:

```
PREFIX pi:<http://genius.tudor.lu/projectinstance.owl#>
PREFIX usi:<http://genius.tudor.lu/usixml.owl#>
SELECT
  ?task_x
FROM
  <http://genius.tudor.lu/projectinstance_task.owl>
WHERE {
  ?task_x rdf:type usi:SaveTask .
  ?task_x rdf:type usi:LoadingTask
  ?task_x usi:TaskDuration ?duration
  filter ( ?duration > duration_x )
}
```

Thanks to this list, the designer is able to choose among the retrieved tasks and chooses to accept or reject the suggestions for adding an abstract individual component dedicated to the time indicator. If the suggestion is accepted, here is an example of a request performed into the project instance to add corresponding statements:

```
PREFIX pi:<http://genius.tudor.lu/projectinstance.owl#>
PREFIX usi:<http://genius.tudor.lu/usixml.owl#>
INSERT IN GRAPH <http://genius.tudor.lu/projectinstance.owl#> {
  pi#AIC_x rdf:type usi:AbstractIndividualComponentOutput
  pi#AIC_x rdf:label "Time Indicator for task_x"
  pi#AIC_x rdf:Comment "Added from suggestion engine"
  pi#TR_x rdf:type usi:ReificationTaskToAUI
  pi#TR_x usi:TransformationSource pi:task_x
  pi#TR_x usi:TransformationTarget pi:AIC_x
}
```

The statements about the link between the computer task and the added abstract individual components are inserted into the graph to preserve the traceability and semantic logs.

Furthermore, other examples directly linked to the UIs coming from our case study are described in Table 2. The list of variants and the influences are established a priori and their relevance needs to be tested and validated by experts in usability.

	Variant 1	Variant 2
Context: "The user task is a validation task"		
<b>Description of transformation</b>	Create a "validation button"	Create a "validation button" and a "confirmation message box"
<b>Involved Models</b>	Task To AUI/CUI	Task To AUI/CUI
<b>Positive influence</b>	Minimal action	Fault tolerance
<b>Negative influence</b>	Operability	Minimal action
Context: "The user task is a selection of items"		
<b>Description of transformation</b>	Create a select box if the number of items is less than 15	Create a filterable list if the number of items is greater than 15
<b>Involved Models</b>	Domain/Task To AUI/CUI	Domain/Task To AUI/CUI
<b>Positive influence</b>	Minimal action Simplicity	Navigability
<b>Negative influence</b>	-	Minimal action

**Table 2. Other examples of transformations.**

## RELATED WORK

A recent proposition of a comprehensive and generic quality metamodel, named QUIMERA [6], has been done in the context of MBUI. This model aims to support the modelling of quality attributes in order to use it for a design rationale within an MDE approach. This metamodel

can be used for the implementation of usability models, such as our usability metamodel.

With regards to design rationale and the QOC approach, the TEAM notation, as presented in [28], provides an extension to the traditional QOC semi-formal notation by taking into account usability concerns and establishing links between a task tree and the options. Our usability metamodel presented here is rather similar but has different objectives. Where TEAM aims at justifying the design choices, we try to operationalise the decision-making process. Thus, we consider not only links between options and tasks but also between options and components pertaining to the AUI and CUI models.

In the MBUI and MDE context, the graph transformations have been explored in [12] to express the transformations between models. However, these transformations are not particularly devoted to usability.

The ontologies are often used as a way for qualifying UI items; for instance, they are used in [1] to describe the UI items (the semantic of containers, widgets, layout, etc.), and the associated annotations are exploited to improve the reorganisation of the UI items for context-aware interactive systems.

In the first and the second step of the GENIUS method, an approach based on ontologies and graph transformations, can take advantage from these works, to enrich the usability metamodel, integrate usability inputs within basic transformations and consider another approach based on ontologies to enhance MBUIs.

## CONCLUSION

In this paper, we have focused on the description of the first two steps of the GENIUS method, which defines a generic model-driven iterative process to design user interfaces. In this process, we propose a way, based on ontologies, to implement the transformations between models since we believe that usability should be specifically taken into account during this modelling step.

To achieve this purpose, we propose to express our models as ontologies, use inference capabilities and querying systems of related tools, and express transformations between models as graph transformations. The ergonomic heuristics and associated indicators are then implemented from the defined usability metamodel, which assists the designer during the modelling of the interactive system. The next step is to define a set of generic and relevant indicators in our scope, associate them positively or negatively to the impacted criteria and validate them with designers through our case study. This framework is implemented, but we still need to evaluate its relevance and, where appropriate, to open to other dimensions to support usability inputs.

The GENIUS method will be carried for one case study in the AEC industry. In this context, each construction project requires well-adapted software-based services and

user interfaces to improve the efficiency of business collaborations as well as the quality of end-user experience for the interfaces and devices that may be used. A comparative study of interface design will be conducted. It will aim to systematically compare the automatic design method described above to a more traditional and iterative method of guaranteeing “user-centred design” [9]. The specific experimental protocol that we will set up is based on a real interface dedicated to the construction domain. Thus, we propose to 1) analyse the HCI used, 2) analyse the generated HCI using the GENIUS methodology and 3) compare the results.

This comparison will be based on quantitative and qualitative approaches. With the existing HCI, the statistical feedback will be obtained through inserting some trace analysis components into the existing HCI. Then interviews with actual users will provide qualitative data on the ergonomic quality of the HCI and possible improvements. With the generated HCI, trace analysis components will automatically provide quantitative feedback on the user interaction, which will be directly compared to the results obtained with the existing HCI results. The use of the generated interfaces in an experimental context in the usability laboratory of the EMACS department (Université du Luxembourg) will then provide qualitative feedback on the generated HCI.

#### ACKNOWLEDGMENTS

The research in this paper was carried out within the project GENIUS, funded by the “Fonds National de la Recherche” in Luxembourg.

#### REFERENCES

1. Brel, C., Dery-Pinna, A.-M., Faron-Zucker, C., Renavier-Gonin, P., and Riveill, M. An Ontology-based Interactive System to Compose Applications. In *Proc. of WEBIST'2011*.
2. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J. A Unifying Reference Framework for Multi-Target User Interfaces. *Interacting with Computers* 15, 3 (2003), pp. 289–308.
3. Czarnecki, K. and Helsen, S. Classification of model transformation approaches. In *Proc. of the 2<sup>nd</sup> OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture* (2003), p. 15.
4. Favre, J.-M. and Calvary, G. Mapping Model: A First Step to Ensure Usability for sustaining User Interface Plasticity. In *Proceedings of the MoDELS, Workshop on Model Driven Development of Advanced User Interfaces* (2006).
5. Fernandez, A., Insfran, E., and Abrahão, S. Integrating a Usability Model into Model-Driven Web Development Processes. In *Proc. of WISE'2009*.
6. Garcia Frey, A., Céret, E., Dupuy-Chessa, S., and Calvary, G. QUIMERA: A Quality Metamodel to Improve Design Rationale. In *Proc. of 3<sup>rd</sup> ACM Symposium on Engineering Interactive Systems EICS 2011* (Pisa, June 2011). ACM Press, New York (2011).
7. Gronier, G., Kubicki, S., Vagner, A., Schwartz, L., Montecalvo, E., Altenburger, T., and Halin G. Model-driven generation of ergonomic user interfaces in visualization services. In *Proc. of 1<sup>st</sup> International Workshop on User Interface eXtensible Markup Language UsiXML'2010* (Berlin, June 20, 2010). Thalès, Paris (2010).
8. Gruber, Thomas R. A translation approach to portable ontology specifications. *Knowledge Acquisition* 5, 2 (1993), pp. 199–220.
9. ISO 13407. Human-centred design processes for interactive systems. International Standards for Business, Government and Society.
10. Jameson, A. User Modeling Meets Usability Goals. In *Proc. of UM'2005*, L. Ardissono, P. Brna, and A. Mitrovic (Eds.). LNAI, vol. 3538. Springer-Verlag, Berlin (2005), pp. 1–3.
11. Jameson, A. Adaptive Interfaces and Agents. In *Human-Computer Interface Handbook*. J.A. Jacko and A. Sears (Eds.). (2003), pp. 305–330.
12. Limbourg, Q. and Vanderdonckt, J. UsiXML: A User Interface Description Language Supporting Multiple Levels of Independence. In *Engineering Advanced Web Applications*, M. Matera, S. Comai, S. (Eds.). Rinton Press, Paramus (2004), pp. 325–338.
13. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., and Víctor López Jaquero. UsiXML: a Language Supporting Multi-Path Development of User Interfaces. In *Proc. of 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems EHCI-DSVIS'2004* (Hamburg, July 11–13, 2004). Lecture Notes in Computer Science, vol. 3425. Springer, Berlin (2004), pp. 200–220.
14. Oliveira, K. Une étude initiale sur les métriques d'utilisabilité pour les modèles de tâches exprimés avec CTT. In *Proc. of INFORSID'2011*.
15. Ontology Definition Metamodel (ODM) Version 1.0, Object Management Group, 2009.
16. Schaefer, R. A Survey on Transformation Tools for Model Based User Interface Development. *Transformation Journal* (2007), pp. 1178–11.
17. Seffah, A., Donyae, M., Kline, R. B., and Padda, H. K. Usability measurement and metrics: A consolidated model. *Software Quality Journal* 14, 2 (2006), pp. 159–178.

18. Sottet, J-S., Calvary, G., Coutaz, J., and Favre, J-N. A Model-Driven Engineering Approach for the Usability of Plastic User Interfaces. In *Proc. Of Engineering Interactive Systems EIS'2007*, pp. 140-157.
19. Vanderdonckt, J. A MDA-Compliant Environment for Developing User Interfaces of Information Systems. In *Proc. of 17th Conf. on Advanced Information Systems Engineering CAiSE'05* (Porto, 13-17 June, 2005). O. Pastor & J. Falcão e Cunha (Eds.). Lecture Notes in Computer Science, vol. 3520. Springer-Verlag, Berlin (2005), pp. 16-31.
20. Nielsen, J. Heuristic evaluation. In *Usability inspection methods*, (1994) pp. 25-62.
21. Bastien, J.M.C. and Scapin, D.L. *Critères ergonomiques pour l'évaluation des interfaces utilisateurs*, Rapport technique INRIA, vol. 156, 1993.
22. Protégé, <http://protege.stanford.edu/>
23. Virtuoso, <http://virtuoso.openlinksw.com>
24. Hermit, <http://hermit-reasoner.com/>
25. Sirin, E., Parsia, B., Sparql-dl: Sparql query for owl-dl.
26. 3rd OWL Experiences and Directions Workshop (OWLED-2007).
27. TheJIT, Javascript Infovis Toolkit, <http://thejit.org>
28. MacLean, A., Young, R.M., Bellotti, V.M.E., and Moran, T.P. Questions, options, and criteria: elements of design space analysis. 1996.
29. Lacaze, X and Palanque, Ph. DREAM & TEAM: A Tool and a Notation Supporting Exploration of Options and Traceability of Choices for Safety Critical Interactive Systems. In *Proc. of INTERACT 2007*.

# Towards a new Generation of MBUI Engineering Methods: Supporting Polymorphic Instantiation in Synchronous Collaborative and Ubiquitous Environments

George Vellis<sup>1</sup>, Dimitrios Kotsalis<sup>1</sup>, Demosthenes Akoumianakis<sup>1</sup>, Jean Vanderdonckt<sup>2</sup>

<sup>1</sup>Department of Applied Information Technology & Multimedia,  
Technological Education Institution of Crete  
Stavromenos 71004, Heraklion, Crete, Hellas  
{g.vellis, kotsalis, da}@epp.teicrete.gr

<sup>2</sup>Louvain School of Management, Université catholique de Louvain  
Place des Doyens, 1 – B-1348, Louvain-la-Neuve, Belgium  
jean.vanderdonckt@uclouvain.be

## ABSTRACT

This paper extends model based UI engineering so as to address polymorphic UI development in the light of distributed synchronous collaborative ubiquitous environments. Our current effort concentrates on extensions to a popular model-based user interface description language, namely: “UsiXML” and proposes a suitable development methodology and a dedicated runtime infrastructure.

## Author Keywords

Polymorphic instantiation, Multi-user interfaces, Social awareness, UIDL, UsiXML, Model-based UI Engineering.

## General Terms

Design, Experimentation, Human Factors, Verification.

## ACM Classification Keywords

D2.2 [Software Engineering]: Design Tools and Techniques – *Modules and interfaces; user interfaces*. D2.m [Software Engineering]: Miscellaneous – Rapid Prototyping; reusable software. H5.2 [Information interfaces and presentation]: User Interfaces – *Prototyping*.

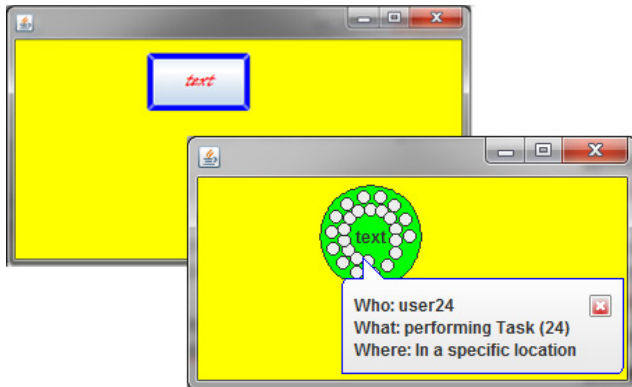
## THE PROBLEM

The remarkable progress which we have witnessed over the last years, in fields relevant to information technology, has resulted in a wide range of novel computational appliances, having deeply penetrated a broad range of our daily practices. Increasingly, users come across of a sharply growing number of interactive applications enabling them to engage in various sorts of collaborative and social endeavors in a wide variety of contexts. In such cases one important aspect to consider relates to the UI design to support group work appropriately. It turns out that such development is quite challenging since prominent engineering methods and supporting tools fail to account for the requirements imposed by diverging user- and usage-context parameters (i.e.: platform, environment, user-stereotype). ‘Polymorphic’ instantiation, conceived of as a capability to manage concurrently alternative instances of the same interaction element at runtime, constitutes key provision in the direction of coping

with this diversity (through extensive and intuitive UI adaptations). Specifically, it separates the role of the object (i.e., why an object is used) from its physical / presentation details in a designated context. It also hides technical features of an object’s implementation from its core pattern that is independent of physical realization, thus common to potentially more than one implementation. In this account, polymorphic instantiation functions as ‘idiom’ in user interface engineering. Nevertheless, prevalent UI toolkits (java/swing, MAUI, QT, etc.) and languages do not make sufficient inroads towards supporting such polymorphic UI arrangements. Instead, they remain committed to supporting provisions for cross-platform generalizations (i.e.: addressing portability) rather than ‘true’ abstractions. Supporting a stronger notion of abstraction (with corresponding toolkit-level provisions) would enable, among other things, richer specifications of interactive behaviors, enumeration of alternative widget incarnation (i.e.: via encapsulation) and finer-grained and dynamic assembly of polymorphic UIs.

A particular scenario where polymorphic instantiation reveals its power as an engineering pattern is in the case of synchronous collaborative sessions supporting real-time distributed team work. Arguably, the state of the art offers a rich insight into the desired groupware functionality and the features to facilitate such functionality (i.e., session management, object sharing, floor control, etc.). However, maintaining synchronized versions of potentially inconsistent multi-user UI instances, due to context-related parameters i.e., platforms or roles, continuous to pose substantial challenges. Toolkit-level sharing as exploited by the state of the art in groupware toolkits (i.e.: MAUI [1]) fails to account for the diversity of target devices involved in emerging ubiquitous environments, since it requires that the same groupware toolkit is available in every target-device. Nevertheless, this turns out to be an impractical assumption since dedicated runtime environments required for operating such toolkits might even not be supported (i.e.: JRE). Furthermore, even if alternative groupware

toolkits, accommodating constraints imposed by target-platforms, were used for assembling each target multiuser interface and that interoperability could be granted in some way, another major obstacle to overcome would be that of supporting awareness across engaged participants. Since different groupware toolkits make use of alternative mechanisms to facilitate awareness (i.e.: telepointers, radar-views, etc.), it would be rather impossible to achieve successful collaboration among users since awareness support would be foredoomed to failure. Finally, it is worth pointing out that groupware toolkits inherit the unimorphic style of programming imposed by their single-user counterparts; therefore they dismiss support for advanced and intuitive adaptation (i.e.: polymorphic instantiation in synchronous sessions). In this paper we focus on addressing these challenges to support complex interaction in synchronous collaborative and ubiquitous scenarios. By this account, our aim is to layout the foundations of a UI engineering methods that combine MBUI methods and toolkit programming to support polymorphic UI instantiation patterns.



**Figure 1. Alternative instantiations of ‘abstractButton’.**

To this effect, we propose to briefly elaborate on a very simplistic but demanding scenario. Figure 1 depicts two versions of a simple UI comprising one container object and one button. Each version of the UI is intended to serve a different user role. The left-hand side depicts the instance for participants (role A) where the button is manifested using a conventional rectangular pattern with two states (as commonly encountered in various toolkits). The right-hand side represents the moderator’s (role B) view of exactly the same button rendered in a synchronous collaborative and distributed setting (i.e.: multiuser). This time the instance of button is augmented (at both physical and syntactic levels) so as to possess visual affordances that convey awareness. In the specific example this is supported by using a dedicated nested radial layout indicating how many participants (i.e., users with role A) are currently engaged in the session or have access to this replica. The button’s label remains the same in both cases just to indicate that we are working with the same abstraction that is capable of polymorphic instantiation.

In light of the above, the present research is motivated by some key questions;

- Can the above UIs be designed so as to be generated from the same specification?
- Can this be supported in a manner that allows designers to designate complex interactive behaviors as means to facilitate affordances such as awareness?
- What kind of widget attributes need to be manipulated (i.e., layout, topology, access policies) at client and server sides?

Responding to these questions is expected to enlighten our views on prominent architectural limitations of current UI engineering methods. Most importantly, however, it is likely to lead to new insights into affordance-based design of UIs and how they may be associated with MBUI engineering.

## RELATED WORK

One approach to address the problem is to craft UIs by combining popular UI toolkit-based programming and model-based UI engineering techniques. Toolkit programming is grounded on the philosophy of building user interfaces as hierarchies of reusable widgets by registering event event handlers. This allows for complex interactive behaviors and customized dialogues [1].

On the other hand, MBUI engineering makes use of abstract notations and mark-up languages to facilitate mapping of abstract components to platform-specific toolkit libraries by delegating the display to a platform-specific renderer [2]. At first site, this seems to provide a better frame of reference to addressing the problem as it offers greater flexibility and provisions with regards to potential heterogeneities in contexts of use (i.e.: cell phones, web, etc.). Nevertheless, the key issue in the problem presented earlier is not portability and the degree to which it is supported for a set of pre-determined widgets.

Rather, it is a matter of utilizing widgets (either native or custom) so as to facilitate polymorphic instantiation with variations at all levels including physical, syntactic and semantics. This is an issue which is only partly and loosely addressed by model-based approaches which are still limited to crafting rather simplistic form-based UIs for conventional presentation vocabularies and contexts of use. This limitation is attributed to lack of provisions a) for enhancing expressive capacity of interactions by allowing augmentation and/or expansion of UI vocabularies and b) to allow alternative instantiations to be encapsulated in the context of supported unimorphic widgets.

## Polymorphic instantiation

Polymorphic instantiation is a demanding notion for UIs which has not been adequately addressed in the recent literature. It was initially introduced in toolkit based systems such as the HOMER user interface management suite [3], and the Platform Integration Module [4]) in unified user in-



terface development. These studies explored the challenge for 4GLs to realize polymorphic UI objects. At the time, the concept did not implicate any kind of design considerations regarding the specification of polymorphic dialogues. Rather, it was conceived as a property of the language, not a desirable affordance to be designed.

Model-based UI approaches promised to alleviate this shortcoming by offering new models and abstractions to reconsider affordance-based UI design. In practice, this never turned out to be the case, as most of the available scholarship concentrates on methodologies, engineering techniques and tools to describe native interaction components and their transformation and mapping from one dialect to another. For instance, COMETS [5] provide support for multi-level UI adaptations at runtime driven by support for semantic networks (no WSL, no advanced widgets). It is worth noticing that none of these systems integrates novel widget specification languages (WSL) or advanced widgets.

### **Collaborative aspects**

Another key issue in reconsidering affordance-based design of UIs is collaboration and in particular synchronous collaboration. Again, here progress has been slow in both toolkit-based systems and model-based UI engineering. Groupware toolkits are around for more than two decades now. Their focus has been on managing technical properties of collaboration such as session management, floor control, object sharing and replication. Awareness was conceived of at multiple levels (task, activity, social awareness) but it was never fully supported.

Notable exceptions include research prototypes such as the MAUI toolkit which supports group awareness, is java-based and facilitates multi-user UI design via a dedicated IDE provided (JBuilder). Other systems such as BEACH [6] concentrate on aspects of colocation in ubiquitous collaborative systems but offer no support for awareness,

Model-based UI engineering, once again, has brought about modest but notable improvements. TOUCHE [7] provides multiuser functionality using adhoc mappings to a custom underlying groupware toolkit. Support for awareness is fixed during design phase. In the same vein, indicative examples include approaches such as CIAM (Collaborative Interactive Applications Methodology) [8] or AMENITIES (A Methodology for aNalysis and desIgn of cooperative systEmS) [9].

These efforts primarily concentrate on devising notations and tools to model cooperative behavior and workflows. In effect, their primary contribution is that they make explicit different elements of collaboration (i.e., roles, responsibilities and tasks) using dedicated notations (i.e.: CIAN [10]). As a result they provide no support for designing the UI. No support for session modeling.

## **APPROACH**

### **First principles**

The present work seeks to improve on the state of the art by addressing several of the challenges and the limitations introduced earlier. Specifically, it focuses on issues related to polymorphic instantiation and collaborative management of UIs in ubiquitous distributed contexts. The approach builds on the Model-based UI engineering paradigm aiming to inject new elements so as to facilitate a more accountable affordance-based UI perspective. To this end, our current effort is grounded on a very popular UIDL, namely UsiXML [11] and inherits a single design process for all supported contexts of use. In turn, this leads to significant reductions in (re-) engineering costs and programming complexity while resulting to more reliable and coherent UIs. UsiXML makes use of three distinct levels of abstraction (i.e.: CTT, AUI, CUI) for incrementally specifying a UI in order to promoting separation of concerns. Specifically, the CTT model captures a UI specification in a computation independent manner while adopting a user centered perspective. Additionally, the Abstract User Interface (AUI) enables the definition and derivation of both modality-independent and multimodal interactive object hierarchies by providing support for the CARE properties. Finally, the CUI-model focuses on a platform-independent (or better toolkit-independent) UI specification. At all times, transition between supported models is enabled via dedicated transformations. A salient feature of UsiXML is support for plasticity [12], which is addressed via the ‘context-model’ enabling context - sensitive transformations to be performed so as to accommodate diverse requirements posed by different contexts of use.

### **Extensions**

Despite the relative ease in designing for multiple environments, UsiXML lacks support for taking advantage of advanced interactive capabilities offered by target platforms since its support is limited only to a reduced set of rather simplistic natively supported form-based elements (i.e.: buttons, labels, etc.). Moreover, as expected, no widget specification language is provided since the widget range is a priori known and hardcoded within the transformation logic via direct calls to platform-specific presentation vocabularies. Besides, support for polymorphic assembly of UIs is completely dismissed, since no dedicated mechanisms or widget specification language exists so as to allow widgets to encapsulate alternative instantiations. In addition, no provisions are made either for modeling or runtime support of distributed multi-user interactions (i.e.: session modeling, multi-user artifacts, awareness support, etc.).

In order to address these shortcomings, our efforts concentrated on a series of modifications in language-level which resulted in either new models (i.e.: language expansions), or enhancements of already existing (i.e.: language augmentations).

	WSL	UsiXML: UI Model											Architecture – Runtime Environment		
		Existing Models						New Models					Collaboration Plugin	Platform Server	Web Services
		CTT	AUI	CUI++	Context	Mapping++	Squad	Widget Resource	Abstraction	Consistency	Behavior	Session			
Non-native Widgets Support	✓			✓				✓			✓			✓	
Polymorphic Instantiation	✓			✓	✓	✓	✓	✓	✓	✓	✓			✓	
Abstraction	✓			✓		✓		✓	✓	✓	✓			✓	
Replication									✓			✓	✓	✓	✓
Collaboration				✓				✓	✓	✓	✓	✓	✓	✓	
Social Awareness	✓			✓		✓		✓				✓	✓	✓	✓

Table 1. Enhancements in UsiXML.

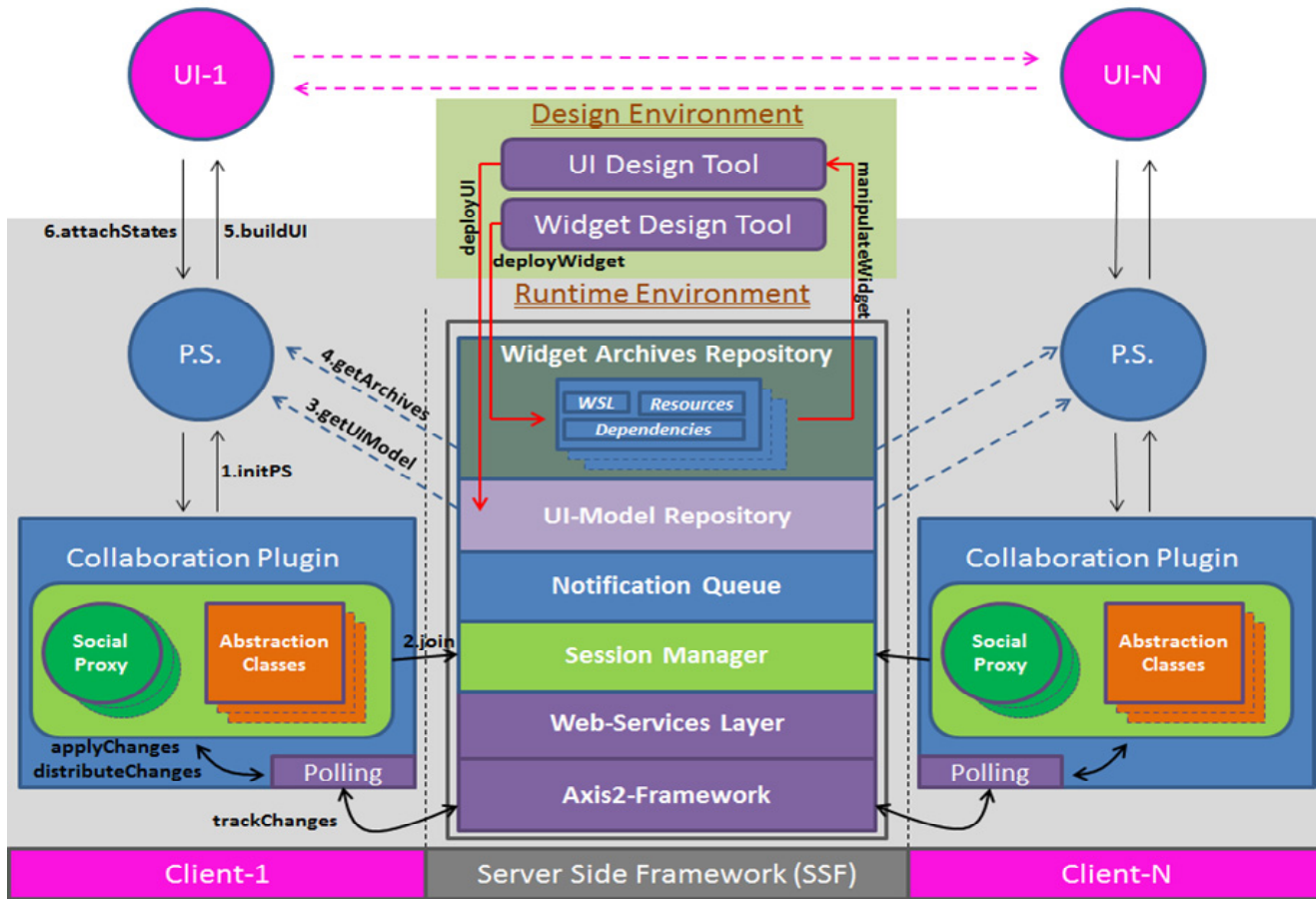


Figure 2. Architectural Abstraction.

Table 1 summarizes the injections introduced into the latest version of UsiXML models reflecting the state of our current research. As shown the focus is on key affordances (left column) and how they are implicated through revisions in or extensions of UsiXML models. In turn, this gives rise to a general purpose architectural abstraction which is depicted in Figure 2.

#### ARCHITECTURAL MODEL AND COMPONENTS

Our architecture makes use of several novel components that extend model-based UI engineering and make new ground in specification-based engineering

#### Widget specifications (for native and custom widgets)

In order to fully exploit the management of truly diverse interaction elements and vocabularies in target platforms (i.e.: natively and/or non-natively supported widgets) we crafted a formal (i.e.: compliant with a dedicated xml schema) specification mechanism referred to as Widget Specification Language (WSL). In order for a widget to be properly integrated and fully deployed by language's constructs, each must be separately introduced in a compliant to the provided WSL schema manner.

A WSL is responsible for capturing all required details for widget's proper utilization, parameterization and smooth

operation. Such details include widget's name, unique id, description, etc. Moreover, specific constraints can be defined, such as: platform availability, runtime environments required (i.e.: JRE), or even constraints on specific versions of external libraries required for widget's proper instantiation. In addition, widget's api (i.e.: accessor and mutator methods, constructors, etc.) is properly codified along with proper mappings to a widget resource model instance that it incrementally manifested exclusively at design time for capturing widget's detailed configuration (i.e., pre-instantiation state) in terms of simple or compound property-value pairs.

### **Polymorphism, abstraction and collaboration**

Support for polymorphic instantiation is primarily granted via provisions in the WSL permitting multiple alternative instantiations to be defined (i.e.: encapsulation) for a specific widget type (i.e.: 'abstract widget'). To this end, also support for abstraction is granted since beyond polymorphic properties enumeration, also abstract properties must be defined, which constitute the common bond across all alternative instantiations. Moreover, it worth noting that widget resource model is also responsible (at design time) for capturing the range of adaptations each polymorphic widget may be permitted to run. Decision logic upon which alternative incarnations are utilized is specified in the light of the context and/or squad models. Each polymorphic instance is expected to define all supported behaviors in terms of properly codified finite state machines. At this level only supported states and permitted transitions among these can be defined and not at any case application-specific logic which is captured at design time by the 'Behavior' model. Abstraction (and the abstraction model) is not only used to facilitate polymorphic instantiation in collocated settings, but also to provide the mechanism to support distributed synchronous collaborative interactions in the context of the 'abstraction' model. Specifically, Abstraction model's prime aim is to establish an abstraction layer between multiuser widgets in the light of thorough or partial commonalities in regards to their models (in terms of Model View Controller architecture) while keeping their corresponding views synchronized.

Abstraction model comprises classes defined using class diagrams (at design time), in an attempt to facilitate model-level sharing which is completely relieved (in contrast to toolkit-level sharing) from physical-level properties, thus providing higher flexibility with regards to potential variations between alternative to be synchronized views. The reason for not selecting to synchronize directly widget corresponding models is mainly due to the need for design and implementation simplicity which is best served by centralizing concerns (i.e.: shared model).

Specifically, it would be much more complex to implement in a peer to peer manner intertwined relations between models each adhering to different widget than defining a single external model and providing appropriate hooks to

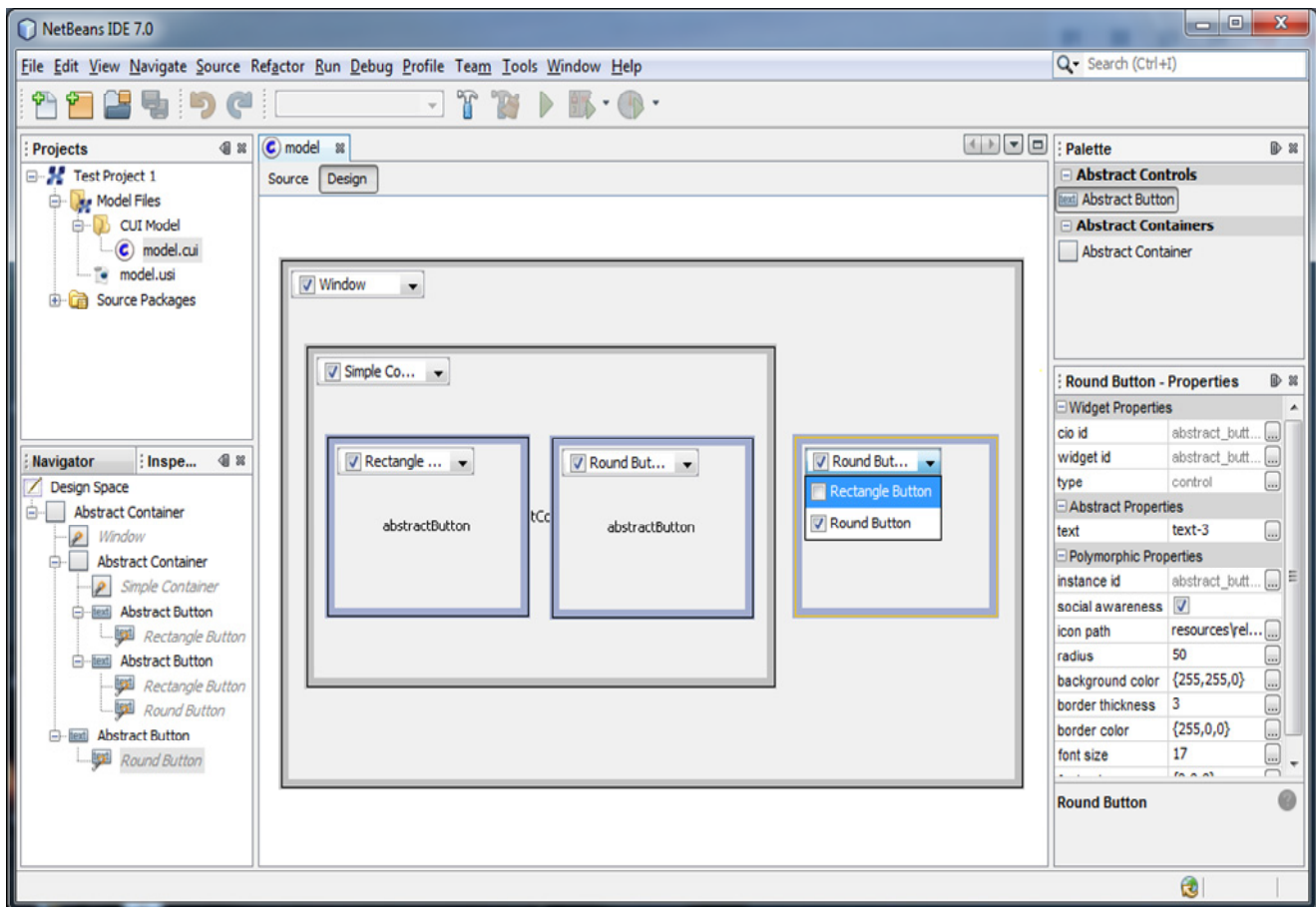
synchronize every widget with that single model. Moreover it would be rather infeasible to inject collaboration-aware code and logic (i.e.: consistency and/or concurrency control) to collaboration unaware widget-models. Abstract classes alleviate this issue since collaboration-aware code is properly injected during their translation to platform specific code so as to automatically broadcast, receive and apply changes made by remote users. There are also dedicated provisions for properly managing consistency or concurrency related issues. Support for feed through, i.e.: apply changes received as an input of another distributed user, is implemented via dedicated mappings between shared properties of classes (i.e.: 'abstraction classes') defined in the context of abstraction model and direct calls to instance-specific API.

### **Replication and awareness**

Moreover, it is worth noting that abstraction classes are distributed by replication in the context of particular synchronous collaborative sessions modeled via the 'session model'. In this context, support for social awareness is natively provided for widgets capable to visualize social scent using mixed dialogues. Specifically, social awareness is enabled inside widgets' specifications at polymorphic-instance level in order to indicate capacity for that specific instance to properly visualize social awareness. Nevertheless, the final decision for engaging social awareness for 'socially-aware' polymorphic instance is determined via a corresponding to that instance entry inside the WidgetResource model instance.

### **SUPPORTING THE DESIGN PHASE**

Having described briefly architectural elements of our approach, an attempt is made to explain how these concepts are implicated during the design phase. We have developed a prototype system that takes advantage of the NetBeans platform, introducing on top of it a number of custom modules to support the development of either single user or distributed collaborative projects. The main differences between these two project types are in the way they employee to 'compile', distribute and execute the produced UI specifications, as well as in the number of available plugins engaged by default. For instance in case of collaborative application, a pre-requisite is the registration of a compatible server side environment dedicated to managing special purpose collaborative aspects (i.e.: synchronization, session management, etc.). Moreover, in distributed collaborative applications where UI models need to be accessible by several users over the network the pre-requisite is a centralized repository for depositing shared resources (i.e.: common models, widget archives, etc.). Furthermore, additional provisions are required for distributing a reference to all users may engage in a particular session (i.e.: 'distributed shortcuts'). Nevertheless, in all cases the development process remains quite common in terms of tool support, since most of custom plugins devised, remain the same (i.e.: editors for manipulating: CTT, CUI, Squad, etc.).



**Figure 3. Design Environment for Polymorphic UIs.**

A representative instance of our system depicting the design of the UI in Figure 1 is presented in Figure 2. At any time, a design project has a dual view – the graphical editor’s visual depiction of abstract interaction components and the source (XML) view of the respective model.

Typically, designers utilize the palette to introduce components in a direct manipulation fashion and specify their properties. Design updates are immediately manifested as XML model changes. The properties of each widget fall in three categories, namely widget-specific properties, abstract properties and polymorphic properties. The way in which these properties are manipulated is dependent on specifications in the widget archive.

#### Widget archives

Widget archives provide the means for introducing advanced customized widgets. They may be non-native interaction components, developed by third-parties and shared to the design community. We have designed several custom widgets, some of them quite complex, to test the concept of a widget archive and the way in which it is articulated using our system. For purposes of illustration, in our example we discuss one such component which is referred to as ‘Round button’. In order for a widget archive such as

that encapsulating the ‘Round button’ incarnation to be deployed into our tool, it must be firstly installed by a series of steps depicted in Figure . Due to space limitations and the focus of the present work, we will not provide further details on this process.

#### Visual manipulation of abstract interaction elements

Each uploaded widget appears in the ‘Abstract Control’ section of the palette and can be utilized in a direct manipulation fashion. Thus, it can be attached to a container object, resized, relocated and specified. For widget archives with polymorphic instantiation, the visual depiction of the abstract component indicates the options available (see right-hand side round button in 3). Further specification of the widget can be attained by manipulating properties in the lower part of the palette leading to a concrete instance. Figure summarizes the mapping for the right-hand side button which exploits polymorphic instantiation capability. We refer to this model as the ‘WidgetResource’ model which conveys the range of possible polymorphic adaptations. Thus, Figure depicts how such customizations are defined (i.e.: triplet comprised of WSL instance id and instances of the WR & CUI model) for the specific widget in Figure 3.

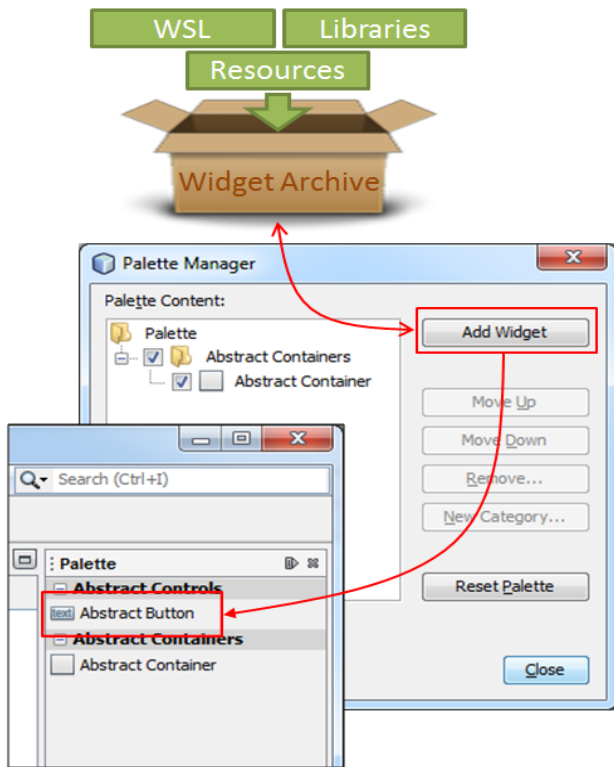


Figure 4. Widget Deployment Workflow.

```

</abstractContainer>
<abstractButton id="abstract_button_3" widgetId="abstract_button" ... />
</abstractContainer>
</cuiModel>
<widgetResource id="wdg_res_6" cioId="abstract_button_3" widgetId="abstract_button">
  <property name="text" value="text-3" ... />
  ...
  <instance id="inst_2" socialAwarenessEnabled="true" ... >
    <property name="iconPath" value="resources\relax.png" ... />
    <property name="radius" value="50" ... />
    ...
  </instance>
</widgetResource>
<widget id="abstract_button" name="abstractButton" type="control">
  ...
  <abstract_properties>
    <property id="abs_prop_1" name="text" ... />
  </abstract_properties>
  ...
  <instances>
    <instance id="inst_2" name="roundButton" isSocialAware="true" ... >
      <polymorphic_properties>
        <property id="inst_2_prop_1" name="icon" ... >
          <property id="inst_2_prop_2" name="iconPath" ... />
        </property>
        <property id="inst_2_prop_3" name="radius" ... />
      </polymorphic_properties>
    </instance>
  </instances>
</widget>

```

Figure 5. Widget's Pre-instantiation Configuration Mapping.

## RUNTIME ENVIRONMENT

In order to support the novel features introduced in the previous sections, advanced software components have been crafted both at the client as well as at the server side (see Figure 2).

### Client-side components

At the client side of particular interest is a runtime infra-

structure developed namely: 'Platform Server' (P.S.), denoting a multifunctional software component which guarantees language's smooth and consistent operation. Dedicated to a particular platform, the P.S. constitutes a virtual software layer between UsiXML models and the underlying system with its role being limited to: distributed class loading (in case of managing non-native interactive elements), event management (as part of facilitating collocated and/or distributed synchronization), as well as compilation (see: Domain model, Abstraction Model) and interpretation of UsiXML models at runtime. In addition, another very important function in the context of collaborative sessions, either synchronous or asynchronous, relates to client-side support for session management. To this end, P.S. handles both grabbing and distribution of shared actions via triggering and managing inter-client (i.e.: inter-P.S.) message exchanges in the context of a particular session. To support this functionality P.S. interoperates with a custom dedicated server-side general purpose framework, built on top of the apache axis2 framework, to support session management. Moreover P.S. is responsible for handling replication process via managing (i.e.: generation of replicaIds, distributed registration, etc.) and maintaining a dedicated client-side replication list with replicaIds associated to corresponding object-references. In case of detected variations (via 'context-sniffer' daemon thread) regarding the context of use, P.S. is responsible for engaging a re-adaptation process driven-instructed from the server-side developed framework. Furthermore, another important function assigned to the P.S. relates to the process of overall handling non-native widgets (based on a Widget Specification Language devised on our side) part of which relates to 'custom events management' and 'widget data model' handling briefly discussed in previous section.

### Server-side components

On the other hand at the server-side there is a generic-purpose server-side framework (SSF). The role of the SSF in the context of distributed settings focus on maintaining a repository of accessible at runtime UsiXML models, for initial or re-adaptation process, correlated to a particular session (either synchronous or asynchronous). The SSF also handles low-level session management (in both modes, i.e.: synchronous and asynchronous), build on top of apache axis2 framework, by performing several functions such as creation, registration, etc., while also maintains a list with all running sessions. Regarding non-natively provided widgets, in the context of a particular UI description specification, the SSF contributes by maintaining a shared repository with platform-specific widget libraries, in respect to widget's platform availability, facilitating P.S. via distributed class loading to handling non-natively supported interactive hierarchies. Finally, it keeps a per-session notification Queue, accessed by performing polling on the client side based on dynamically determined intervals, facilitating synchronization of distributed multi-user components.



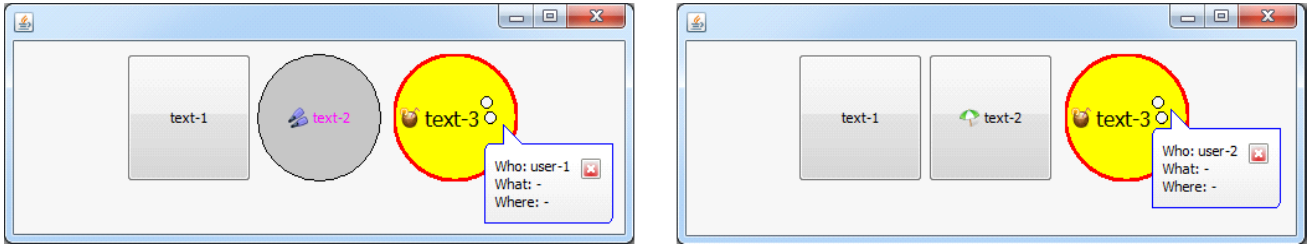


Figure 6. Runtime Instance of Final UIs.

#### AN EXAMPLE DESIGN CASE - UI GENERATION

In the present section we will elaborate on an indicative scenario in an attempt to provide a more detailed insight into the role of the supported models, the tools comprising the overall environment and the development practices. More specifically, let's assume that support was required so as to facilitate synchronous collaborative interactions in the light of heterogeneous contexts of use. In such cases a convenient point for engaging the design process would be that of CUI. Supposing that the UI design properly addressing our scenario is that displayed in Figure 3 then, in the course of alternative UI - execution flows, two potential instantiations (i.e.: assemblies) for two random users could be these depicted in the Figure 6. Henceforth the view sided on the left in Figure 6 would be referred as "user's 1" while the other as "user's 2". Access to these UIs is granted after a user-request is made, triggered by clicking on a dedicated 'shortcut' available in both target platforms (i.e.: user1 and user2), for engaging to the appropriate collaborative session with which the corresponding UI specification has been associated with (at design time). Upon successful engagement to a collaborative session, P.S. automatically gains access and downloads all data (i.e.: UI models, dependencies, etc.) required in order to proceed to assembling the UI. Following this, UI generation begins by interpreting the retrieved CUI specification. In order for the P.S. to facilitate polymorphic instantiation, it requires decision logic (D.L.) which will allow it to decide which instance is to be delivered. For the purposes of our example, this decision is to be made on the grounds of socially-aware criteria (i.e.: community membership) which are properly codified inside the three supported models, i.e.: the squad model, the WRM and the CUI model. Specifically, we defined the middle 'abstractButton' to incarnate as a non-native circular button in case the user belongs to 'community-1' while on the other hand, it should be instantiated as a two-state rectangular button in case the user is a member of 'community-2'. Figure 7 depicts how these relations are codified in the supporting models so as to deliver the desired effect.

Runtime instantiation of RoundButtons (i.e.: non-native interactive element), is instructed by widget's specification language which facilitates P.S. to fully exploit instance-specific apis properly codified so as to standardize among other dynamic linking to external libraries required for its instantiation (i.e.: dependencies) as well as allow direct calls to be made so as to alter its state which could be use-

ful in order to applying instance-specific configurations available from design phase (i.e.: icons, border color, etc., see: bottom right hand side in Figure 3). Each time a polymorphic interactive incarnation is instantiated in collaborative settings, P.S. seeks to determine (i.e.: WSL) whether or not that instance type provides native support for social awareness (SA, i.e.: social awareness enabled).

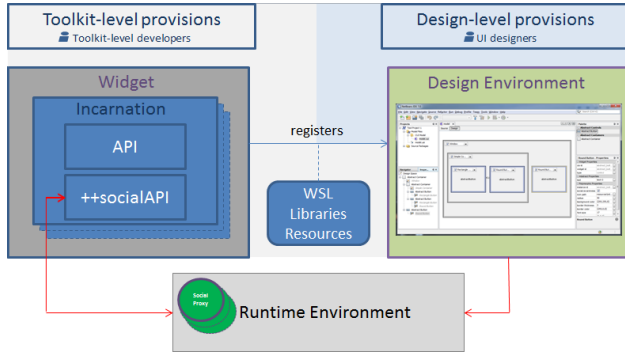
```
<squadModel>
  <squad id="squad 1" ... >
    <userProfile id="usr_1" ... >...</userProfile>
    <userProfile id="usr_2" ... >...</userProfile>
  </squad>
  ...
  <communities>
    <community id="community_1" name="community_1" ... >...</community>
    <community id="community_2" name="community_2" ... >...</community>
  </communities>
  <belongsTo id="bt_1">
    <source sourceId="usr_1"/>
    <target targetId="community_1"/>
  </belongsTo>
  <belongsTo id="bt_2">
    <source sourceId="usr_2"/>
    <target targetId="community_2"/>
  </belongsTo>
</squadModel>
<widgetModel>
  <widgetInstanceSensitiveTo id="icst_1" cioId="abstract_button_2">
    <source sourceId="inst_1"/>
    <target targetId="community_1"/>
  </widgetInstanceSensitiveTo>
  <widgetInstanceSensitiveTo id="icst_2" cioId="abstract_button_2">
    <source sourceId="inst_1"/>
    <target targetId="community_2"/>
  </widgetInstanceSensitiveTo>
  ...
  <widgetResource id="wdg_res_5" cioId="abstract_button_2" ... >
    <property name="text" value="text-2" />
    <instance id="inst_1" ... >
      ...
    </instance>
    <instance id="inst_2" socialAwarenessEnabled="false" ... >
      ...
    </instance>
  </widgetResource>
```

Figure 7. Rule for Polymorphic Decision Logic.

Native support implies that toolkit-level provisions have been made in the course of widgets' development so as to: a) enable social scent be properly visualized as well as b) give access for its manipulation by appropriating a dedicated standardized API (i.e.: 'addIndicator()', 'removeIndicator()', etc.) ready to be used by the runtime framework. In the present case, Round Button constitutes the only incarnation that has been properly manifested to provide support for social awareness. To this end, we had to apply toolkit-augmentation in many ways so as to create a circular bounded widget (note that non-rectangular widgets are not directly supported in java/swing) as well as a special purpose topology-policy to radially layout social indicators. In addition to physical-level enhancements, for SA to be properly managed, we had also to semantically augment



Round Button so that it can be appropriated by the runtime framework. Notably, such technical details are transparent to designers (see: Figure 3) whose role is limited to enabling or disabling this feature in case it is supported. Figure 8 attempts to clarify the allocation of provisions to handle SA in the current implementation.



**Figure 8. Implications in supporting Social Awareness in the course of the Design & Implementation process.**

Moreover, in case of native support, a final check is also performed in order to determine whether or not that affordance has been enabled (WRM) for that polymorphic instance. In case both checks are successful, P.S. engages SA (via polymorphic method binding to that widget) and performs a direct call to the Collaboration plugin which then assigns a social proxy keeping social scents up to date.

```
<widget id="abstract_button" name="abstractButton" type="control">
...
<instances>
  <instance id="inst_1" name="rectangleButton" isSocialAware="false" ... >
    ...
  </instance>
  <instance id="inst_2" name="roundButton" isSocialAware="true" ... >
    ...
  </instance>
</instances>
</widget>
<widgetResourceModel>
  <widgetResource id="wdg_rsc_4" cioId="abstract_button_1" widgetId="abstract_button">
    ...
    <instance id="inst_1" ... >...</instance>
  </widgetResource>
  <widgetResource id="wdg_rsc_5" cioId="abstract_button_2" widgetId="abstract_button">
    ...
    <instance id="inst_1" ... >...</instance>
    <instance id="inst_2" SocialAwarenessEnabled="false" ... >...</instance>
  </widgetResource>
  <widgetResource id="wdg_rsc_6" cioId="abstract_button_3" widgetId="abstract_button">
    ...
    <instance id="inst_2" SocialAwarenessEnabled="true" ... >...</instance>
  </widgetResource>
</widgetResourceModel>
```

**Figure 9. Enabling/Disabling Social Awareness Support.**

Figure 9 shows how SA is disabled for the round button in the middle of the UI, and respectively enabled for the round round button, in the right hand side. Finally, widgets' behaviors (i.e.: Finite State Machines, Behavior model) and abstraction classes are compiled on the fly to machine code. Once behaviors get compiled they are automatically attached, by a dedicated P.S. module instructed by widget's specification language, to the proper instances of the UI. In

the case of our example press state was synchronized (i.e.: shared property of an Abstraction Class).

## CONCLUSIONS AND FUTURE WORK

Currently, we have a fully implemented version of all the components (models, plug-ins, architectural components, etc.) introduced earlier and working prototypes of several UIs that exhibit the required properties. In fact, all examples presented in the paper are realized using our research prototype. Ongoing work concentrates on several research lines. One is aiming to extend the widget archiving method to handle more complex and customized widgets intuitively. Another related line integrates the required revisions so as to further enhance the framework's capabilities to cope with more advanced affordances and quality attributes such as social translucence and UI plasticity in collaborative ubiquitous settings.

## ACKNOWLEDGMENTS

We gratefully acknowledge the support of iSTLab ([www.istl.teicrete.gr](http://www.istl.teicrete.gr)) and the ITEA2 UsiXML project. Jean Vanderdonckt would like to acknowledge of the ITEA2-Call3-2008026 USiXML (User Interface extensible Markup Language) European project and its support by Région Wallonne DGO6.

## REFERENCES

- Hill, J. and Gutwin, C. The MAUI toolkit: Groupware widgets for group awareness. In *Computer Supported Cooperative Work*, 2004, vol. 13, pp. 539-571.
- Lee, C., Helal, S., and Lee, W. Universal Interactions with Smart Spaces. *IEEE Pervasive Computing* 5 (Jan.-Mar. 2006), pp. 16-21.
- Savidis, A. and Stephanidis, C. Developing Dual User Interfaces for Integrating Blind and Sighted Users: the HOMER UIMS. In *Proc. of ACM Conf. on Human Factors in Computing Systems CHI'95* (Denver, 1995). ACM Press, New York (1995), pp. 106-113.
- Savidis, A., Stephanidis, C., and Akoumianakis, D. Unifying toolkit programming layers: a multi-purpose toolkit integration module. In *Proc. of the 4th Eurographics Workshop on Design, Specification and Verification of Interactive Systems DSV-IS'1997* (Granada, 1997), pp. 177-192.
- Demeure, A., Calvary, G., Coutaz, J., and Vanderdonckt, J. The Comets Inspector: Towards Run Time Plasticity Control based on a Semantic Network. In *Proc. of 5<sup>th</sup> Int. Workshop on Task Models and Diagrams for User Interface Design TAMODIA '2006* (Hasselt, 23-24 October 2006). K. Coninx, K. Luyten, K. Schneider (Eds.). Lecture Notes in Computer Science, vol. 4385, Springer-Verlag, Berlin (2007), pp. 324-338.
- Tandler, P. The BEACH application model and software framework for synchronous collaboration in ubiquitous computing environments. *Journal of Systems and Software* 29, 3 (2004), pp. 267-296.

7. Penichet, V., Lozano, M., Gallud, J., and Tesoriero, R. User Interface Analysis for Groupware Applications in the TOUCHE Process Model. *International Journal Advances in Engineering Software* (2009).
8. Molina, A.I., Redondo, M.A., Ortega, M., and Hoppe, H.U. CIAM: A methodology for the development of groupware user interfaces. *Journal of Universal Computer Science* (2007).
9. Garrido, J.L., Gea, M., and Rodríguez, M.L. Requirements engineering in cooperative systems, In *Requirements Engineering for Sociotechnical Systems*. Idea Group, Inc., (2005), pp. 226–244.
10. Molina, A.I., Redondo, M.A., and Ortega, M. A conceptual and methodological framework for modeling interactive groupware applications. In *Proc. of 12<sup>th</sup> International Workshop on Groupware CRIWG'2006* (Valadolid, 2006). Lecture Notes in Computer Science. Springer-Verlag, Berlin (2006).
11. Limbourg, Q. and Vanderdonckt, J. UsiXML: A User Interface Description Language Supporting Multiple Levels of Independence. In *Engineering Advanced Web Applications*, M. Matera, S. Comai, S. (Eds.). Rinton Press, Paramus (2004), pp. 325-338.
12. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J. A Unifying Reference Framework for Multi-Target User Interfaces. *Interacting with Computers* 15, 3 (2003), pp. 289–308.
13. Souchon, N. and Vanderdonckt, J. A Review of XML-Compliant User Interface Description Languages. In *Proc. of 10th Int. Conf. on Design, Specification, and Verification of Interactive Systems DSV-IS'2003* (Madeira, 4-6 June 2003). J. Jorge, N.J. Nunes, J. Cunha (Eds.). Lecture Notes in Computer Science, vol. 2844, Springer-Verlag, Berlin (2003), pp. 377–391.

# Technique-Independent Location-aware User Interfaces

Ricardo Tesoriero<sup>1,2</sup>, Jean Vanderdonckt<sup>2</sup>, and José A. Gallud<sup>1</sup>

<sup>1</sup>Computing Systems Department, University of Castilla-La Mancha  
Campus Universitario de Albacete, 02071, Albacete (Spain)

<sup>2</sup>Louvain School of Management, Université catholique de Louvain  
Place des Doyens, 1, B-1348, Louvain-la-Neuve (Belgium)

ricardo.tesoriero@uclm.es, {ricardo.tesoriero, jean.vanderdonckt}@uclouvain.be, jose.gallud@uclm.es

## ABSTRACT

Many techniques have been developed to support location awareness in user interfaces. Most of them are expressed at the code level, making them inflexible for modification of the location-awareness logic. They are very specific to a certain domain of application, making them hard to reuse or transfer. This paper introduces a model-based approach for specifying location-awareness throughout the user interface development life cycle from the task and domain level propagated until the final user interface level. In this manner, multiple techniques for location-awareness could be expressed and supported, with flexibility and reusability, thus obtaining technique-independent location-aware user interfaces. For this purpose, the space concept is defined at the task and domain levels and then linked to an extension of the concrete user interface model. In order to exemplify the model-based approach, some techniques for location-awareness will be expressed and, a case study will be explained: the Location-aware Remote Control system.

## Author Keywords

Location-awareness, model-based approach, region definition, space model, ubiquitous computing.

## General Terms

Design, Experimentation, Human Factors, Verification.

## Categories and Subject Descriptors

D2.2 [Software Engineering]: Design Tools and Techniques – *Modules and interfaces; user interfaces*. D2.m [Software Engineering]: Miscellaneous – Rapid Prototyping; reusable software. H5.2 [Information interfaces and presentation]: User Interfaces. I.2.4 [Artificial Intelligence]: Knowledge Representation Formalisms and Methods.

## INTRODUCTION

Heterogeneous interacting devices and surfaces are part of our surroundings in our lives. Through the popularity of mobile devices and communication technologies, the ubiquitous computing scenario envisioned by Weiser is becoming reality [27]. Consequently, a new set of interactive applications where the context of use, defined as “any information that can be used to characterize the situation of an entity” [7], allows applications to make assumptions about users’ current situation and act accordingly [26].

Thus, location-aware applications employ users’ location to interact with the system. There are many fields in which lo-

cation-aware applications are useful: home automation (e.g., location-aware remote controls adapt the User Interface to the nearest device), office automation where users are assisted by the system when locating resources (e.g., the nearest printer to print a document), m-learning applications where students perform learning activities “in-situ” (e.g., recognizing masterpieces in art galleries), outdoor guiding systems where users get information according to their position (e.g., the nearest restaurant), indoor guiding systems where visitors are guided through the building using a mobile device, multiplayer games where participants may win various points depending on where they are or where they have been through, and augmented reality.

Designing and developing location-aware UIs is still challenging today: there is no global conceptual approach for characterizing what the location is, how to properly characterize a location change, and how to express location-awareness logic. This logic is often produced programmatically, thus making it inflexible to modify. Multiple techniques for location-awareness exist based on different definitions, and relying on deployment platforms that are not interoperable, thus making them very specific to a domain, and hard to reuse in another domain of discourse.

In order to address these challenges, based on the related work section, this paper discusses motivations and working hypotheses for a model-based approach for location-awareness. Based on these requirements, the space model section defines a generic space conceptual model that enables expressing multiple location-aware techniques, rules, and logics thanks to a unified specification language, along with a model-based approach. This model-based approach consists of a series of models, and extensions of them, based on the UsiXML User Interface Description Language (UIDL) to capture the location-awareness aspects at different levels of abstraction. Afterwards, in the case of study section, we expose how models are defined in the location-aware remote control (LARC) scenario. Later, the implementation section exposes issues regarding the implementation of the editors used to define the models. Finally, we present conclusions and future work.

## RELATED WORK

In this section we expose the most relevant location technologies used by location-aware applications. Then, we expose different examples of applications in different do-

mains where location-awareness is a key issue to board. Finally, we analyze the Topiary environment used to prototype location-aware applications.

### Location technologies

According to the environment where the location awareness is achieved, the systems are classified into three categories: *indoor*, *outdoor*, and *hybrid* (that operate in both).

While most outdoor location systems are based on GPS technology, the technology employed in indoor environments is very diverse. For instance, [2] uses RF; Ekahau (<http://www.ekahau.com/>) employs Wi-Fi, BlueBot [19] uses RFID and RF technology, SmartFloor [16] uses pressure sensors, among many others.

Hybrid approaches are: Place Lab [11] that combines Bluetooth, GSM, and Wi-Fi technologies; “Bridging the gaps” [10] that combines an INS with infrared technology for drifting correction; the GETA Sandals [15] that combine dead-reckoning sensors with passive RFID technology, etc.

### Location-aware applications

In this section we describe some examples of location-aware applications with respect to their domain.

Regarding office automation, the system described in [3] presents information according to the office room the user is located. It also provides users with the ability to search for the nearest resources and attach UNIX directories to office rooms. In [1], the mobile device is used as a remote control. A system to check people in and out from a building is presented in [20], the Dynamic Ubiquitous Mobile Meeting Board (DUMMBO) detects the presence of two or more people in a meeting room and automatically gathers information for the meeting summary, such as participants, date and conversation recording.

Regarding indoor guiding systems, location-aware information retrieving applications are frequently exploited to guide visitors in cultural environments (such as, art galleries and museums). These applications: [14, 22] employ from RFID to IRdA technologies to carry out this task.

Regarding learning activities, the emergence of mobile learning applications used to teach “in-situ” or other places is obvious. Thus, location-awareness becomes a key factor in informal science settings and nomadic learning [8, 17].

To sum up, the use of location-aware applications is not restricted to a particular area or domain. Besides, they all use different techniques and technologies to accomplish their purposes. Therefore, we need a model that should be independent of the technology to be employed. To cope with this problem, we have studied the most relevant approaches that adhere to *Model-Driven Engineering* (MDE) and a selected set of the most relevant methods that cope with the development of context-aware applications. We observed that none of them targets the problem from the UI perspective explicitly.

The reusability issue of previously developed UIs is a key

concept from a software engineering perspective that is not tackled by any of the methods we analyzed. They are focused on the reuse of the *Platform Independent Models* (PIMs) to generate source code targeted for different platforms, but they do not support the reuse of higher level models, such as, reusing a previously designed task model in a new context of use that is similar to the previous one.

### The Topiary environment

Among others, Topiary [12] rapidly prototypes location-enhanced applications based on design patterns. It allows designers to create a map that models the location of people, places, and things. It uses an active map to demonstrate scenarios, depicting location contexts creating storyboards that describe interaction sequences with a wizard. While this certainly fosters rapid prototyping, there is no underlying model that captures the results of this prototype that could be further used in the development life cycle.

However, some important UI aspects were not covered by this approach: the application is not conceived with models at different levels of abstraction, leading to a huge lack of reuse of information when developing location-aware UIs. *Routes* are one of the most important aspects of location-aware applications. Topiary defines routes in a non-deterministic way, where the order of the location events cannot be specified properly. The route alternative (e.g., the capability to define the same behavior for different routes) is also not supported by the tool. To sum up, the actual approaches are not capable enough to support the development of multi-model and multi-technique UIs for location-aware applications efficiently.

Besides, the lack of explicit models for representing the location-awareness logic in the methods analyzed prevents designers from reusing parts or whole of a previously designed logic. Perhaps, it may not be an important issue when modeling logic for small scenarios such as, small buildings or houses; but it may result in an interesting resource when modeling complex buildings, such as town halls or big companies or hospitals, where the complexity of the application behavior regarding the location is crucial to the application success.

### MOTIVATIONS

As the result of the related work analysis we have arrived to the following motivations:

#### The need for a conceptual model

A location-aware UI could be considered basically as an Event-Condition-(re-)Action (ECA) system where rules express a location-aware logic decomposed into *events* (what occurred that requires some consideration of the location), *conditions* (under which circumstances do we need location-awareness, it is not necessary to do something for all changes of location), and *reactions* (what do we need to do in order to react to a change of location). Based on this decomposition, we have specialized Norman’s mental model in a variation for addressing location-awareness with seven stages for each entity (Figure 1):

1. *Goals for location-awareness*: any entity (i.e., user, UI, or external third party) may be responsible for specifying and maintaining a formal expression of goals for ensuring location-awareness. Although this process is ultimately intended to be beneficial for the end user, it could be achieved with respect to any location aspect. Such aspects could fall into three categories: *location-central information* (information that describes the location, such as coordinates, temperature, pressure, humidity, closeness), *location-peripheral information* (information related to the location, but not describing the location, such as vicinity, surroundings, connections with other regions), and *meta-location information* (information about location information, such as how the location is characterized). The goals are said to be *self-expressed*, *UI-expressed*, *locally* or *remotely*, depending on their locus of control: in the user's head, in the local UI, or in a remote system. In this work, we mainly assume that these goals will be expressed in the UI, with a specification language to be defined.
2. *Initiative for reaction*: this stage is refined into formulation for a reaction request, detection of a reaction need, and notification for a reaction request, depending on who is in charge: the user, the UI, or a third party.

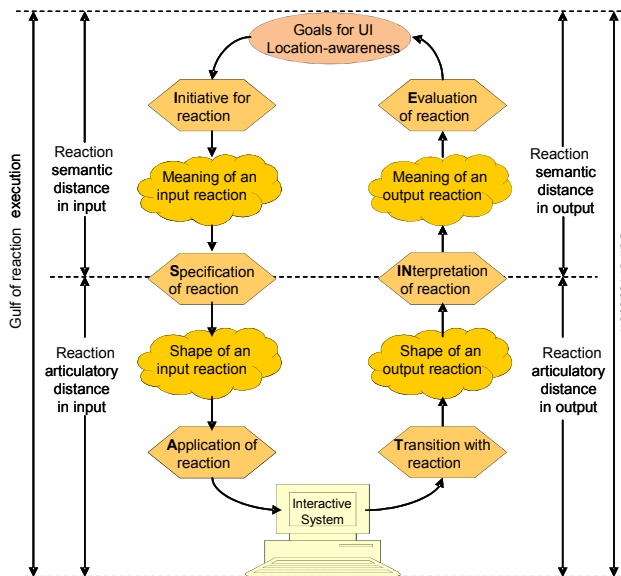


Figure 1. Norman seven stages revisited for location-awareness.

6. *Interpretation of reaction*: this stage specifies which entity will produce meaningful information in order to facilitate the understanding of the reaction of other entities. When the UI performs some reaction without any explanation, the end user may be confronted to a problem of understanding what has been changed depending on the location information type. When the user performs some reaction, she should teach the system how to interpret this reaction for future usage.

3. *Specification of reaction*: this stage is further refined in specification by demonstration, by computation, or by definition, depending on their origin: respectively, the user, the UI, or a third-party. When the user wants to trigger a reaction to a location change, she should specify the actions required to make this reaction, such as with programming by demonstration or by designating the operations required for this purpose. When the UI is responsible for this stage, it should compute one or several reaction proposals depending on location information available (of any type). When a third party specifies the reaction, a definition of the operations required to execute the reaction could be provided.
4. *Application of reaction*: this stage specifies which entity will apply the reaction specified in the previous stage. Since this adaptation is always applied on the UI, it should provide some mechanism to support it. If the user applies the reaction, UI mechanisms should be offered such as through customization or personalization.
5. *Transition with reaction*: this stage specifies which entity will ensure a smooth transition between the UI before and after the location change. For instance, if the system is responsible for this stage, it should provide some visualization techniques for explaining the transition.

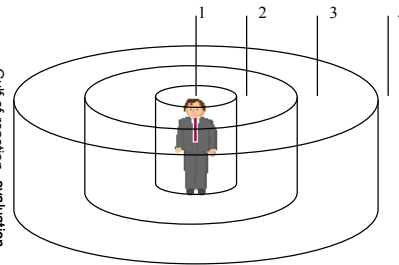


Figure 2. The four sociologic zones.

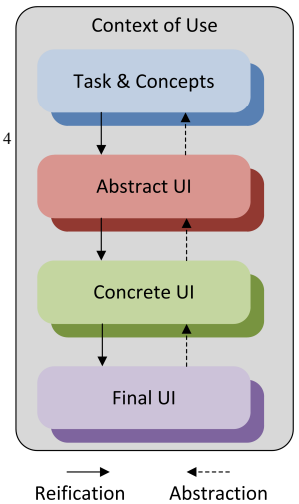


Figure 3. CRF simplified.

7. *Evaluation of a reaction*: this stage specifies the entity responsible for evaluating the quality of the location-awareness performed so that it will be possible to check whether the goals initially specified in Stage 1 are satisfied. If the UI maintains some specification of goals, it should be able to update these specifications according to the reactions executed. If the goals are in the users' mind, they could be also evaluated with respect to what has been conducted in the previous stages.



### The need for controlling the reaction

Another shortcoming found for location-aware UIs is that they can undertake reactions proactively. In terms of the model of Figure 1, it means that the application of the reactions, perhaps including the initiative and the specification that lead to this reaction, escaped from the user's control. For instance, Mobi-Timar [21] users are not allowed to change the reaction as it is pre-defined according to the required tasks for the user class depending on their location, but they can change some individual preferences, like interface layout or level of details. For instance, when a museum UI guides the visitor, it automatically prompts the description of the artwork that is closest to the visitor, thus preventing her from viewing the same artwork from a distant location. Therefore, a requirement states that the end user should be in control of the location-awareness process.

### The need for gathering feedback for reaction

In the museum example, if a user comes close to an artwork and the UI does not prompt anything new, the user may become puzzled by the UI reaction. The conditions for triggering the reaction could be satisfied, but they are not observable (perhaps browsable). Or they are not satisfied and the reasons why are not observable neither. Therefore, a need arises for expressing feedback, positive or negative, but also in terms of contents (e.g., why and why not?).

### The need for sociologic zone definition

Perceptual psychology may inform us to what extent a person considers an object far or close. Sociology interprets this as four spatial zones depending on the distance from subject (Figure 2). Central zone, respectively personal, social, public correspond to a focus distance from the user of 0 to 45cm, respectively 46cm to 1.2m, 1.3 to 3.6m, bigger than 3.6m [23]. This topic is also addressed in [5].

### The need for expressing location awareness logically

In order to express the location-awareness in a logical way, there is a need to rely on some models in order to state the ECA system. For instance, the reaction should be expressed logically on the UI independently of any location-awareness technique. For this purpose, a User Interface Description Language (UIDL) should be selected that is compatible with the Cameleon Reference Framework (CRF) [4] (a simplified version is reproduced in Figure 3, which shows the development process divided into four development steps). These steps are:

- In the *Tasks & Concepts* (T&C) step, we describe users' tasks to be carried out, and the domain-oriented concepts required to perform these tasks.
- In the *Abstract UI* (AUI) step, we define the abstract containers and the individual components [13] that will represent the artifacts on the UI. *Containers* are used to group subtasks according to various criteria (e.g., task model structural patterns, cognitive load analysis, and the identification of semantic relationships). *Individual component* represents an artifact that describes the behavior of a UI component in a modal-independent way

(navigation, task performance, etc.). Thus, an AUI abstracts a CUI with respect to interaction modality.

- In the *Concrete UI* (CUI) step, we concretize an abstract UI for a given context of use into Concrete Interaction Objects (CIOs) [24] defining the widget layouts and the interface navigation. It abstracts a FUI into a UI definition that is independent of any computing platform. The CUI can also be considered as a reification of an AUI at the upper level and an abstraction of the FUI with respect to the platform.
- In the *Final UI* (FUI) step, we represent the operational UI running on a particular computing platform either by interpretation (e.g., through a Web browser) or by execution (e.g., after code compilation)

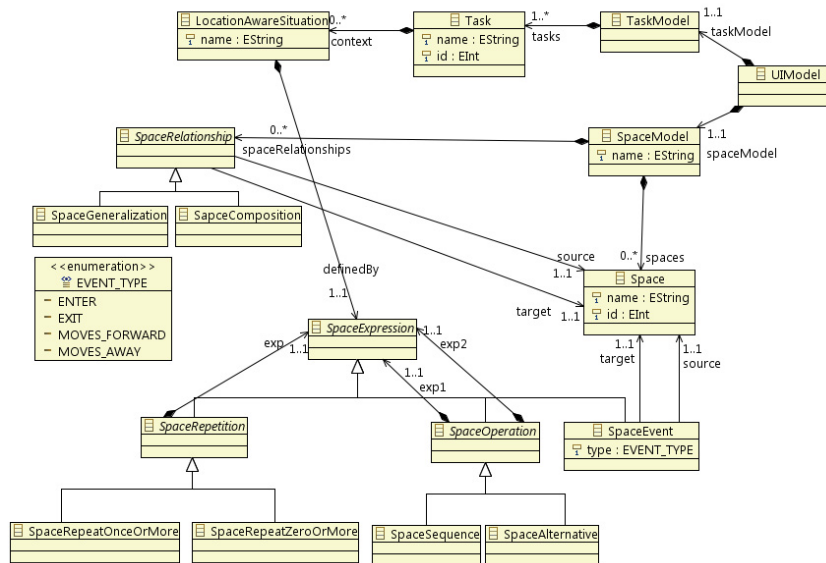
UIDL candidates that follow the CRF are: MariaXML (<http://giove.isti.cnr.it/tools/Mariae/>), UIML (<http://www.uiml.org>), UsiXML (<http://www.usixml.org>), or XIML (<http://www.ximl.org>).

We chose UsiXML [25] for the following additional criteria, but other UIDLs could be selected equally: the UsiXML modeling language must be described in terms of the Meta-Object Facility (MOF) language to enable the metamodels to be understood in a standard manner, which is a precondition for any activity to perform model-to-model (M2M) transformation and model-to-code (M2C) generation, a transformation model is also compliant with a metamodel expressed in the same way. To support the conceptual modeling of UIs and to describe UIs at various levels of abstractions, the following set of models is defined in UsiXML [25]:

- The *taskModel* describes the task as viewed by the end user interacting with the system. It represents decomposition and temporal relationships among tasks.
- The *domainModel* describes the classes of objects manipulated by a user while interacting with a system. Typically, it could be a UML class diagram or an entity-relationship-attribute model.
- The *mappingModel* contains a series of related mappings between models or elements of models. It gathers inter-model relationships that are semantically related (reification, abstraction and translation)
- The *contextModel* describes the three aspects of a context of use in which an end user is carrying out an interactive task with a specific computing platform in a given surrounding environment.
- The *auilModel* describes the UI at the abstract level as previously defined.
- The *cuiModel* describes the UI at the concrete level as previously defined.

As other UIDLs, UsiXML does not support location-awareness, but hold the appropriate concepts to express location-awareness logically. Thus, we propose to add a Space model to describe the location environment where users interact with the application. Besides, we extend the task, the AUI, and the CUI models accordingly.





**Figure 4. Space metamodel extension for Tasks & Concepts.**

## THE SPACE MODEL

The space model enables the definition of the space environment the application is aware of and how the user location affects the application.

The extension affects the Task & Concepts, the CUI, the Mapping and the FUI models of UsiXML. The AUI model was not extended because the mapping between AUIs and CUIs propagates all the vital information.

While in the Tasks & Concepts layer we describe the space environment and how it affects the application, in the CUI and FUI layer we describe how the information is perceived from the environment and how it is propagated to the application.

## The Task & Concept extension

The space metamodel extension for the Tasks & Concepts layer is depicted in Figure 4. The *SpaceModel* represents the space environment the application is aware of. It is the root of the space model and it is characterized by spaces and a set of space relationships among them.

### The space

*Spaces* represent location references that could be located and dimensioned by the system. The space concept covers a broad range of spaces according to the system capability. For instance, if the system is able to retrieve GPS coordinates of users and locate them in a city, a square, a building, a floor or a room, then all these physical spaces may be represented by spaces. However, the space concept is not restricted to physical space representations only. It may also be related to virtual representations, such as forms, dialogs [28], widgets, and so on in traditional UIs. They may also represent the virtual representation of a physical device, such as the mouse cursor. Even more, we can assign user spaces, to model colocation, or different sociological

zones for the same user, as discussed in the previous section (Figure 2).

This high-level abstraction provides designers with the ability to describe “mixed” relationships among virtual and physical spaces providing highly dynamic behavior to the application. For instance: click “lights on” button in my room turn the lights on.

To provide richer location aware situations *Spaces* may be related to other *Spaces* through *SpaceRelationships*. There are two types of space relationships: *SpaceCompositions* and *SpaceGeneralizations*.

### *The space composition*

*SpaceCompositions* represent a set of spaces that are treated as “the same” space. It means that if space *S* is composed by space *A* and space *B*. If user *U* enters *A*, then goes to *B* and then exits *B*. From *S* point of view, the user has entered *S* and then exited *S*. For instance, suppose that we have to model the personal presence in an office building. The “building” space should be composed by the “office” spaces, the “hall” spaces, the “elevator” spaces, and so on. Thus, to express that someone is outside we have to write “U out Building”.

### The space generalization

*SpaceGeneralizations* are used to express common behavior among a set of spaces. It means that if space *S* is generalizes space *A* and space *B*. If user *U* enters *A*, then goes to *B* and then exits *B*. From *S* point of view, the user has entered *A*, exited *A*, entered *B* and exited *B*. For instance, suppose that we have to model who is in the rest room in an office building. As there are several rest rooms in the building, we should define the “RestRoom” space as the generalization of all the rest room spaces of the building. Thus, to express that someone is in the rest room we have to write “U in RestRoom”.

#### The location-aware situation

The *LocationAwareSituation* is a key concept in the space model because it is “the link” between location awareness and the TaskModel. Through this relationship, and the relationship between the TaskModel and the DomainModel, the location information reaches to the core of the system. Thus, while the *Space* and the *SpaceRelationship* describe the space environment, *LocationAwareSituations* define how the environment affects system and user tasks. These situations are defined by *SpaceExpressions* that describe the situation.

#### The space expressions

Space expressions describe a location-aware state of the system in terms of regular expressions that are modeled following the Composite Design Pattern [9].

#### The space event

The simplest *SpaceExpression* is the *SpaceEvent*. It represents an event that relates two *Spaces*. Note that one of them is usually the user. There are 4 types of *SpaceEvents*, according to the patterns exposed in [12]. For instance: if *U* represents the *Space* for a user, and *X* an arbitrary *Space* (e.g., a Room), the *SpaceEvent* expression *U ENTER X* occurs when *U* entered in *X*. Or if *U* represents the *Space* for a user, and *X* an arbitrary *Space* (e.g., a Room), the *SpaceEvent* expression *U EXIT X* occurs when *U* left *X*. Besides, if *U* represents the *Space* for a user, and *X* an arbitrary *Space* (e.g., a sculpture), the *SpaceEvent* expression *U MOVES\_FORWARD X* occurs when *U* is approaching to *X*. Finally, if *U* represents the *Space* for a user, and *X* an arbitrary *Space* (e.g., a sculpture), the *SpaceEvent* expression *U MOVES\_AWAY X* occurs when *U* is going away from *X*. Note that exiting a region does not necessarily imply an entering in the next region: it depends how regions are configured together.

#### The space operations

*SpaceEvents* can be composed into *SpaceOperations*. A *SpaceOperation* represent a relationship between two *SpaceExpressions* (*exp1* and *exp2*). There are two types of *SpaceOperations*: *SpaceSequence* and *SpaceAlternative*. While the *SpaceSequence* represents a “path” of *SpaceEvents*, the *SpaceAlternative* represent a “selection” between two events.

#### The space sequence

The space sequence is used to identify patterns of behavior.

The tracking of the user position may result in a very valuable resource for both, controlling and gathering feedback for reactions. *SpaceSequences* can be used to make predictions and subsequently, ask the user for a decision (controlling reaction) or perform an action (provide feedback to the user). Thus, let *Seq (E<sub>1</sub>, E<sub>2</sub>)* be the *SpaceSequence* expression where *E<sub>1</sub>* and *E<sub>2</sub>* represent expressions, too. The situation *Seq(U ENTER S, U EXIT S)* occurs if the user has en-

tered and then exited space *S*. For instance, we want to be aware if the user *U* has gone to the cinema, to suggest him/her a place for dinner. *Seq(U ENTER CINEMA, U EXIT CINEMA)*.

#### The space alternative

The space alternative is a valuable resource when dealing with the same *LocationAwareSituation* in different *Spaces*. Thus, let *Alt (E<sub>1</sub>, E<sub>2</sub>)* be the *SpaceAlternative* expression where *E<sub>1</sub>* and *E<sub>2</sub>* represent expressions too. The situation *Alt(U ENTER S<sub>1</sub>, U ENTER S<sub>2</sub>)* occurs if the user has entered space *S<sub>1</sub>* or space *S<sub>2</sub>*. For instance, “set light intensity” to user defined if the user enters into the kitchen or the dining room *Alt(U ENTER KITCHEN, U ENTER DINING)*.

#### The space repetitions

The *SpaceRepetitions* expressions are analogous to the Kleene Closure in regular expressions. They are used to express repetition of expressions. There are two types of space expressions: *SpaceRepeatZeroOrMore* and *SpaceRepeatOneOrMore*. On the one hand, the *SpaceRepeatZeroOrMore* represents an optional repetition of a *SpaceExpression*; it means that the expression may occur or not, if it does it may occur more than once. On the other hand, the *SpaceRepeatOneOrMore* represents a mandatory repetition of a *SpaceExpression*; the expression occurs at least once. Therefore, let *Rep(E)* be the *SpaceRepeatZeroOrMore* expression where *E* expression too. The situation *Rep(Seq(U ENTER S, U EXIT S))* occurs if the user has entered and exited *S* more than once or has not entered and exited *S* at all. Similarly, let *Rep1(E)* be the *SpaceRepeatOneOrMore* expression where *E* expression too. The situation *Rep1(Seq(U ENTER S, U EXIT S))* occurs if the user has entered and exited *S* more than once. Finally, we present a shortcut to a very common *SpaceExpression*: *U IN S => Seq (U ENTER S, U EXIT S)*. For instance, suppose that we want to suggest a visitor *V*, in an art gallery, pieces of the same author. If all pieces of author *A* are generalized by the *A Space*, and the visitor space is represented by *V*, then the expression *Rep1(V ENTER A)* represents that the user have visited at least one piece of *A*.

#### The mapping, CUI and FUI extensions

These extensions deal with the lower level representation of the location awareness and the relationships among high-level and low-level models of the system. The Figure 5 shows the extension. The information is perceived by the system through *Sensors*. These *Sensors* provide data that is interpreted by *LocationInterpreters*. These *Interpreters* are in charge of turning the sensor data into space information that can be processed by the location system in a platform independent way. Thus, this information is matched to *LocationAwareSituations* that are relevant to the system in order to propagate this information through the tasks to the domain model.

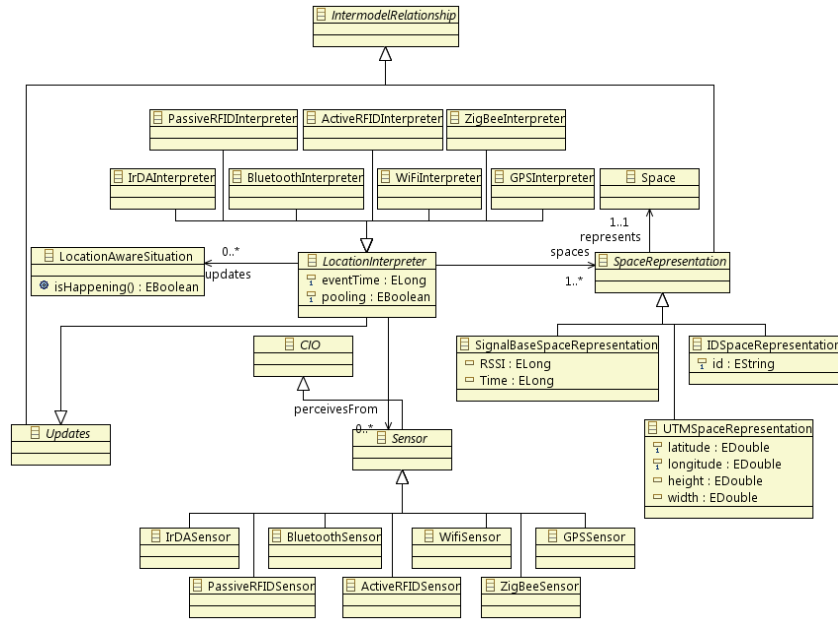


Figure 5. Space metamodel extension for CUI.

#### The space representation

The *SpaceRepresentation* provides an abstraction of the technology used by the location system. It provides a common identification for spaces in the system. Thus, when interpreters get data from sensors, they turn them into Spaces through the use of the *SpaceRepresentation* abstraction. Concretely, we defined three types of *SpaceRepresentations* according to the type of technology being used. The *IDSpaceRepresentation* is used by technologies that base object location on IDs. The *UTMSpaceRepresentation* is used by GPS based technologies. The *SignalSpaceRepresentation* is used by RF based technologies.

#### The location sensor

The *LocationSensor* is seen as a Concrete Interaction Object (CIO) that is defined in the Concrete User Interface (CUI) model. It is used to represent the technology, or set of them, to be used by the location system.

As the CUI model is part of the Platform Specific Model (PSM), the inclusion of new technologies or brands should be reflected into this model. We have initially introduced the Wi-Fi, the Bluetooth, the Passive and active RFID, the ZigBee and the GPS technologies. While whole new technologies should be added as new *Sensors*, new brands of products may be sub-classified from existing technologies to improve the reusability.

#### The location interpreter

*Location interpreters* are the glue that relates the spaces defined in the space model to spaces in the real or virtual world through the location sensors. In this way, *LocationAwareSituations* perceives the space environment that is forwarded to the domain model according to their relevance. To carry out this task a *LocationInterpreter* is defined for *Sensor* technology to translate information re-

ceived from sensors, and turn it into events that will be processed by *LocationAwareSituations* to check for the occurrence of them. Regarding infrastructure of the location system, it can be either centralized or distributed because the application interface between the *SpaceRepresentation* and the *LocationInterpreter* can be defined using the Proxy design pattern [9]

#### CASE STUDY

This section explains how to model location-aware characteristics of the Location-Aware Remote Control (LARC) application using the extensions of UsiXML to support location-awareness.

LARC is a home automation application that allows users to control different home devices through different UIs (one for each device) that are automatically displayed when the user is at the range of the device control. The Figure 6 (on the left) depicts the scenario with three devices: the TV, the air condition system and the electric blinds. The description is focused on the UI modeling because infrastructure issues are out of the scope of this article.

#### Space model

The space model is described with a UML class diagram like notation (see Figure 6 on the right). While the Space relationship is represented as an UML class without attributes or methods, the *SpaceComposition* and *SpaceGeneralization* relationships are described using the UML class composition and generalization respectively.

The physical spaces from where devices are controlled are represented by the **AC**, **TV** and **Blinds Spaces**. They are all part of the **Room Space**. Besides, we have defined three main virtual spaces representing the UI for each device. They are the **ACRC** for the AC remote control, the **TVRC** for the TV device and the **BlindRC** for the electric blinds

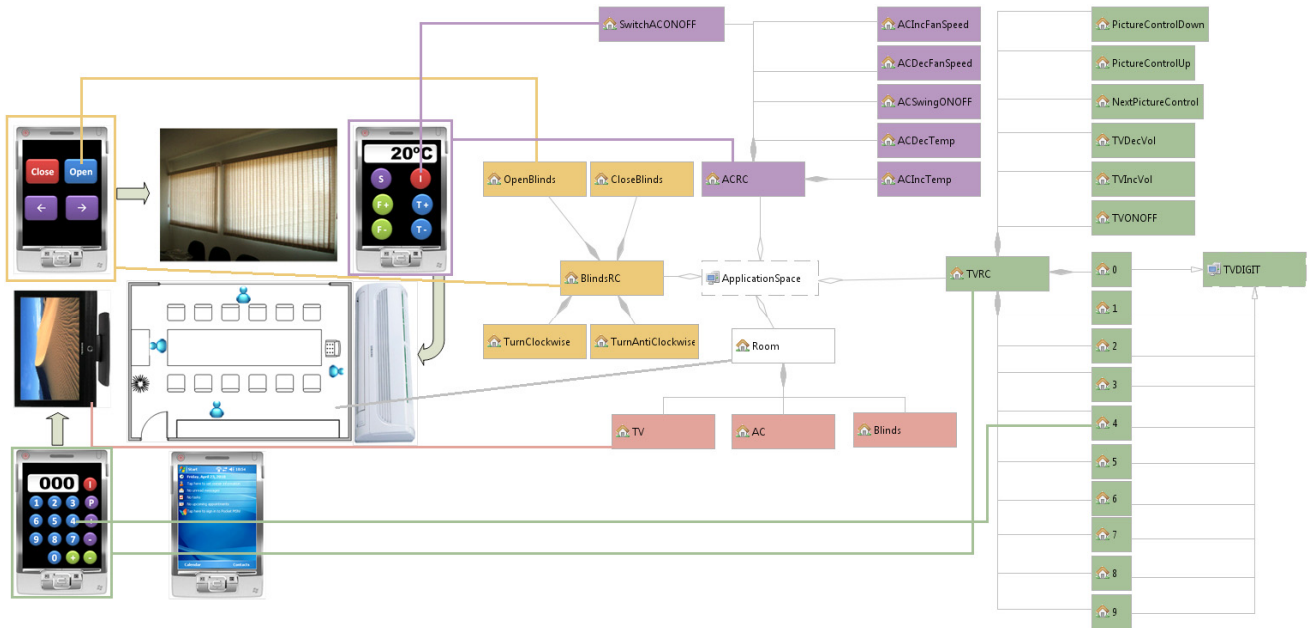


Figure 6. The LARC space model.

The space representation for each control is very similar. Therefore, we focus our attention on the **TVRC** representation. The **TVRC** is composed by a set of spaces that may represent the controls on the PDA screen (we say “may” because we do not know how they will be represented at this stage of the development process). There spaces are **0-9**, **TVONOFF**, **TVIncVol**, **TVDecVol**, **NextPictureControl**, **PictureUp** and **PictureDown**. In order to select a channel, the user introduces a sequence of digits. To define this sequence we need a common identified for all digits (0-9). Therefore, we have defined the **TVDIGIT** space as the generalization of all digits.

#### Task model

As we have mentioned before, the description is not complete. The Figure 7 depicts the Task model for the LARC.

As UsiXML describes task models using the CTT [18] notation, we use an extension to CTT notation in order to introduce the *LocationAwareSituation* concept. *LocationAwareSituations* are defined within Tasks. They define the *SpaceExpressions* using the textual notation we have described on previous section.

For the sake of clarity, the Figure 7 does not show the whole task model. Although it depicts the task model for all models, the only one that is complete is the AC remote control UI. However, the complete definition for the rest of the models is analogous to the AC remote control UI.

Analyzing the model we notice that the *ShowACUI* task is executed when the *OnEnterAC* situation matches the *U ENTER AC* space expression. It means that the AC Remote Control UI will be shown when the user U enters into the

AC space. This is a physical space relationship between the user and the control zone of the AC device. However, as we have mentioned before, we can define virtual relationships among physical and virtual spaces. For instance, The *SwitchACONOFF* task defines the *OnACSwitch* situation where the expression *U ENTER ACONOFF*, *U EXIT ACONOFF*, defines a relationship between the space that may be represented by the user’s finger on the PDA screen, and may be a Button control that is being displayed on the PDA screen. Thus, the space expression defines the “on click” event on the screen.

#### Concrete and Final User Interface

The concrete user interface regarding location awareness is based on the *LocationSensor* concrete interaction object. The *LocationSensor* provides the user interface with the information to act. It is related to the technology (RFID, GPS, etc.) employed. This information is interpreted by the *LocationInterpreter* according to the location technique (fingerprint, triangulation, etc.) employed. Thus, using *SpaceRepresentations* that are compatible with the technology and technique employed, the *LocationAwareSituation* is evaluated according to the space expression in order to provide the user interface with the situation occurred. The implementation of the approach is based on the mappings described by the *LocationInterpreter* (*LocationAwareSituation*, *Space*) through the *SpaceRepresentation*, and the *LocationAwareSituation* mapping between (*Space*, *Task*).

#### IMPLEMENTATION

Both, the Task and Space model editors were developed as Eclipse plugins, using the Eclipse Modeling Framework (EMF) and the Eclipse Graphical Modeling Project (GMF).

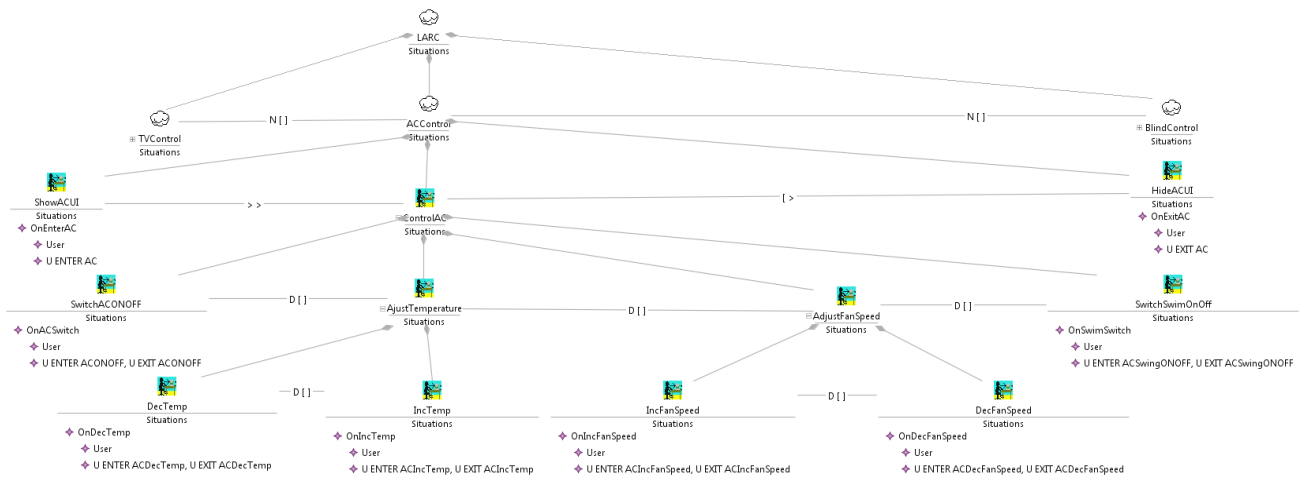


Figure 7. The LARC task model.

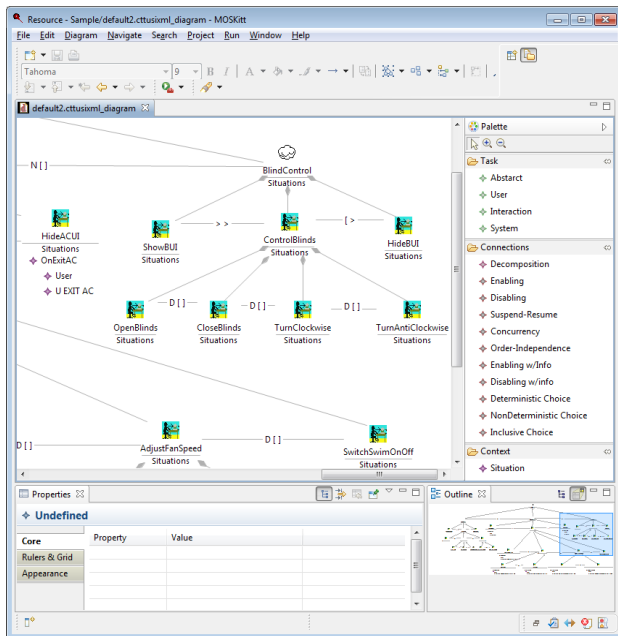


Figure 8. Task model editor.

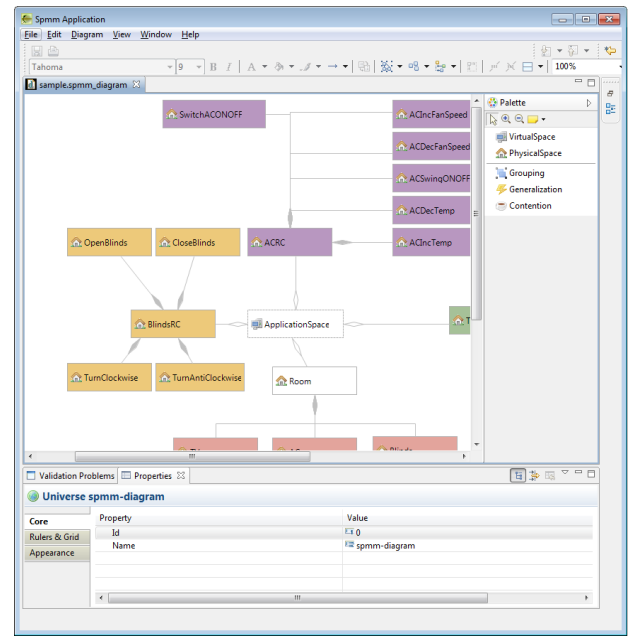


Figure 9. Space model editor.

Both editors are built on the base of Essential Meta-Object Facility (EMOF) metamodels using the ECORE format. Model validation is provided by the means of constraints defined in Object Constraint Language (OCL). Models are manipulated and stored in XMI (XML Metadata Interchange format).

To build the task model editor (see Figure 8), we have defined a modified version of the CTT where tasks can be related to different situations according to different location-aware situations. It provides designers with the ability to define space expressions for each situation. To build the space model editor (Figure 9), we have employed an UML-like Domain Specific Language (DSL) based on class diagrams.

## CONCLUSION AND FUTURE WORK

This work presents how to model and develop location-aware UIs employing the UsiXML methodology. To accomplish this goal we have extended the metamodels defined by the CRF process. As discussed in the Motivation section, actual approaches to develop location-aware applications lack of technology independence, conceptual space modeling, sociologic space support, and feedback for reaction, and control for reaction, too. To cope with technology independence we have adopted a MDA approach where the PIM allows designers to define applications that are platform independent. Thus, the PSM defined by the CUI model (*Sensors*, *SpaceRepresentations*, etc.) abstracts the technological issues regarding the location awareness.

The adoption of the MDA approach has also provided us with a CIM to represent the space environment using the space model where spaces can be modeled independently in a conceptual way. Regarding the need for conceptual model for spaces, the need for gathering feedback for reaction and the need for controlling the reaction, we have revisited the Norman seven stages to the space and task metamodels for location awareness as follows: the Goals for location-aware UI are defined by the Goals of the system, the *initiative for reaction* is represented by the *SpaceExpression*, the *specification of the reaction* is represented by a *LocationAwareSituation*, the *application of the reaction* is performed by the *Task*, the *transition with reaction* is represented by the relationship (*Task to Domain Model*), the *interpretation of the reaction* is exposed by the UI through the domain model, finally the *evaluation of the reaction* is performed by the User. Regarding the need for *sociological space*, we introduced a general definition of space where users have their own space that may be dynamically changed (the same user can be represented by different spaces according to the situation).

The idea of the homogeneous view of spaces is also very useful to represent virtual spaces such as UI artifacts. Thus, GUIs and physical spaces are abstracted to such level that space situations may reference any of them indifferently. As future work, we are following different research lines. The position is the initial point of the gesture.

Thus, we plan to introduce gestures to UsiXML methodology in order to extend its power of expression. Besides, the location awareness is just a subset of context awareness. Therefore, we are doing research on how to introduce other aspects of context awareness into UsiXML such as the physical environment or the infrastructure. Finally, the social environment is an important part of the context awareness. Therefore, the definition of the social structure of the application is a current key issue.

## ACKNOWLEDGMENTS

We gratefully acknowledge the support of the ITEA2 Call 3 UsiXML project under reference 20080026 by Région Wallonne DGO6 and the FP7-ICT5-258030 Serenoa project funded by the European Commission.

## REFERENCES

1. Anastasi, G., Bandelloni, R., Conti, M., Delmastro, F., Gregori, E., and Mainetto, G. Experimenting an indoor bluetooth-based positioning service, in *Proc. of ICDCS'2003*, 480–483.
2. Bahl, P., and Padmanabhan, V. N. RADAR: an in-building rf-based user location and tracking system in *Proc. of INFOCOM 2000*, IEEE, 775–784, 2000.
3. Brown, P. J., Bovey, J. D., and Chen, X. Context-aware applications: from the laboratory to the marketplace. *IEEE Personal Communications*, 4(5), 58–64, Oct. 1997.
4. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J. A unifying reference framework for multi-target user interfaces. *Interacting with Computers*, 15(3):289–308, 2003.
5. Prante, T., Röcker, C., Streitz, N. A., Stenzel, R., Magerkurth, C., van Alphen, D., Plewe, D. A.: Hello.Wall- Beyond Ambient Displays. In *Proc. of the 5th Intern. Conference on Ubiquitous Computing*, Seattle, Wash., USA, Oct. 12-15, 2003.
6. Chou, L.-D., Lee, C.-C., Lee, M.-Y., and Chang, C.-Y. A tour guide system for mobile learning in museums. In *Proc. of WMTE 2004*, IEEE Computer Society, 195–196, 2004.
7. Dey, A. K. Understanding and using context. *Personal and Ubiquitous Computing*, 5, 4–7, 2001.
8. Fleck, M., Frid, M., Kindberg, T., O Brein-Strain, E., Rajani, R., and Spasojevic, M. From Informing to Remembering: ubiquitous systems in interactive museums, *Pervasive Computing*, 1(2), 13-21, 2002.
9. Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*.
10. Hallaway, D., Feiner, S., and Höllerer, T. Bridging the gaps: Hybrid tracking for adaptive mobile augmented reality. *Applied Artificial Intelligence*, 25, 477–500, 2004.
11. LaMarca, A., Chawathe, Y., Consolvo, S., Hightower, J., Smith, I., Scott, J., Sohn, T., Howard, J., Hughes, J., Potter, F., Tabert, J., Powledge, P., Borriello, G., and Schilit, B. Place Lab: Device positioning using radio beacons in the wild, in *Proc. of PerCom'2005*, 116–133, 2005.
12. Li, Y., Hong, J. I., Landay, and J. A. Topiary: a tool for prototyping location-enhanced applications. In *Proc. of UIST'2004*.
13. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., and Victor López Jaquero. UsiXML: a Language Supporting Multi-Path Development of User Interfaces. In *Proc. of 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems EHCI-DSVIS'2004* (Hamburg, July 11-13, 2004). Lecture Notes in Computer Science, vol. 3425. Springer, Berlin (2004), pp. 200-220.
14. Long, S., Kooper, R., Abowd, G. D., and Atkeson, C. G. Rapid prototyping of mobile context-aware applications: The Cyberguide case study. In *Proc. of MobiCom'96*, 97–107.
15. Okuda K., Yeh, S., Wu, C., Chang, K., and Chu, H. The GETA sandals: A footprint location tracking system. In *Proc. of LoCA'2005*. LNCS, 3479, 120–131, 2005.
16. Orr, R., Orr, R. J., and Abowd, G. D. The smart floor: A mechanism for natural user identification and tracking. In *Proc. of CHI 2000*, ACM Press, 1–6, 2000.
17. Pascoe, J., Ryan, N., and Morse, D. Using while moving: HCI issues in fieldwork environments. *TOCHI* 7(3), 417-437, 2000.



18. Paternò, F. *Model-Based Design and Evaluation of Interactive Applications*. Applied Computing. Springer-Verlag, 1999.
19. Patil, A., Munson, J., Wood, D., and Cole, A. Bluebot: Asset tracking via robotic location crawling, *Computer Communications*, 31(6), 1067–1077, 2008.
20. Salber, D., Dey, A. K., and Abowd, G. D. The context toolkit: Aiding the development of context-enabled applications. In *Proc. of CHI'99*, 434–441, 1999.
21. Sharifi, G., Deters, R., Vassileva, J., Bull, S., and Röbig, H. Location-Aware Adaptive Interfaces for Information Access with Handheld Computers. LNCS, vol. 3137. Springer-Verlag, Berlin (2004), pp. 305–328.
22. Tesoriero, R., Gallud, J. A., Lozano, M. D., and Penichet, V. M.R. Tracking autonomous entities using RFID technology. *IEEE Trans. on Cons. Elec.* 55, 650–655, May 2009.
23. Trevisan, D.G., Gemo, M., Vanderdonckt, J., and Macq, B. Focus-Based Design of Mixed Reality Systems. In *Proc. of 3<sup>rd</sup> Int. Workshop on Task Models and Diagrams for user interface design TAMODIA'2004* (Prague, November 15–16, 2004). Ph. Palanque, P. Slavik, M. Winckler (Eds.). ACM Press, New York (2004), pp. 59–66.
24. Vanderdonckt, J., and Bodart, F. Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection. In *Proc. of the ACM Conf. on Human Factors in Computing Systems INTERCHI'93* (Amsterdam, 24–29 April 1993). ACM Press, New York (1993), pp. 424–429.
25. Vanderdonckt, J. A MDA-Compliant Environment for Developing User Interfaces of Information Systems. In *Proc. of 17th Conf. on Advanced Information Systems Engineering CAiSE'05* (Porto, 13–17 June, 2005). O. Pastor & J. Falcão e Cunha (Eds.). Lecture Notes in Computer Science, vol. 3520. Springer-Verlag, Berlin (2005), pp. 16–31.
26. Vanderdonckt, J., Grolaux, D., Van Roy, P., Limbourg, Q., Macq, B., and Michel, B. A Design Space for Context-Sensitive User Interfaces. In *Proc. of ISCA 14<sup>th</sup> Int. Conf. on Intelligent and Adaptive Systems and Software Engineering IASSE'2005* (Toronto, 20–22 July 2005). International Society for Computers and their Applications, Toronto (2005), pp. 207–214.
27. Weiser, M., and Brown, J. S. The coming age of calm technology. *Beyond Calculation: The Next Fifty Years of Computing*. Copernicus, 75–85, 1997.
28. Winckler, M., Trindade, F., Stanciulescu, A., and Vanderdonckt, J. Cascading Dialog Modeling with UsiXML. In *Proc. of 15th Int. Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'2008* (Kingston, July 16–18, 2008). Lecture Notes in Computer Sciences, vol. 5136. Springer, Berlin (2008), pp. 121–135

# Adaptive User Interface Support for Ubiquitous Computing Environments

Heiko Desruelle<sup>1</sup>, Dieter Blomme<sup>1</sup>, George Gionis<sup>2</sup>, and Frank Gielen<sup>1</sup>

<sup>1</sup>Ghent University – IBBT, Ghent, Belgium

<sup>2</sup>National Technical University of Athens – DSS lab, Athens, Greece

{heiko.desruelle, dieter.blomme, frank.gielen}@intec.ugent.be, gionis@epu.ntua.gr

## ABSTRACT

Application developers are increasingly facing the need to cover a wider variety of target devices. The diversity of devices supporting software applications is expanding from PC, to mobile, home entertainment systems, and even the automotive industry. Maintaining a viable balance between development costs and market coverage has turned out to be a challenging issue when developing applications for such a ubiquitous ecosystem. In this paper, we present the webinos approach as a means to enable adaptive user interfaces for web-based applications in ubiquitous computing environments. We propose a model-based user interface adaptation framework driven by a rich description of the target delivery context.

## Author Keywords

Abstract user interface, adaptation, context awareness.

## General Terms

Design, Reliability, Human Factors, Theory

## Categories and Subject Descriptors

D.2.2 [Software Engineering]: Design Tools and Techniques – *User interfaces*. H.5.2 [Information Interfaces and Presentation]: User Interfaces – *User-centered design*.

## INTRODUCTION

The availability of connected devices is growing rapidly. In our everyday life, we already use a multitude of personal devices that are connected to the Web. The number of shipped smart-phones at the end of 2010 even surpassed the traditional computer segments for the first time in the US [10]. From PC, to mobile, to home entertainment and even in-car units, consumers should prepare for a connected experience. Through the Web, applications can be accessed whenever and wherever the user wants, regardless of the type of device that is being used. However, from an application development perspective, ubiquitous environments generally introduce challenging and time-consuming requirements.

The variety of presentation and interaction modalities makes it very hard to support a wide range of devices. Even with standardized and cross-platform web technologies such as HTML, CSS, and JavaScript, efficiently managing ubiquitous variability points remains an important ongoing research topic [9].

Web-based software systems are traditionally modeled by separation of concerns. The models are engineered along three orthogonal dimensions: the development phases, the system's views, and its aspects (as illustrated in Figure 1) [12]. The phase dimension sets out the different stages of web development, ranging from analysis, to design and implementation. Each of these phases requires a number of specific views addressing the system's content, its navigation structure, and its presentation to the end-user. Finally, the aspects dimension defines both the structural and behavioral aspects of each of the view models. The growing importance of ubiquitous applications emphasizes the need for fragmentation management within this web engineering model. This concern has to be handled throughout every stage of the application's development life cycle. As proposed by Kappel et al., adaptability can be considered as an additional web engineering dimension, crosscutting all three other web modeling dimensions [11].

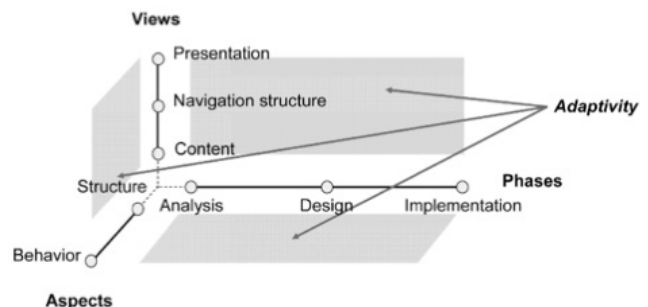


Figure 1. Adaptability as a crosscutting aspect on the traditional modeling dimensions of web engineering (from Koch et al. [12]).

For developers, the straightforward incorporation of such adaptability still remains an important challenge [17]. Ubiquitous applications should adapt dynamically to the current context of use, and even to contextual situations not foreseen at the application's design time. From this perspective, the webinos project aims to deliver a platform for web applications across mobile, PC, home media and in-car devices [22]. Webinos is a service platform project under the European Union's FP7 ICT Programme. The project represents a leap forward as a federated web runtime that offers a common set of APIs to allow applications to easily access cross-user, cross-service, and cross-device functionality in an open yet secure manner. Within the webinos

project, work is being done to achieve a maximum level of independence from the various underlying operating systems and hardware. The aim of our research is to propose a number of frameworks that enable the development of self-adaptive ubiquitous web applications. The Model Driven Development (MDD) approach provides very useful support for this type of challenge. In this paper, we focus on enabling the adaptability of user interfaces according to an approach derived from the CAMELEON Reference Framework (CRF) [3]. We achieve this goal by incorporating webinos runtime support for context-aware transformation of abstract user interfaces description models.

The remainder of this paper is structured as follows. Section 2 discusses related work. Section 3 provides background on the webinos project and discusses the core web application runtime platform. Section 4 elaborates on the webinos approach in offering adaptive user interface support to application developers. In section 5 we discuss a case study in the e-learning domain for providing learning assistance to students with a disability. This use case demonstrates the goal of our approach in order to reach adaptability support that is driven by various contextual dimensions. Finally, future work and our conclusion are presented in Section 6.

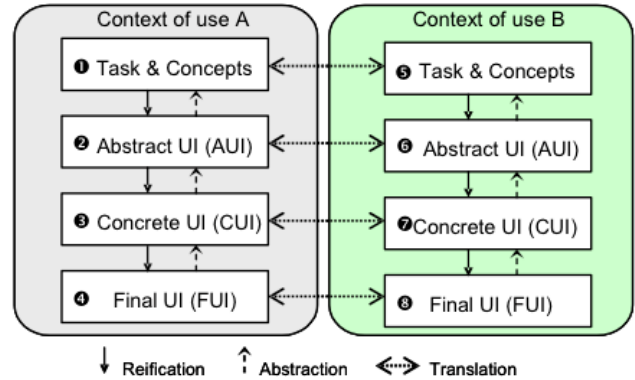
## RELATED WORK

The CAMELEON Reference Framework (CRF) [3] is a result of the EU-funded FP5 CAMELEON Project [4]. The framework defines a context-sensitive user interface development process. The process is driven by an intrinsic notion of the current user context, the environment context, as well as the platform context. According to the CRF approach, User Interface (UI) development consists of four subsequent stages:

1. Specification of the task and domain model, defining a user's required activities in order to reach his goals.
2. Definition of an abstract user interface (AUI) model. The AUI model expresses the application's interface independently from any of the delivery context attributes.
3. Definition of a concrete user interface (CUI) model, which generates a more concrete description of the AUI by including specific dependencies to the delivery context.
4. Specification of the final user interface (FUI), corresponding with the user interface in its runtime environment.

Figure 2 shows the interconnections and transformations between the above-mentioned CRF stages. The downward arrows depict reification processes. Reification is the transformation from a higher-level abstraction to a lower-level abstraction phase, hence inferring a more concrete UI description. The upward arrows, on the other hand, specify the abstraction processes. An abstraction is the inverse

transformation of reification. The third transformation type is the translation, depicted by the horizontal arrows. The translation deals with adapting the UI description to changes in one of the context of use models. In this case, the UI description's abstraction level remains the same when performing a translation.



**Figure 2. Context-aware UI development according to the CAMELEON Reference Framework (from Calvary *et al.* [3]).**

The Morfeo MyMobileWeb project [14] offers a framework that simplifies the development of web-based applications. MyMobileWeb specifically focuses on the mobile ecosystem. Alternative device spaces are not addressed by the project at this point. The framework applies a model-based approach to enable an automated application adaptation process based on the target delivery context.

MyMobileWeb uses the IDEAL2 language [6] to enable higher abstraction levels during the application's development. Adaptation decisions are made at runtime, based on a contextual description of the target platform. The context space is modeled according to the W3C Delivery Context Ontology [5]. The mappings between the AUI and CUI descriptions are expressed using the standardized syntax of Cascading Style Sheets Level 2 (CSS2) [1].

The Model-Based UI Working Group (MBUI WG) [21] is a recently chartered W3C working group as part of the consortium's Ubiquitous Web Activity (UWA) [18]. Its goal is to work on standards that enable the authoring of context-aware user interfaces for web applications. The MBUI WG aims to achieve this type of adaptivity by means of a model driven design approach. In this context, the semantically structured aspects of HTML5 will be used as key delivery platform for the applications' adaptive user interface.

## THE WEBINOS PLATFORM

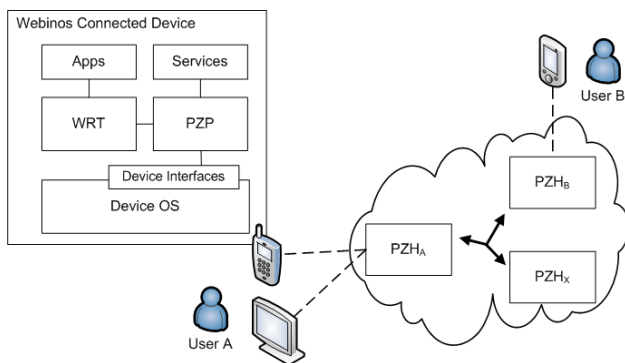
Webinos is an EU-funded project. The project aims to define and deliver an open source platform, to enable web-based applications and services to be used in a consistent and secure manner over a broad range of connected devices. The supported devices range from mobile, to desktop,

to home entertainment systems, and in-car units. In order to support this variety of devices and minimize the required efforts for application developers, webinos upholds a “single service for every device” ideology. Figure 3 depicts a high-level overview of the webinos platform structure. The system’s components are spread over the devices, as well as the cloud. The cloud components represent an important aspect of the platform, as these components enable users to access applications and services regardless of their device’s physical boundaries.

This seamless interconnection principle is centered around the notion of a so called *Personal Zone*. The Personal Zone groups a user’s personal devices and services. To enable external access to devices and services in the zone, the webinos platform defines the Personal Zone Hubs (PZH). Each user has his/her own PZH running in the cloud. This facilitates access to someone’s services over the Web from other devices.

The PZHs are federated and provide support for discovering other people’s hub, allowing users to easily share data and services. Although the system is designed with a strong focus on taking benefit from online usage, all devices in the Personal Zone have access to a local context model. This allows users to still operate their applications when being offline, or temporarily unable to access the Internet. Webinos provides offline support through the Personal Zone Proxy (PZP) component on the device. The PZP acts in place of the PZH when no Internet access is available.

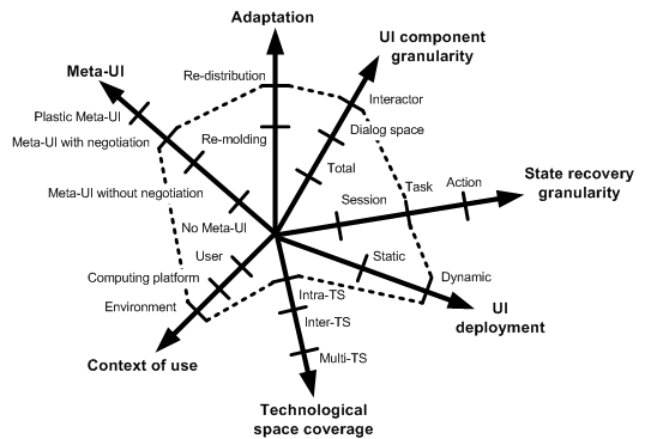
The webinos Web Runtime (WRT) component can be considered as an extension to a traditional browser. It is capable of rendering web applications specified using standardized web technologies such as HTML, CSS, and JavaScript. The webinos WRT maintains a tight binding with the local PZP. This binding allows the webinos WRT to be much more powerful than traditional browser-based application environments, as it enables the runtime to interface with local device APIs and services. Moreover, the PZP also allows the system to connect and synchronize with other webinos devices through its binding with the PZH.



**Figure 3. High-level overview of the webinos ubiquitous application runtime platform.**

## CONTEXT-DRIVEN USER INTERFACE ADAPTATION

As described in the previous section, the webinos platform uses the web to provide a rich delivery channel for ubiquitous applications. Nevertheless, there is still a need for optimizing applications to their specific delivery context. We use the Context-Aware Design Space (CADS) to visualize the required degree of application adaptability that is aimed for by the webinos platform (see Figure 4). The CADS is proposed by Vanderdonckt *et al.* [20] as a means to analyze and compare a software system’s dimensions subject to adaptation. By default, the design space defines seven adaptation dimensions, but it can be extended with additional dimensions based on the application domain’s specific needs. As shown in Figure 4, the strong fragmentation of ubiquitous computing in terms of interaction methods, hardware characteristics, software capabilities, etc. clearly requires a high level of adaptability.

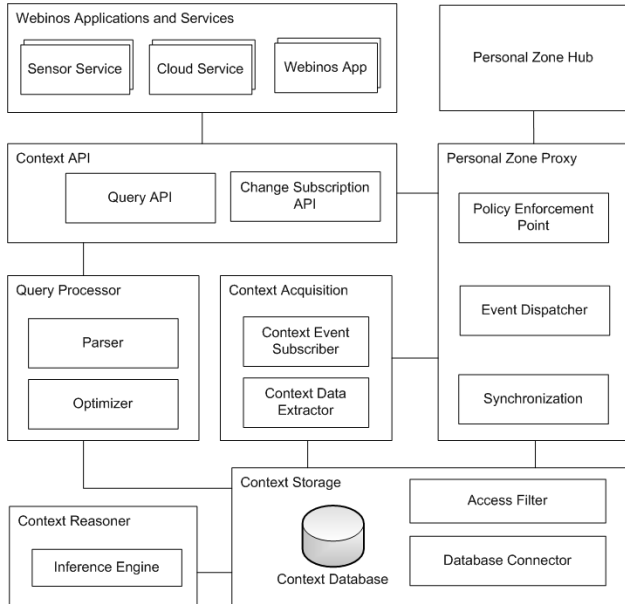


**Figure 4. Context-aware design space (CADS) visualizing the adaptability levels supported by the webinos platform for each of the design dimensions.**

## Webinos Context Model and Framework

Context management is an important aspect of the webinos platform. The webinos context model comprises four top-level submodels: the user context, the device context, the environment context, and the application context. The first three models are internally managed by the webinos system, whilst the application context model provides developers the opportunity to structure a situation’s contextual description from their application’s perspective. In addition, structuring the application’s context information allows users to more easily open up data access from other devices and services, or even allow access from the outside world (e.g., by friends, family, colleagues, etc.). The webinos context framework is built on top of these models. The framework is one of the core webinos service (see Figure 3) and provides the necessary functionality for acquiring, storing, and inferring rich contextualized data. Other webinos applications and services rely on this framework to support the need for context-awareness during the execution of their operations. The framework is responsible for extracting and storing context data through the identifica-

tion of specific context-related events that happen within webinos-enabled devices. In addition, the framework provides applications with an API to access such context information either by querying against the in-storage data or by being notified in real time regarding specific context changes. The context framework is closely coupled with the webinos policy and privacy enforcement framework in the PZP. This binding aims to ensure the secure handling of the often highly sensitive context data that is being stored and accessed. Figure 5 describes the conceptual architecture of the webinos context framework.



**Figure 5. Architectural overview of the webinos context framework.**

The Context API constitutes a component that enables applications to access the underlying volume of contextual data in a uniform way. The API provides interfaces that support two different modes for accessing context information:

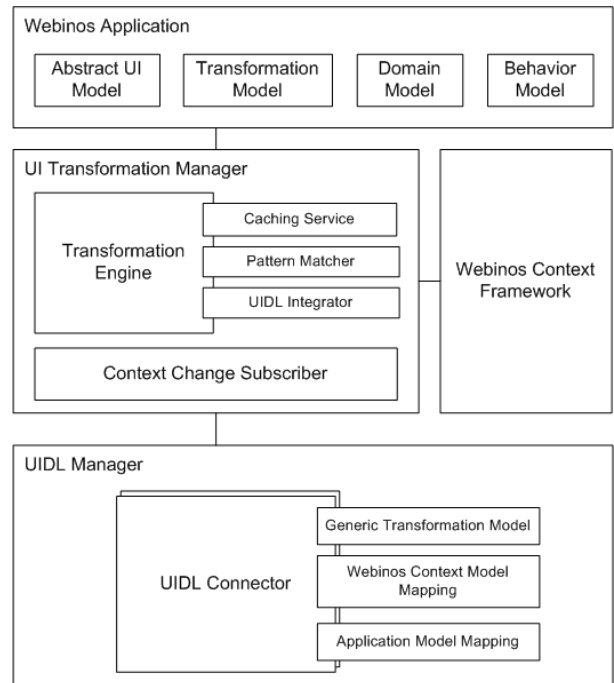
- The Query API enables applications to execute targeted queries for specific context data in the storage system.
- The Change Subscription API enables applications to subscribe for specific events that are triggered by a contextual change within webinos. Subsequently, subscribed applications are notified in real time when such events occur.

API access requests are passed to the Query Processor. The processor parses the request and checks its execution rights in collaboration with the PZP. In case the request is granted by the PZP, the query is optimized and executed. Besides acting as policy enforcement point, the PZP is also responsible for dispatching context events to the Context Acquisition component, and synchronizing contextual data between the local device and the Personal Zone (cfr. description of the general webinos architecture in Section 3).

### Webinos User Interface Framework

The UI Framework is another core webinos service (see Figure 6). It enables application developers to focus the design of their user interfaces on the AUI abstraction level. The framework takes care of processing UI-related platform and context dependencies by using the AUI model to drive the CUI generation in a (semi-)automated way. Whilst the framework is capable of generating CUIs without human intervention, webinos wants to keep developers in the loop and enables them to specify their own transformation requirements throughout the entire UI generation process. The framework addresses this requirement by providing pluggable support for User Interface Description Languages (UIDL). The extensible nature of the framework allows developers to use their preferred CAMELEON Reference Framework-compliant UIDL (e.g., UsiXML [19], or MariaXML [16]) to define their application's user interface. The only requirement to use a specific UIDL in the webinos UI framework is the presence of a connector component for this particular UIDL in the webinos UI framework. The UIDL-specific connectors are required in order to enable the framework's automated transformation inference.

The UI Transformation Manager component is at the heart of the framework. The component drives the transformation processes, aiming for a contextually optimized final user interface through a series of reification and translation operations. The Transformation Manager comprises an inference engine to attain this goal. The webinos UI transformation process relies on inference through dynamic pattern matching.



**Figure 6. Architectural overview of the webinos adaptive user interface framework.**

Each transformation rule  $\Phi$  within the transformation model can be represented as a conditional substitution operation:

$$\Phi := P? [S] : [T] . \quad (1)$$

The predicate  $P$  in Equation (1) is used to set the required condition before executing the transformation operation. The actual transformation is expressed in terms of the substitution  $S=e[l:=r]$ , consisting of an expression  $e$  which is matched for pattern  $l$  that in turn is substituted by the expression  $r$ .  $T$  is an optional substitution, executed in case the required condition  $P$  is not met. The structural patterns are searched for in the AUI model. This matching step is performed in order to detect structural abstractions (i.e., widget structures) within the application's user interface description. The transformation's precondition expression  $P$ , on the other hand, is based on the variables in the context model and the application's domain model. The transformation rules are selected from the transformation model. In case multiple transformation rules match the discovered UI pattern, the default conflict resolution strategy is to allocate higher priority to rules defined by the application developer. This way, application developers maintain maximum control over the transformation process.

#### CASE STUDY: LEARNING ASSISTANCE FOR DISABLED STUDENTS

In this section, we demonstrate the concepts of our approach based on a case study from the e-learning domain. The presented case aims to provide optimized e-learning facilities to students with a certain disability. For students with a disability, learning assistance services can be indispensable to the successful pursuit of education. In general, course material can significantly benefit from accessibility adaptations for sighted, blinds, hearing impaired persons, etc. This process is very resource consuming and often requires the allocation of dedicated caretakers. Technologies such as Braille readers, text-to-speech (TTS), and speech recognition have considerably increased the accessibility of user interfaces. Using these technologies, disabled persons are given the opportunity to become more independent from their environment. Nevertheless, every type of disability imposes its own usability and structuring requirements on applications. Providing end-users with an optimal experience requires developers to address each of these requirements individually. This approach leads to ad hoc development processes, in which developers have to create and maintain multiple versions of their learning application for each specifically supported disability.

This case study emphasizes the need for adaptability processes driven by the user context, as well as the device and environment context. The remainder of this section will elaborate on the dynamic adaptation of an e-learning AUI into a contextually optimized CUI, performed within the webinos UI framework. We will examine the case in which a user needs to be presented with an interactive form-based

user interface. Form-filling is a frequently performed action in e-learning environments (e.g., online assessments, submitting assignments, etc.).

#### Abstract User Interface Model

An application developer starts the UI design process by defining the AUI model. The developer can choose any CRF-compliant description language, as long as the webinos UI framework contains the necessary connector component for that specific UIDL. In this case, the developer needs to create an abstract form-based user interface for applying to a certain e-learning course and decides to use W3C XForms [] as abstract UI description language. The XForms description provides an overview of which control types should be presented in the UI, whilst remaining at a high abstraction level and not specifying how to exactly display these controls. A partial representation of the application form's AUI is shown in Figure 7.

```

1  <xforms xmlns:dm="http://www.myapp.com/domainmodel">
2  <group ref="dm:enrollment/student">
3    <group ref="dm:enrollment/student/name">
4      <input ref="dm:enrollment/student/name/first-name">
5        <label>Your first name</label>
6      </input>
7      ...
8    </group>
9
10   <input ref="dm:enrollment/student/born">
11     <label>Your birthday</label>
12   </input>
13
14   <group ref="dm:enrollment/student/address">
15     <label>Home address</label>
16     ...
17   </group>
18
19   <select ref="dm:enrollment/student/marital-status">
20     <label>Your marital status</label>
21     <item>
22       <label>Single</label>
23       <value>single</value>
24     </item>
25     ...
26     <item>
27       <label>Married</label>
28       <value>married</value>
29     </item>
30   </select>
31   ...
32
33   <submit submission="dm:enrollment">
34     <label>Submit</label>
35   </submit>
36
37 </group>
38 </xforms>

```

Figure 7. Partial representation of the AUI model for a context-aware e-learning application using the W3C XForms standard.

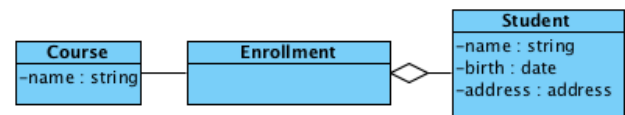


Figure 8. Simplified representation of e-learning application's domain model.



The form controls are abstracted to a level of input and selection components. Grouping controls provide a high level means of hierarchical interface structuring. These grouping controls will also help in the detection of widget structures by the UI Transformation Engine. Moreover, all controls contain a reference to their corresponding entity in the application's domain model (see Figure 8, and the webinos UI Framework description in Section 4). For form-based interfaces these bindings provide additional semantics regarding structure, data types, data ranges, etc. This information can in turn be exploited by the UI Transformation Engine as rich a-priori knowledge concerning the adaptation process.

### Transformation Model

The system's transformation model encompasses the reification and translation steps in order to obtain a final user interface that is optimized for the end-user's current context of use. The webinos UI Framework enables developers to fall back on a generic set of transformations provided by the platform. The standard set of transformations ranges from screen-fitting media adaptations, to adaptations based on generic accessibility and resource saving best practices [7,8]. On the other hand, application developers are still able to maintain control over the transformation process. Developers are free to refine their own transformation sets, specific to the application domain. As elaborated in Section 4, reification and translation rules can be expressed in terms of conditioned substitutions. Substitutions define structural changes to the AUI model. Furthermore, the link between the structural transformation and the conditionally required context attributes is realized through the specification of the transformations' conditional predicate description. These transformation preconditions can contain references to variables within the application's domain model, as well as the webinos Context Framework model. Nevertheless, based on privacy and security considerations, model queries are only permitted as long as the Personal Zone Proxy grants the application access to that specific type of information.

The process of filling in forms with a mobile device can be tedious task, as these devices lack the presence of a decent keyboard [15]. A potential translation rule for a form-based user interface in a mobile delivery context could be to match the AUI for optional input controls. If such a pattern is detected, the Transformation Manager can be instructed to remove the matching controls from the AUI. Alternatively, the translation rule can state to make such optional controls less obtrusive by rearranging them to the back of the form. Moreover, at the reification transformation level, the AUI can be transformed to a CUI with a look-and-feel matching the target device context.

On the other hand, taking the user context into account is also an important aspect. As indicated above in the case study description, the use of abstract user interfaces can support the process of optimizing UIs for people with spe-

cific disabilities. A useful reification rule in this context can be to transform AUI structures to a VoiceXML-based [13] CUI in case the end-user is blind or sighted. Figure 9 depicts the code snippet of a VoiceXML-based form filling CUI model after application of such reification ruleset. In the same way, motor impaired persons can be supported. E.g, by transforming the default rendering of form interaction controls in order to simplify their selection.

```

1  <vxml xmlns="http://www.w3.org/2001/vxml">
2  <form name="enrollment">
3    <field name="course">
4      <prompt>
5        Specify your course enrollment
6      </prompt>
7      <grammar src="elearning.xml#courses"
8        type="application/grammar+xml"/>
9    </field>
10   ...
11   <field name="birthday" type="date">
12     <prompt>
13       Enter your birthday
14     </prompt>
15   </field>
16   ...
17   <field name="marital-status">
18     <prompt>
19       Specify your marital status
20     </prompt>
21     <option>Single</option>
22     <option>Married</option>
23   </field>
24   ...
25   <block>
26     <submit />
27   </block>
28 </form>
29 </vxml>

```

**Figure 9. Partial representation of the VoiceXML-based CUI model for a context-aware e-learning application.**

### CONCLUSION AND FUTURE WORK

It is essential to provide end-users with an UI design that is optimized to their specific context of use. When developing for ubiquitous computing environments, this requirement is even further emphasized. It is not a sustainable business to require developers to manually address all variability points of the ubiquitous ecosystem. Such an approach only leads to developers rapidly loosing market share, and on the other hand entire consumer segments being ignored due to their marginal revenue potential. Hence, the development of ubiquitous applications requires a higher level of abstraction.

In this paper, we presented the webinos platform approach as a means to support adaptive ubiquitous user interfaces. We propose an adaptive UI framework driven by a detailed description of the target delivery context. The framework handles the processing of context related user interface dependencies, whilst still providing developers the means to keep control over the adaptation process. The UI adaptation is performed on a high abstraction level through the incorporation of the CAMELEON Reference Framework (CRF).

We are currently working on a reference implementation of the webinos platform. The extensive evaluation of our platform has yet to be carried out. An iterative evaluation process is planned throughout the implementation. Various test groups will be addressed in order to validate our approach from the perspective of developers as well as end-users.

#### ACKNOWLEDGEMENTS

The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7-ICT-2009-5) under grant agreement number 257103.

#### REFERENCES

1. Bos, B., Celik, T., Hickson, I., and Lie, H.W. Cascading Style Sheets Level 2. <http://www.w3.org/TR/CSS2>
2. Boyer, J.M. XForms 1.1. <http://www.w3.org/TR/xforms>
3. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L. and Vanderdonckt, J. A Unifying Reference Framework for Multi-Target User Interfaces. *Interacting with Computers* 15, 3 (2003), pp. 289–308.
4. Cameleon project, <http://giove.isti.cnr.it/projects/cameleon.html>
5. Cantera, C.M. and Rhys, L. Delivery Context Ontology. <http://www.w3.org/TR/dcontology>
6. Cantera, C.M., Roderiguez, C., and Diaz, J.L. IDEAL2. <https://files.morfeo-project.org/mymobileweb/public/specs/ideal>
7. Chisholm, W., Vanderheiden, G., and Jacobs, I. Web Content Accessibility Guidelines 1.0. <http://www.w3.org/TR/WAI-WEBCONTENT>
8. Connors, A. and Sullivan, B. Mobile Web Application Best Practices. <http://www.w3.org/TR/mwabp>
9. Frederick, G.R. and Lal, R. The future of the mobile web. *Beginning smartphone web development*. Springer, Berlin (2009) pp. 303–313.
10. International Data Corporation (IDC). <http://www.idc.com/research>
11. Kappel, G., Proll, B., Retschitzegger, W., and Schwinger, W. Modeling ubiquitous web applications: the WUML approach. *Conceptual Modeling for New Information Systems Technologies*. Springer (2002), pp. 183–197.
12. Koch, N., Knapp, A., Zhang, G., and Baumeister, H. UML-Based web engineering. *Web engineering: modeling and implementing web applications*. Springer, Berlin (2008), pp.157–191.
13. McGlashan, S., Burnett, D.C., Akolkar, R., Auburn, R.J., Baggia, P., Barnett, J., Bodell, M., Carter, J., Deshmukh, M., Oshry, M., Rehor, K., Yang, X., Young, M., and Hosh, R. Voice Extensible Markup Language (VoiceXML). <http://www.w3.org/TR/voicexml30>
14. Morfeo MyMobileWeb project, <http://mymobileweb.morfeo-project.org>
15. Nakagawa, T. and Uwano, H. Usability Evaluation for Software Keyboard on High-Performance Mobile Devices. In *Proc. of HCI International 2011*. Springer, Berlin (2011), pp. 181–185.
16. Paterno, F., Santoro, C., and Spano, L.D. MARIA: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. *ACM Trans. Comput.-Hum. Interact.*, (2009), pp. 324–353.
17. Schauerhuber, A., Wimmer, M., Schwinger, W., Kapsammer, E., and Retschitzegger, W. Aspect-oriented modeling of ubiquitous web applications: the aspectWebML approach. In *Proc. of 14<sup>th</sup> International Conference and Workshops on the Engineering of Computer-Based Systems*, (2007), pp. 569–576.
18. W3C Ubiquitous Web Applications Activity. <http://www.w3.org/2007/uwa>
19. UsiXML, <http://www.usixml.org>
20. Vanderdonckt, J., Coutaz, D., Calvary, G., and Stanculescu, A. Multimodality for Plastic User Interfaces: Models, Methods, and Principles. In *Multimodal user interfaces: signals and communication technology*, Lecture Notes in Electrical Engineering. Springer-Verlag, Berlin (2007), pp. 61–84.
21. W3C Model-Based UI Working Group Charter. <http://www.w3.org/2011/01/mbui-wg-charter.html>
22. Webinos project, <http://www.webinos.org>

# Supporting Models for the Generation of Personalized User Interfaces with UIML

Firas Bacha, Káthia Marçal de Oliveira, Mourad Abed

UVHC, LAMIH, FRE CNRS 3304, F-59313 Valenciennes, France  
{firas.bacha, kathia.oliveira, mourad.abed}@univ-valenciennes.fr

## ABSTRACT

User interface (UI) personalization aims at providing the right information, at the right time and on the right support. Personalization can be performed on the interface containers (e.g. layout, screen size and resolution); and on the content provided in the interface (e.g., data, information, document). This paper explores the content personalization in the description of UI using a well-know User Interface Description Language (UIDL), named User Interface Markup Language (UIML). To address this goal, we defined a set of personalization models that are used to annotate a UI task model. By transforming this model in a model-driven architecture implementation, one can be able to generate a UIML code to generate the final UI.

## Author Keywords

Content personalization, Personalization models, Annotation, BPMN, UIML.

## General Terms

Design, Reliability, Human Factors, Theory.

## Categories and Subject Descriptors

D.2.2 [Software Engineering]: Design Tools and Techniques – *User interfaces*. H.5.2 [Information Interfaces and Presentation]: User Interfaces – *User-centered design*.

## INTRODUCTION

The relevance of the delivered information, its intelligibility and its adaptation to the usages and preferences of users are key factors for success or rejection of information systems [6]. Indeed, proliferation and continuing growth of different computer devices with several interaction modes allowing users to access information anywhere and anytime, brings new personalization challenges to these systems. In fact, systems should be adapted to the context changes [**Error! Reference source not found.**].

Recognizing this fact, some initiatives [10] [5] [7] considering personalization aspects in the user interface design emerged. Personalization can be performed on the interface containers (i.e., layout, colors, sizes, and other design elements), and on the content (data, information, document) provided in the User Interface (UI) [7]. Unlike content personalization, many works explore the personalization of containers for User Interface (UIs).

According to several approaches that consider UI adaptation since design time and specially the CAMELEON Ref-

erence Framework (CRF) [8], the context of interaction with the UI and the business domain are two elements that are essential to implement this adaptation.

In this context, we decided to implement UI content personalization by using a User Interface Description Language (UIDL) that could describe the personalized UI. For that reason, we proposed a set of models, called personalization models that contain a context model and a domain ontology which are linked with a mapping model. The personalization models will serve to annotate a UI task model, which is the high-level specification of a UI. By transforming this model, one can be able to generate a User Interface Markup Language (UIML) [12] code, our chosen UIDL that describes the personalized UI.

The reminder of this paper proceeds as follows. We first present briefly the definition and some studies about personalization. Then we define our proposed models. In section 4 we explain the way this models could support UIML, to generate personalized UIs. Finally we present our conclusion and ongoing works.

## USER INTERFACE PERSONALIZATION

There is no consensual definition of personalization. Usually, authors define personalization based on their specific goals and applications [6]. In this article, we can say that personalization deals with the capacity of adaptation of a UI considering some information related to this user and his context. This adaptation can be performed basically on the interface containers presentation; i.e., layout, colors, sizes, and other design elements; and, on the content elements; e.g., data, information, document [7].

To allow the UI personalization or adaptation, several authors proposed to use context models (e.g., [8] [14]). When a system uses context to provide relevant information and/or services to the user, it is considered context-awareness. A context is defined [11] as any information that can be used to characterize the situation of an entity (person, place, or object) considered relevant to an interaction between a user and a system. [8] named context as context of use, which is composed of three classes of entities: user, platform and physical environment where the interaction takes place. In this paper, we use the term “context”

The personalization of UIs has been studied by different research groups in Human-Computer Interaction community. The consideration of content personalization is not inte-

grated in proposed UIDLs. Even if they were used within declarative approaches, the main focus of these formalisms was the container adaptation. One of the most important works is the CAMELEON Reference Framework [8]. In this framework, the UI development is done through a set of models and transformations according to the context of use. Four models are proposed: task and concepts models, that describes the user's tasks to be carried out and the domain-oriented concepts required by these tasks; abstract UI, that describes a UI independent of any modality of interaction; concrete UI, that concretizes an abstract UI for a given context of use; and, the final UI, that is the operational UI running in a specific platform.

Several proposals, interested in UI personalization, were defined based on CAMELEON. UsiXML [18,19] is a UIDL that provides a high level of abstraction for the design of the UI, using a set of models including a domain, a context and a mapping models and providing a complete XML based language for modeling UIs. This language was also used to define an environment for the generation of multi-platform interfaces, and for different contexts of use in an MDA approach [14]. Although, the authors proposed several UI designing models, only the container personalization was referred either for the language or for the approach.

Ali [1] proposes an approach for the generation of multi-platform UIs, by extending a task model specified in CTT [17] which will be transformed into UIML code. The approach and the generated code did not consider any form of UI adaptation.

Brossard et al. [7] proposed a methodology for the design of personalized information system for the transportation domain that, like us, aims to perform content personalization at runtime. Contrariwise, this proposal is specific to transportation software development and authors do not define a context model to be used in UI design. Indeed, the authors did not use any UIDL to define their UIs.

#### INTEGRATING CONTENT PERSONALIZATION IN UIML

We recall that our main goal is to include some content personalization into the UIML description of a specific UI, that is, to define, when possible, which personalized information should be provided for each UI entry (input/output). To address this goal and following the CAMELEON framework [8], we consider the information about the domain application to which the system is developed; we consider also the information about the context of use [8], since delivered information depends on the context; and finally we try to include these two aspects into the UIML code in order to enrich and to personalize the UI description.

To define the domain model, one can use class diagram or domain ontology. The importance is that this domain model works as a vocabulary of the domain for the whole application components, i.e., UI, implemented software functions and, mainly, the database definition since the content to be

provided in the UI comes from the database of the system application.

The context model is necessary to provide the relevant information that should be taken into account to provide personalized information. In this way the information of this context should be directly related to the domain information (some authors, even choose to define context models specific for a particular domain [9] [13]). We decided to use a generic context model and to associate it with the domain information to assure the personalization. We need, therefore, to describe "relations of influence" from context model elements to domain model elements. We call these relationships "mappings".

Moreover, it was clear for us that we have to consider these models into the UIML code. To reach our goal, we decided to integrate our models within a MDA architecture [15] where the Computational Independent Model (CIM) level is composed of the UI task model that is annotated with the personalization information. By transforming the UI task model, one can be able to generate a UIML code that describes the final personalized UI and that defines the Platform-Independent Model (PIM) and the Platform-Specific Model (PSM) levels of that architecture.

Next section presents each one of those models.

#### UI DESIGNING MODELS DEFINITION

##### Domain model

In order to define our domain model, we decided to use domain ontology since it represents the concepts of the domain independent of the system application. Domain ontologies express conceptualizations (i.e., description of entities and their properties, relationships, and constraints) that are related to a specific domain (e.g., medicine or transportation) and it captures the knowledge of a domain to be used in several applications from the same domain.

Since we are interested in content personalization the ontology will also play the role of an interface to access to the data which represents instances of that ontology.

##### Context model

As defined previously the context model should contain information about the user, platform and environment. To define this model, we did a large literature review. Although eighteen context model propositions were found none of them was considered complete. Some propositions considered only one of the context dimensions (user, platform or environment). Others were specific to a particular domain (e.g. smart phones, e-commerce) and/or not enough detailed. We decided, therefore, to integrate the main information from all propositions and to propose our context model [2] (See figure 1).

The central element of the context model is the *user profile* that is composed of five major parts allowing to specify the user when interacting with the final interface: *Contact information* that contains personnel data; *Demographic information* that contains basic and unchanged user data.

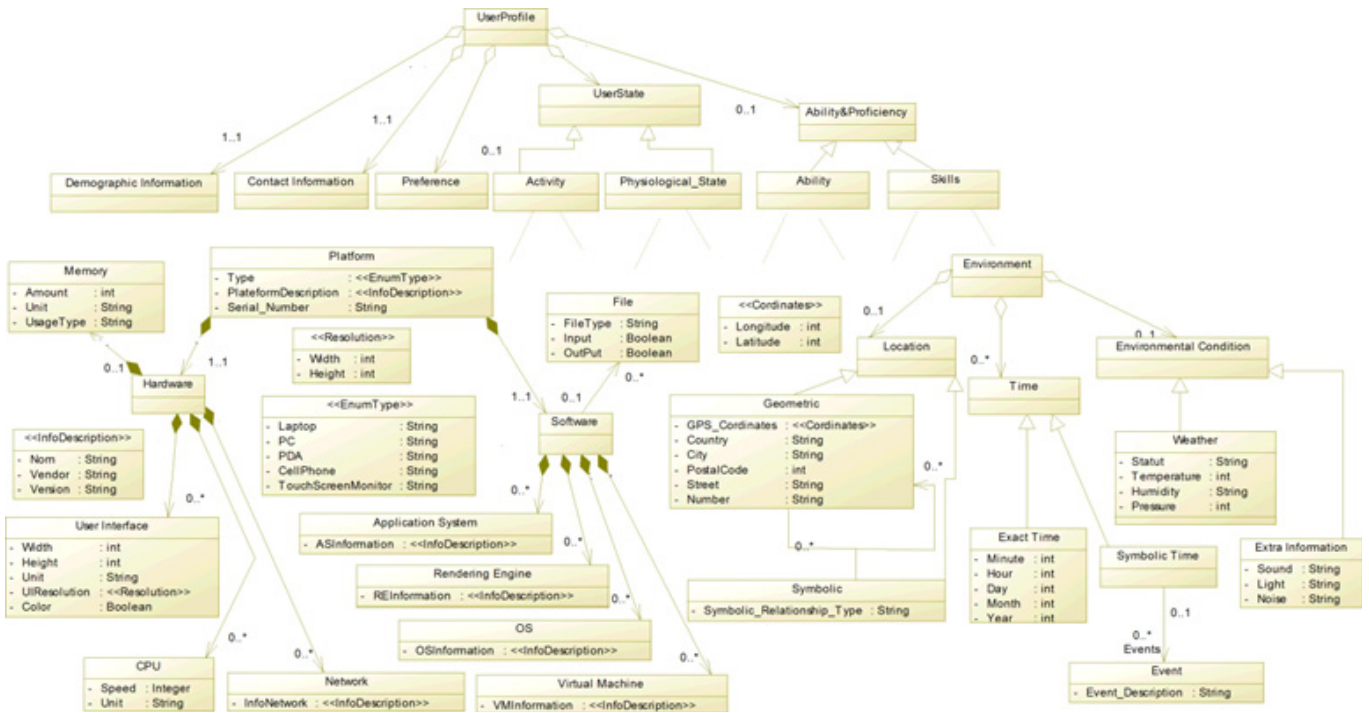


Figure 1. Proposed context model.

*Preference* to describe user interests and preferences; *User State* to describe the physiological user state and the activity s/he is practicing; and finally *Ability and Proficiency* that specify the user skills and abilities. The information about the platform used by the user is organized in two main classes: *Hardware*, describing the physical aspects; and, *Software*, with description of computer systems. The information about environment is organized in three main classes: *Location* refers to the place where the user is located at the time of interaction (the geometric, i.e. exact, location and the symbolic that is relative to another location); *Time* that describes the interaction moment (exact time or by a symbolic one - summer, school holidays...) and the *Environmental Condition* identified at the moment of user interaction.

#### Mapping between domain and context models

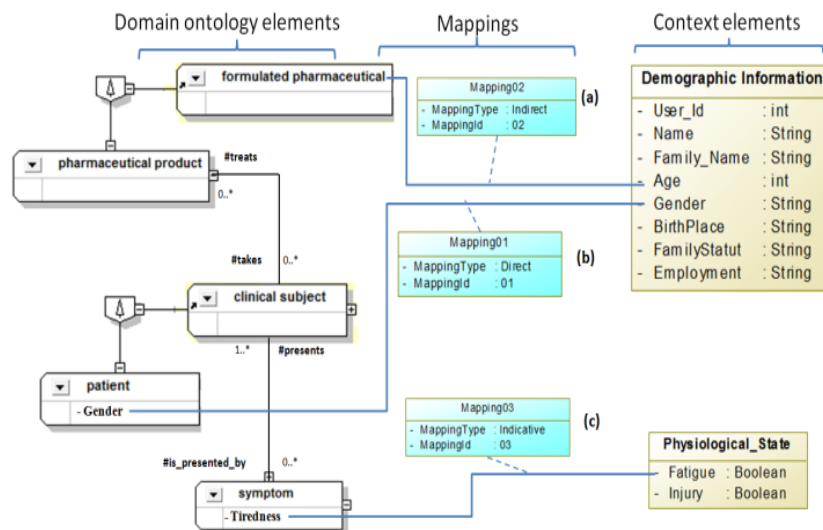
Since the context model is independent of domain and we look for personalization for a specific application domain, we have to find a correspondence between the context model and the domain specific ontology. For that reason, we created the mapping model that defines the relationship between the context elements and the ontology ones (see figure 3). In fact, the domain elements (classes or attributes) should be analyzed against the context model elements looking for each context element that could influence the domain concept. Once we find an element, we should set a mapping with the domain concept. In this way the information provided in the UI will be personalized considering the context. Analyzing domain models, we identified three main cases of mappings.

The first one refers to concepts that are exactly the same information present in the context model although, sometimes, with different name. That means the information of the context has a direct influence on the content of the same information of the domain model. We say, therefore, that we have a *direct mapping*. Figure 2 shows an example of mapping of a domain ontology adapted from the translation medicine domain ontology [4]. For example in figure 2 (b), since the *Gender* ontology attribute and the *Gender* context attribute are synonyms, we can set a direct mapping between them.

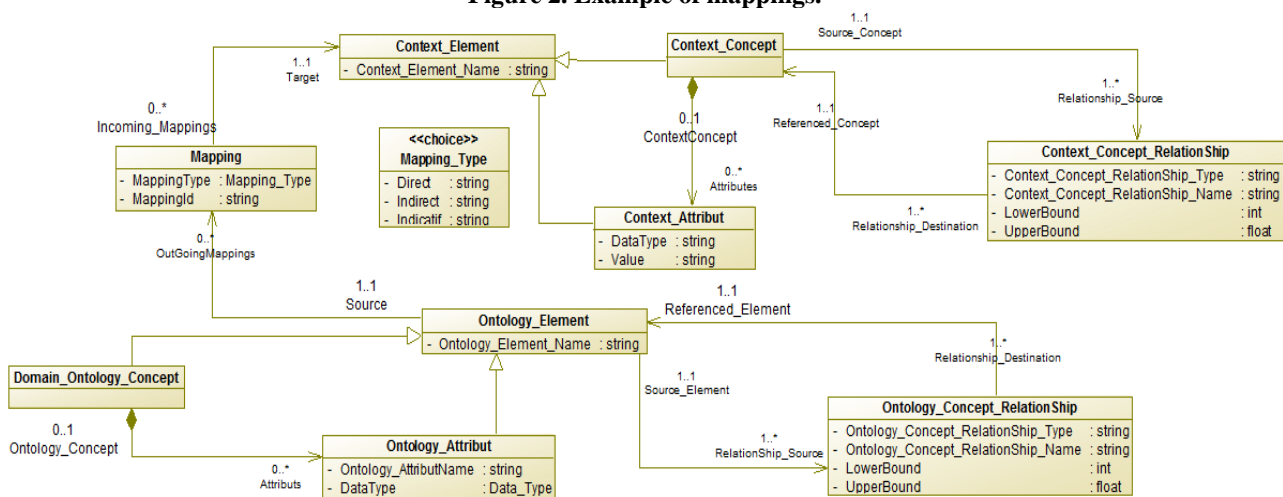
A second case refers to the class attributes from the context model that just indicates a specific state of the user. They can influence some domain concept by defining the existence or absence of that information. We say, therefore, that we have an *indicative mapping*. It is similar to the direct mapping, except that in this case the context attribute element must have the *Boolean* type.(See figure 2 (c)).

Finally, there are some concepts from context model that could have an indirect influence on the domain model elements. To define an indirect mapping, the designer should verify if there is any information in the domain model that could change depending on some data modeled in the context. In figure 2 (c), the *FormulatedPharmaceutical* domain concept is indirectly associated with *Age* attribute from *Demographic Information* class. That means that the chemical substance of a medicine (*PharmaceuticalProduct*) depends on the user age (for example, some drugs are used only for adults).





**Figure 2. Example of mappings.**



**Figure 3. Meta-model for mapping context and ontology elements.**

Figure 4 presents the algorithm that a designer could follow when defining mappings. In what follows, we refer to an ontology element (classes or attributes) by “o” and to a context model element by “c”. “O” represents the set of all ontology elements and “C” all context attributes.

## Annotation of UI Task Model

UI task models have been used as a high-level specification for UI design. Several notation has been used (e.g., Concur Task Tree (CTT) [17], Business Process Modeling Notation (BPMN) [16]). We used BPMN because of its capability to model the passage of information flow and the dynamic application aspect. It allows modeling both human and non-human tasks and it allows specifying for each task, the actor performing it. In general the BPMN is used as a task model description as follows [7]: the processes and sub-processes are set of elementary tasks that represent the interactive and non-interactive task of a UI.

```

Begin
  For "o" in "O"    do
    For "c" in "C"    do
      If ("o" and "c" have the same
        meaning) and (the type of "c" is
        different from Boolean)
      then
        create a direct mapping between "o"
        and "c".
      End
      If ("o" and "c" have the same
        meaning) and (the type of "c" is
        Boolean) then create an indicative
        mapping between "o" and "c".
      End
      If (the value of "o" can change
        according to the value of "c")
      then
        create an indirect mapping between
        "o" and "c".
      End
    End
  End
End
End

```



The connection between tasks represents the sequence of execution with or without the transfer of some information. They represent the information flow between them. Figure 5 presents an example of a task model implemented using the BPMN formalism for a software system that aims to provide medicine recommendation that does not depend of doctor's prescription. We will use in the example, two main user interfaces. The first UI ask the input information from the user: his/her disease area (e.g. head, throat), his/her sex, and some pre-defined symptoms (fever, tiredness, eye pain, cough, dizziness, trembling). The second interface presents the recommended medicines based on the input data and user age. The tasks 3 to 13 represent the first interface with the static information (e.g. task 3 "enter the disease area") and the required input of a UI (e.g. the Disease area input field). Task 15 represents a second UI with the result information.

To provide the content personalization for all input/output information, we should annotate each user or system tasks with:

- The input/output element manipulated in the task, that we named interaction elements. The interaction elements are an abstract view of types of interaction between user and system. The interaction elements that we are particular interested for the content personalization are the different types of input of information (e.g., informed by the user – named UIFieldManual, selected from a defined set of options – named UIFieldOneChoice, etc.) and output information (named UIFieldOutput). Other interaction elements were also defined to be associated with any task modeled with BPMN (e.g., to represent a group of information – named UIUnit, to represent a UI with several groups of information – named UIGroup, etc.). In the figure 5, we associated to the task number 6 an UIFieldManual interaction element since it has to be filled manually by user who has to mention his/her gender. The UIFieldOneChoice element associated to the task number 8 means that is the user has to select or not this alternative. Finally the task number 15 is enriched with the UIFieldOut interaction element since it is a task where the system has as an output the results of searching process.
- The concept of the domain model, whenever possible, and its pertinent mapping with the context model. Direct mappings are chosen when we want that the domain concept provides, at runtime, the content of the related context element. Indicative mappings are chosen when we want to set at runtime the selection/not selection of a domain element from a list of options. This selection is defined by the value (true/false) of the associated context element. Finally, indirect mappings are chosen when we want to show results that depends on many other domain elements to which we associated indirect mappings with context elements.

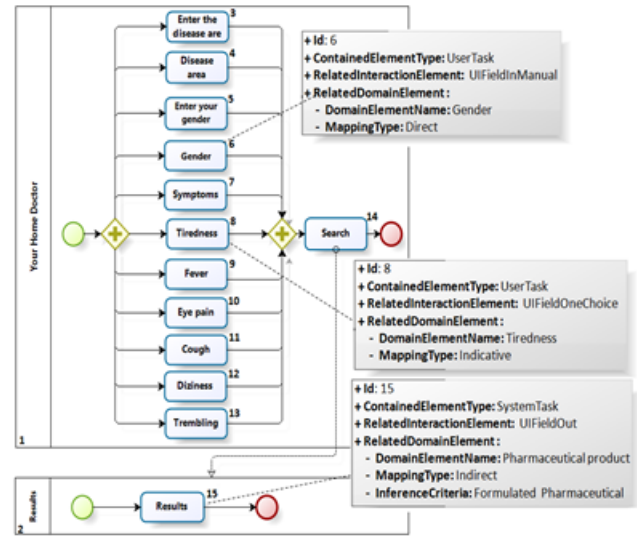


Figure 5. Examples of tasks annotation.

In the Figure 5, we specify for the task (6), the *Gender* ontology attribute as an *OntologyElementName* attribute and we choose the Direct option (for the mapping type attribute). That means that the value of Gender input field corresponds to the value of the context element mapped directly to it, in this case the *Gender* attribute. We associate to the task number (15), the searched concept, in this case *Pharmaceutical Product* (through the *OntologyElementName* attribute) and we specify the parameter that have to be taken into account when searching it, in this case the *Formulated Pharmaceutical* ontology concept (mapped to the *Age* context attribute). Since the designer knows the defined mapping, that means that when delivering the result of searching the pharmaceutical products, the system must take into account the age of the user.

The definition and the annotation of the BPMN were better detailed in [3].

## UIML CODE GENERATION

As mentioned previously, we decide to generate content-personalized UI that are described through UIML. UIML was chosen because it is a UIDL that provides facilities to manipulate content and to work with dynamic information. Indeed, several tools for the conversion from UIML to other source code in different platforms were proposed (for example, the toolkit LiquidApps allows the conversion from UIML to Java, HTML, WML and VoiceXML).

A UIML model is composed of two main components: interface and peers. The interface component represents the description of the interface through four parts: structure, that represents the organization and hierarchies of all UI parts; content that describes the set of the application information that will be displayed (e.g. in

different languages), behavior that represents the behavior of the application at the user interaction time, and style that defines all properties specific for each UI element. The peers component links the generic UI elements and their properties, to a specific platform using the presentation part. Indeed, it describes the calling conventions for methods that are invoked by the UIML code in the logic part. The logic part links methods that are used in UIML with other ones used in a platform-specific source code.

In our case, the PIM level is composed of the structure, behavior, content and style (that manipulates content) parts.

The PSM level is composed of the style, presentation and logic parts. The style part here contains the layout information using the appropriate style properties based on the chosen platform.

In order to describe the behavior of the UI that the user is interacting with, UIML provides the rule statements which are composed of a set of conditions and associated actions. The condition is used to keep the dynamics of the application modeled in the BPM when transforming to UIML.

In order to ensure the sequencing of tasks also, UIML provides the <Event> tag that could be a condition to trigger a rule or that could be the result of an action of a rule. This tag is used to represent the interactive tasks produced by the user or the system. But UIML does not propose tags to define non-interactive tasks.

The solution we proposed was to create the *activation variables*, to ensure the passage of the flow of information between interactive and non-interactive tasks while transforming the annotated BPMN model to UIML. In fact, for each element of the BPMN diagram, we generate at the UIML code, a variable called *activation variable*. Once the variable activation of an element is equal to True, a set of UIML actions is generated depending on the type of that element (see Figure 6(a)).

In order to manipulate the personalized content in UIML, we generate in the behavior part two methods: the `GetValueFromContext` and `Get-Element` methods.

The `GetValueFromContext` method is used to implement the direct and the indicative mappings. For the direct mapping it is used to set the value of some fields that will be filled-in automatically by the value taken from the context by calling the `GetValueFromContext` method. In the example of Figure 5, for the task number 4, the value of the context attribute, that is mapped directly to the *Gender* ontology element, will be extracted by the `GetValueFromContext` and will be used to fill the `UIFieldManual` element. Figure 6(b) presents the part of the UIML code generated for this purpose.

For indicative mappings, the value of the context element (true/false), returned by the `GetValueFromContext` method is verified to decide the value (selected/not selected) of the interaction element. This is done by including a condition statement in the UIML rule that verifies the value of the context element mapped to the ontology element. The generated `when-true` statement, will decide to select or not that UI element. For example, in the Figure 5, in the case of the task 6, if the `GetValueFromContext` returns a true value (the user is tired), then select this alternative.

The `Get-Element` method is used to implement the indirect mapping. In fact, using the UIML `call` statement, the *DomainElementName* annotation attribute presents the searched element, first argument of the method, and the *inferenceCriteria* presents parameters to take into account when searching it, remaining method parameters.

For the example in Figure 5, in the case of the task number 15, the searched element *Pharmaceutical Product* will be the first parameter of this method, and the searching parameters is the *Formulated Pharmaceutical* ontology concept (mapped to the *Age* context attribute). Figure 6(c) presents the part of the generated UIML code related to this method.

For each `call` generated, a <logic> statement will be added with the information about the implemented code for this method. This code is implemented by the software designer to search the required information based on the defined parameters. Fig. 6 shows some parts of the PSM when the target platform will have Java as a programming language.

The structure class named `G:TextField` will be mapped to the `JTextField` Swing library class. The generated method named `16GetValueFromContext` is mapped to the platform-specific method named `Lamih.Context.GetValueFromContext` that allows getting information from context. The `15Get-Element` method sets the method to search an element (`param1_15`) considering a criteria (`param2_15`).

## CONCLUSION

This paper proposed a set of designing models that permit generating UIML code describing content-personalized UIs. These models belong to an MDA architecture that considers the content personalization since the UI design. The core of this approach is the personalization models composed of the context that is mapped onto a domain ontology. These models are used to annotate the UI task model. By transforming these models, we are able to generate a UIML code.

We are currently working in the development of an integrated environment to support all features of this proposal, from the definition of the UI designing models till the generation of the final UI.



**Figure 6. Example of UIML code generated by transformation from BPMN to PIM.**



**Figure 7. Example of UIML code integrated to PSM Presentation part (a) and Logic part (b)).**

## REFERENCES

1. Ali, M.F. A transformation-based approach to building multi-platform user interfaces using a task model and the user interface markup language. PhD thesis, Blacksburg, Virginia, USA, (2004).
2. Bacha, F., Oliveira, K., and Abed, M. Using Context Modeling and Domain Ontology in the Design of Personalized User Interface. *International Journal on Computer Science and Information Systems IJCSIS*, (2011).
3. Bacha, F., Oliveira, K., and Abed, M. Providing Personalized Information in Transport Systems: A Model Driven Architecture Approach. In *Proc. of IEEE International Conference on Mobility, Security and Logistics in Transport MSLT'2011* (June 2011). IEEE Computer Society Press, Los Alamitos (2011), pp. 452-459.
4. Batchel O. *et al.*: Translational Medicine Ontology. Available at <http://translationalmedicineontology.googlecode.com/svn/trunk/ontology/tmo.owl>
5. Bouchelliga, W., Mahfoudi, A., Benammar, L., Rebai, S., and Abed, M. An MDE Approach for User Interface Adaptation to the Context of Use. In *Proc. of 3rd International Conference on Human-Centred Software Engineering HCSE'2010*. R. Bernhaupt et al. (Eds.). LNCS, vol. 6409. Springer, Berlin (2010), pp. 62-78.
6. Bouzghoub, M. Action Spécifique sur la Personnalisation de l'Information. CNRS, Paris (2004).
7. Brossard, A., Abed, M., and Kolski, C. Modélisation conceptuelle des IHM : Une approche globale s'appuyant sur les processus métier. *Ingénierie des Systèmes d'Information (ISI) - Networking and Information Systems 12*, (2007), 69-108.
8. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J. A Unifying Reference Framework for Multi-Target User Interfaces. *Interacting with Computers 15*, 3 (2003), pp. 289-308.
9. Chen, H., Perich, F., Finin, T., and Jochi, A. SOUPA: Standard ontology for ubiquitous and pervasive applications. IEEE Computer Society, Los Alamitos (2004), pp. 258-267.
10. Clerckx, T., Luyten, K., and Coninx, K. Dynamo-aid: A design process and a runtime architecture for dynamic model-based user interface development. In *Proc. of EHCI/DSV-IS 2004*, Lecture Notes in Computer Science, 3425, Springer, (2004), 77-95.
11. Dey, A. Understanding and Using Context. *Journal of Personal and Ubiquitous Computing 5*, (2001), pp. 4-7.
12. Helms, J., Schaefer, R., Luyten, K., Vermeulen, J., Abrams, M., Coyette, A., and Vanderdonckt, J. Human-Centered Engineering with the User Interface Markup Language. In *Human-Centered Software Engineering*, A. Seffah, J. Vanderdonckt, M. Desmarais (Eds.). Chapter 7, HCI Series, Springer, London (2009), pp. 141-173.
13. Kim, E., and Choi, J. An Ontology-Based Context Model in a Smart Home. *Computational Science and Its Applications*, (2006), pp. 11-20.
14. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., and Lopez, V. UsiXML: a Language Supporting Multi-Path Development of User Interfaces. In *Proc. of 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems EHCI-DSVIS'2004*. LNCS, vol. 3425. Springer-Verlag, Berlin (2005), pp. 200-220.
15. OMG, MDA Guide, Version 1.0.1, Available at <http://www.omg.org/cgi-bin/doc?omg/03-06-01> (2003)
16. OMG, Business process modeling notation specification, (2006).
17. Paternò, F. *Model-based design and evaluation of interactive applications*. Springer, Berlin (1999).
18. Vanderdonckt, J. Model-Driven Engineering of User Interfaces: Promises, Successes, and Failures. In *Proc. of 5th Annual Romanian Conf. on Human-Computer Interaction ROCHI'2008* (Iasi, September 18-19, 2008), S. Buraga, I. Juvina (Eds.). Matrix ROM, Bucarest (2008), pp. 1-10.
19. Winckler, M., Trindade, F.M., Stanculescu, A., Vanderdonckt, J., Cascading Dialog Modeling with UsiXML. In *Proc. of 15th Int. Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'2008* (Kingston, July 16-18, 2008). Lecture Notes in Computer Sciences, vol. 5136. Springer, Berlin (2008), pp. 121-135.

# Architecture for Reverse Engineering of Graphical User Interfaces of Legacy Systems

Óscar Sánchez Ramón, Jesús Sánchez Cuadrado, Jesús García Molina

Department of Informatics and Systems, University of Murcia

Campus de Espinardo, Murcia (30100).

+34 868 884642 - {osanchez, jesus, jmolina}@um.es

## ABSTRACT

The CAMELEON reference framework is aimed at specifying multi-modal user interfaces at several abstraction levels. This is a proposal that is growing in popularity and there are some User Interface Description Languages (UIDL) that are aligned to this framework. In this paper, we propose a model architecture to perform reverse engineering of legacy systems using UIDLs that conform to the CAMELEON framework. The general approach is also illustrated in the specific context of a reverse engineering tool we are developing to modernise applications created with Rapid Application Development (RAD) environments.

## Author Keywords

User Interface Description Language (UIDL), Cameleon Reference Framework, Reverse Engineering, Legacy Systems.

## General Terms

Design, Reliability, Human Factors, Theory.

## Categories and Subject Descriptors

D.2.2 [Software Engineering]: Design Tools and Techniques – *User interfaces*. H.5.2 [Information Interfaces and Presentation]: User Interfaces – *User-centered design*.

## INTRODUCTION

The birth of the internet and mobile technologies has led to an explosion of platforms that can run interactive applications. Thus, there has been a need for developers to build User Interfaces (UIs) that can run across different platforms and devices. However, different platforms have different features and constraints that hinder that a UI programmed for a particular platform could be run in a different one. At the same time, legacy systems are still in use in many companies, but they cannot take advantage of recent UI developments.

UIDLs have recently gained in popularity since they are useful for reducing the effort of creating UIs for different platforms. When using a UIDL a developer only has to focus on defining interfaces and leaves the implementation details to the tool. Although this is probably the main characteristic of UIDLs, they are very heterogeneous in terms of coverage, aims, goals, or software support. In addition,

some UIDLs were not originally conceived to be used in reverse engineering but for easing UI design.

Model-based approaches have been widely used and have proved to be useful in varied contexts due to its ability to automate repetitive tasks [1]. Model-based and automatic techniques allow designers to focus on the definition of interfaces at a high level [2], which is especially interesting when there is a wide range of UI target devices and platforms available. In this sense, the CAMELEON Reference Framework (CRF) [3] is a conceptual framework aimed at helping structuring the development process of multi-target user interfaces.

At present we are building a model-based reverse engineering tool for legacy systems developed with Rapid Application Development (RAD) environments [4]. In this kind of applications, GUI code is typically mixed with business logic, so recovering the different aspects of the GUI from the source artefacts is essential. This task requires defining models at different levels of abstraction, and building model transformations to extract information from lower-level models. In this way, the Horseshoe Model [5] provides a conceptual framework for dealing with different levels of abstraction in a reverse engineering process.

In this paper we discuss a model-based infrastructure to enable obtaining the models proposed in the CRF [3] from a legacy application. In this way, we will align the CRF to the Horseshoe Model in order to provide a model-based framework for reverse engineering GUIs.

We will discuss which auxiliary models are needed, and how model transformations relate the different models involved. As a concrete example of the application of the framework, we will apply it to the reverse engineering of applications developed with RAD environments.

In the next section we will outline the integration of the CAMELEON framework with the Horseshoe Model. Then we will present the model architecture we propose to perform reverse engineering based on the CAMELEON framework, and we will show the infrastructure we have devised to reverse engineering the RAD legacy scenario. We will finish with the conclusions and future work.

## REENGINEERING ARCHITECTURE

The CAMELEON Reference Framework is an approach that defines several levels of abstractions for specifying user interfaces. This framework can be aligned with the Horseshoe Model [5], which is a conceptual model for software reengineering. In Figure 1, we show the Horseshoe model tailored for UI reengineering.

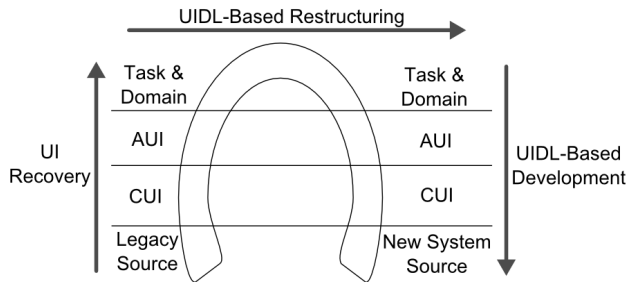


Figure 1. Horseshoe model for UI reengineering.

The approach consists of three main phases, which are depicted with arrows. The first phase is the reverse engineering of the legacy system that starts with the analysis of the legacy artefacts. This step can entail dealing with different kinds of artefacts. For instance, the UI can be defined in a programming language, XML or a proprietary format (text or binary), whereas event handlers are commonly expressed in some imperative language. From the legacy artefacts, a Concrete User Interface Model is obtained, which defines the UI in a platform-independent way. An Abstract User Interface Model is obtained by raising the abstraction level of the CUI so it is modality-independent.

The CRF [3] goes beyond and proposes the extraction of the Task and Domain Models that capture the intention of the original UI design. Note that the Task and Domain Models can be considered as Computing-independent Models (CIM) of the MDA approach [6]. As a result of the reverse engineering (recovery) phase, we will obtain information of different aspects that were not explicit in the original artefacts. This information can be expressed in the form of models of some UIDL. However, as we will explain, for being effective for this task UIDLs require some features which they typically not provide.

The second phase involves performing structural and or behavioural changes in the original system in order to move it to a different platform or technology. The restructuring is applied on the UIDL that captures the source system.

Finally, the Horseshoe Model advocates for a forward engineering approach where the generation of the low level artefacts is guided by the higher-level models. In the case of the CRF, the Task Model, the Domain Model, and the AUI can be used to generate (semi-automatically) the CUI and the final UI.

Although not all the UIDLs conform to the CRF, we will build our reverse engineering tool upon this framework since it is suitable for reengineering as it is based on similar principles as the Horseshoe Model.

## MODEL-BASED REVERSE ENGINEERING APPROACH

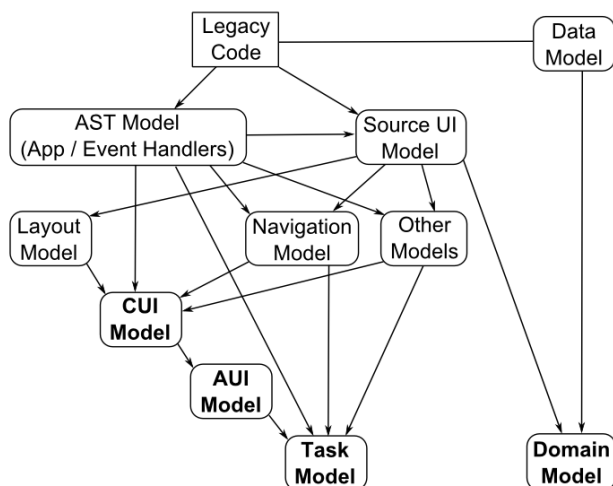
In this section we will detail the reverse engineering phase (UI recovery) of the architecture shown above, discussing which additional models are need to generate the models proposed by the CRF for representing UI at different abstraction levels. The type of legacy UIs that we want to encompass with our approach are:

- **Textual User Interface (TUI).** They are also known as Character-based User Interfaces (CUIs), but we prefer the former in order to avoid misunderstanding. TUIs only use text, symbols and colors available on a typical text terminal, while Graphical User Interfaces (GUIs) typically use high-resolution raster graphics.
- **Rapid Application Environment (RAD).** Oracle Forms and Borland Delphi are two examples of these kinds of environments. They became popular in the 90's by allowing developers to create management applications quickly by means of visual facilities. They reduce the development time by facilitating GUI design and coupling data access to GUI controls.
- **Primitive web applications.** These are web 1.0 applications that were developed using HTML for the presentation layer, and events were handled in the server side (Javascript was not widely used yet). CSS was not widely used to layout web controls but this was frequently done with tables. They are featured by a poor user experience since neither Ajax nor embedded objects (like Adobe Flash players) were used.

The architecture we have devised is shown in Figure 2. Arrows represent model-to-model transformations, but the arrows that are originated in the Legacy Code, which are injectors. Note that arrows also state the order in which the process must be applied. An *injector* bridges two technical spaces [7], that is, it transforms artefacts in a specific formalism into a different one. For instance, code conforming to a grammar (grammarware) injected to some modeling framework (modelware) such as EMF/Ecore. *Model-to-model* transformations allow us to automate the conversion of models at different levels of abstraction.

Cross-references from a model to the models that originated it have been omitted for clarity. These *cross-references* between models allow us to keep relationships between the different recovered aspects, as well as maintaining the traceability from the source artefacts. *Legacy code* represents the collection of artefacts that makes up the system (programming language files, configuration files, UI files, and so forth).





**Figure 2. Reverse engineering approach proposed.**

In general obtaining a high-level representation of a system requires going through several abstraction levels and transformation steps, and in the case of the UI the CAMELEON Reference conceptually provides such levels. However, in our experience dealing with reverse engineering we have found out that a simple transformation chain is not enough, but support models are needed to gather and reorganize the information, so that the transformation chain is complemented with these models. For example, in Figure 2, a Navigation model contributes the transformation to obtain the CUI and the Task Model.

#### Low-level support models

Next we explain some of the support models we have identified so far, and later we discuss how they are used to obtain models from the CRF.

##### AST Model

An *Abstract Syntax Tree* (AST) model conforms to a metamodel which represents the abstract syntax of the underlying language. In this paper we are using the term AST due to it is a well-known concept to refer to a structure that captures the abstract syntax of a language, but actually this model is not a tree but a graph. If the definition of the interface is mixed with the event handling (and in some cases also mixed with business logic or persistence) in programming language code, the AST model will represent all the code. This is the case when reverse engineering TUIs. Conversely, if the definition of the interface and the rest of the code are placed in separate modules (or at least in separated sections of the same module), the AST should just represent the code of the event handlers.

AST models are obtained in different manners, depending on the nature of the source files. If the source code can be expressed with a grammar, then Gra2MoL [8] can be used to extract the models. When the source code is XML, there exist facilities to get models, such as the utilities of the Eclipse Modelling Framework (EMF) [9]. Sometimes the source code is stored in a binary proprietary format, so in this case company utilities are needed to convert the code

to a well-known format such as XML. In other cases, a dedicated parser could be used.

AST models are dependent on the language in which the code is expressed and it is possible to abstract these models to obtain language-independent models. This has the advantage that later reverse engineering algorithms can be re-used for different languages. In this sense, Knowledge Discovery Metamodel (KDM) [10] could be used to represent language-independent code.

##### Source UI Model

This is a model which mirrors the user interface of the source system. A metamodel to represent such a system must be built. When the user interface is stored in a separate definition (e.g., an XML file), the model is generated directly from the definition. When the user interface definition is mixed with other code, as in TUIs, this model is obtained from the AST model.

##### Layout Model

Legacy systems commonly have an implicit layout, i.e., the widgets are located in the window by means of absolute or relative coordinates. Expressing the layout in that way is not a good practice. For example, when the layout is expressed with coordinates, if the size of the window is changed its content will not be resized accordingly. This is the case of some RAD applications and some hard-coded interfaces. Explicit extraction of the implicit layout is not a trivial task. This problem is addressed in [11]. When using a markup language for defining the user interface, e.g. HTML, the parts of the window are defined with marks. In many cases, tables were used to layout old web pages, and this is a discouraged practice by the W3C, so extracting an explicit high-level layout from web pages is also interesting. Due to parts of the window are limited with marks, the layout discovery is easier than the previous case. Some works like [12,13] deal with the layout extraction from web pages.

##### Navigation Model

This model represents the navigation relationships between the windows of the applications. They are frequently represented by means of *Finite State Machines* (FSMs). Navigation Models are useful, for example when the original desktop application needs to be restructured to fit the Ajax platform. In this case the navigation flows identified could be turned into interactions among parts of the same window. Some works [14] deal with the extraction of navigation flow from code models (AST, KDM, and others). In [15] authors perform static analysis on a text-based UI and obtain a transition graph which is represented with AUIDL.

##### Data Model

A variety of data models are used in legacy systems and for each kind a metamodel must be built. For instance, a metamodel for the SQL DDL could be used to represent a logical data model. It could be refined to obtain an entity-relationship model.

### *Other Models*

Some other information can be discovered from the UI definition model and the AST model. For example, a model that represents the validators that can be extracted for certain widgets. Another example could be a model representing dependencies among widgets. All this information will ease later restructuring and forward engineering phases.

### **CAMELEON Models**

Given that *support models* have been obtained CUI, AUI and Task models are derived by means of model transformations.

#### *CUI Model*

The Concrete User Interface (CUI) Model will contain the information extracted from the different aspects of the source interface (e.g. layout, event handling or navigation) in a platform-independent way. This is not a complex task since this model is constructed by querying and information gathered in the previously obtained models. As an example, in [16] authors obtain a CUI from web pages.

#### *AUI Model*

It is a modality-independent representation of the CUI. This is a relatively straightforward transformation since is based on abstracting the concepts that appear in the CUI model. In [17] some examples to obtain AUI elements from CUI are explained for the UsiXML language.

#### *Task Model*

Obtaining high-level tasks from a source system is not a trivial issue. Task models cannot be derived just with the information contained in the AUI model, but additional information is required. There are works like [18] in which static program analysis techniques such as control flow analysis, data flow analysis, or pattern matching are performed in order to extract the user interactions. Also dynamic analysis is performed to get the user interactions, such as in [19], which propose analysing the user traces to get the tasks.

This problem is tightly-related with the problem of obtaining business process models (BPMN) from low-level artefacts. In [20], the authors address the problem of obtaining BPMN models from KDM models based on pattern recognition. We believe that the construction of the Task Model can be facilitated by some of the models that we have previously obtained. For example, the Navigation Model captures the windows that can be reached from a certain window. If we assume that different windows are used to achieve different goals, we can use heuristics over the Navigation Model to know which tasks are derived from other tasks. Other additional models can also be useful. For example, if we have an Interaction Model that captures the dependencies among widgets, it is possible to know which widgets are affected by others. Then we could think of a heuristic that groups those widgets that are related and associate them with the same task.

### **Domain Model**

In many cases is represented with a UML class model although the source system does not need to be object-oriented. Most legacy systems store application data in a relational database. If the database schema is available, Domain Model can be easily obtained. In this information is not available, we can analyse the Source UI Model and or the AST to see what information is displayed in the screen. Even if the database schema can be used, this can include much more information than the information involved in the application, so using the UI Model could enhance the Domain Model. The information contained in the Layout Model can be useful as a hint to know about close fields in the window, which are likely to be conceptually-related.

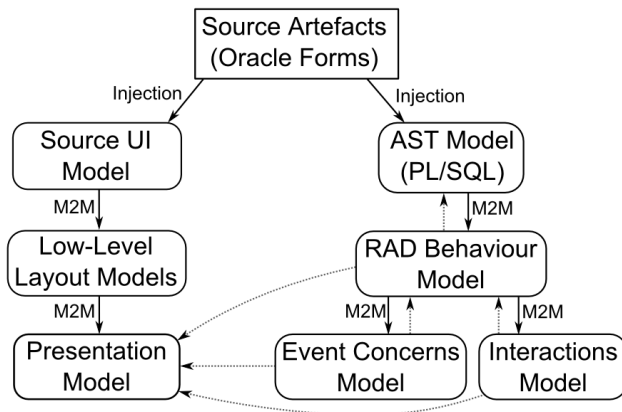
### **APPLICATION SCENARIO: RAD APPLICATIONS**

As explained in the introduction, we are developing a reverse engineering tool in the context of RAD applications, where GUIs are an essential element.

RAD-based applications have some specific constraints that affect the UI recovery. We particularly highlight two features. First, the layout is not explicitly defined by using high-level layouts (e.g., Flow Layout in Java Swing), but it is implicitly defined by the position of the elements, which are expressed in terms of relative or absolute coordinates. Keeping this type of layout is not a good practice, because this means that, for example, when a window is resized the widgets are not resized or rearranged accordingly. The second feature is that code managing the GUI is mixed with business logic and database access, i.e., there is no clear separation among the different concerns of the application. This entails that extracting information of a certain aspect requires analysing all the code.

In Figure 3 we show the model architecture we have implemented. Solid lines are injectors and model-to-model (M2M) transformations. Dotted lines are cross-references among models.

At present we have just reach the CUI level, which is composed of: The Presentation Model, the RAD Behaviour Model, the Event Concerns Model and the Interactions Model. The architecture is exemplified with Oracle Forms as the source technology, but it could be used with any other RAD technology such as Borland Delphi or Microsoft Visual Basic. The different models involved are described next, exemplified for Oracle Forms. All the model transformations have been implemented in RubyTL [21].



**Figure 3. Architecture proposed for reverse engineering RAD-based applications.**

### Source Artefacts

The source artefacts in the case of Oracle Forms are XML files that include the definition of the different windows of the application. These files also include some other information, e.g. information about database tables linked to widgets. The presentation elements such as widgets or windows can have some event handlers associated with them, which are expressed in the form of PL/SQL triggers. The PL/SQL code is embedded in the XML files.

From the XML files, two models are obtained. The *Source UI Model* is obtained by means of an EMF tool for injecting models from XML code, and the *AST model* (PL/SQL models in our case) which is obtained with Gra2MoL, which allow us to get models from code that conforms to a grammar.

### Source UI Model

It is a model that represents the source GUI. Although it is not explicitly depicted in Figure 3, two different models are involved. Firstly, a model that mirrors the original Forms GUI is obtained. From this model, a model-to-model transformation gets a second model, called *RAD Model*, which represents the source system in terms of common concepts provided by RAD environments. This is a kind of normalization model for GUIs built with a RAD, in order to make the rest of the process independent of the specific RAD environment.

### Low-Level Layout Models

As said, in desktop legacy applications (such as applications created with RAD) the layout is implicitly defined by the position of the elements, which are expressed in terms of coordinates. This means that, for example, when a window is resized the widgets are not resized or rearranged accordingly. We have defined a couple of low-level layout models to support the discovery of an explicit layout model. Although they are auxiliary models, they are useful to bridge the gap between the original layout based on coordinates and the CUI model. More details about these models and the algorithms involved are given in [11].

### Presentation Model

This model defines the structure of the GUI (i.e., what views compose the GUI, and which are the containment relationships among the graphical elements of the views), as well as the layout (i.e., how the graphical elements are arranged in the views). An example that shows the Source UI Models, the Low-Level Layout Models and the final Presentation Model can be found in [11].

### RAD Behaviour Model

This model captures the behaviour of the source code in terms of simple primitives which are common in RAD environments, such as read data from a database or write some data in the GUI controls. This model is the basis on which we extract more knowledge about the source code since it eases static analysis of a RAD system. Particularly, it let us get the Event Concerns Model and the Interactions Model [22].

To adapt this architecture to a different legacy technology, this model should be replaced with a domain-specific behaviour model if it is possible to identify commonalities in the event handler code of a domain. Another option would be to suppress this model and perform all the static analysis based on the AST Model.

### Event Concerns Model

Legacy systems often mix UI code with business logic or control code in event handlers. This support model is aimed at identify fragments of code which are related to the same concern so that it is possible to achieve separation of concerns. It helps us to the understand event handlers, and it can be useful when restructuring the source system in order to create a new one with better quality in terms of extension and maintainability. In [22] a real example of event handlers that are reverse engineered can be found, which shows the RAD Behaviour and Event Concerns Models obtained.

### Interaction Model

The goal of this model is twofold. On the one hand it represents the navigation flow by means of a state machine. On the other hand, it explicitly shows the dependencies that exists among the GUI elements. For example, provided there is a checkbox that enables a panel when it is checked, this model will capture the interaction that exists between the checkbox and the panel.

### Putting the pieces together: CUI Model

In summary, our CUI model represents GUIs developed with RAD environments, and it is specifically tailored for the reverse engineering of these kinds of applications. The models that compose the CUI level are:

- The *Presentation Model* which defines the elements in the GUI and their layout.
- The *RAD Behaviour Model* which defines the behavioural part of the GUI (the event model).

- The *Interactions Model* which defines navigation relationships and dependencies among widgets.
- The *Event Concerns Model* that enriches the *RAD Behaviour Model* by specifying categories of code in the event handlers.

### CONCLUSION AND FUTURE WORK

In this paper we have presented a model-based architecture to perform reverse engineering of legacy UIs based on the Cameleon Reference Framework [3]. We have shown the concrete model architecture we have built to reverse engineering RAD-based applications. As a future work, we will continue with the development of the reverse engineering tool in order to reach all the abstraction levels defined in the Cameleon Reference Framework. We will also generate UsiXML models from the information gathered. In addition, the architecture will be extended to consider TUI and legacy web applications.

### ACKNOWLEDGMENTS

This work is partially supported by Consejería de Universidades, Empresa e Investigación (grant 129/2009) and Fundación Séneca (project 15389/PI/10).

### REFERENCES

1. Selic, B. Personal reflections on automation, programming culture, and model-based software engineering. *Automated Software Engineering*, 15, 3-4 (December 2008), 379-391.
2. Myers, B., Hudson, S. and Pausch, R. Past, present, future of user interface tools. *ACM Trans. Comput.-Hum. Interact.* 7, 1 (March 2000), 3-28.
3. Calvary, G., Coutaz, J., Thevening, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J. A Unifying Reference Framework for Multi-Target User Interfaces. *Interacting with Comp.* 15, 3 (June 2003), 289-308.
4. James, M., *Rapid Application Development*. Macmillan Publishing Co., Inc. 1991.
5. Kazman, R., Woods, S. G. and Carrière S. J., Requirements for Integrating Software Architecture and Reengineering Models: CORUM II. *Proceedings of the WCRE'98*, 154-163.
6. OMG Model Driven Architecture. <http://www.omg.org/mda/>.
7. Bézin, J. and Kurtev, I. Model-based Technology Integration with the Technical Space Concept. *Proceedings of the Metainformatics Symposium*. 2005.
8. Cánovas Izquierdo, J. L. and García Molina, J. A Domain Specific Language for Extracting Models in Software Modernization. *Proceedings of the ECMDA-FA'09*, 82-97.
9. Eclipse Modeling Framework Project (EMF). <http://www.eclipse.org/modeling/emf>.
10. OMG Knowledge Discovery Meta-Model (KDM) v1.0. 2008. <http://www.omg.org/spec/KDM/1.0/>.
11. Sánchez Ramón, O., Sánchez Cuadrado, J. and García Molina, J. Model-Driven Reverse Engineering of Legacy Graphical User Interfaces. *Proceedings of the ASE'10*. 147-150.
12. Bandelloni, R., Mori, G. and Paternò, F. Dynamic Generation of Web Migratory Interfaces. *Proceedings of the MobileHCI'05*. 83-90.
13. Cai, D., Yu, S., Wen, J. and Ma, W. VIPS: a Vision-Based Page Segmentation Algorithm. *Microsoft Research TechReport*. 2003. <http://research.microsoft.com/apps/pubs/default.aspx?id=70027>.
14. Staiger, S. Reverse Engineering of Graphical User Interfaces Using Static Analyses. In *Proceedings of the WCRE'07*.
15. Merlo, E., Gagné, P., Girard, J., Kontogiannis, K., Hendren, L., Panangaden, P., De Mori, R. Reengineering User Interfaces. *IEEE Software*, 12, 1 (1995), 64-73.
16. Bouillon, L., and Vanderdonckt, J., Retargeting Web Pages to other Computing Platforms with VAQUITA. In *Proc. of IEEE Working Conf. on Reverse Engineering WCRE'2002* (Richmond, 28 October-1 November 2002). A. van Deursen, E. Burd (Eds.). IEEE Computer Society Press, Los Alamitos (2002), pp. 339-348.
17. Limbourg, Q., and Vanderdonckt, J. Multipath Transformational Development of User Interfaces with Graph Transformations. *Human-Centered Software Engineering: Software Architectures and Model-Driven Integration, Chapter 8, Vol. II*, Springer HCI Series, Springer-Verlag. 2007.
18. Paganelli, L. And Paternò, F. Automatic Reconstruction of the Underlying Interaction Design of Web Applications. In *Proc. of the SEKE'02*. 439-445.
19. El-Ramly, M., Iglinski, P., Stroulia, E., Sorenson, P and Matichuk, B. Modeling the System-User Dialog Using Interaction Traces. *Proceedings of the WCRE'01*.
20. Zou, Y., Lau, T., Kontogiannis, K., Tong, T. and McKegney, R. Model-Driven Business Process Recovery. *Proceedings of the WCRE'04*.
21. Sánchez Cuadrado, J. and García Molina, J. Modularization of model transformations through a phasing mechanism. *Software and System Modeling*, 8, 3 (July 2009), 325-345.
22. Sánchez Ramón, O., Sánchez Cuadrado, J. and García Molina, J. Reverse Engineering of Event Handlers of RAD-Based Applications. *Proceedings of the WCRE'11*.
23. Bouillon, L., Limbourg, Q., Vanderdonckt, J., and Michotte, B., Reverse Engineering of Web Pages based on Derivations and Transformations. In *Proc. of 3rd Latin American Web Congress LA-Web'2005* (Buenos Aires, October 31-November 2, 2005), IEEE Computer Society Press, Los Alamitos, 2005, pp. 3-13.

# Model-based Reverse Engineering of Legacy Applications User Interfaces

**Francisco Montero, Víctor López-Jaquero, Pascual González**  
LoUISE Research Group, I3A, University of Castilla-La Mancha  
Campus, s/n, 02071 (SPAIN)  
+34 967 59 92 00 - {fmontero, victor, pgonzalez}@dsi.uclm.es

## ABSTRACT

User Interface Description Languages (UIDL) are languages used in Human-Computer Interaction (HCI) in order to describe User Interfaces (UI) independently of any implementation technology. These languages can be used effectively and efficiently when proper software is available. In this paper a suite of software tools is introduced. This suite of tools is useful for supporting Reverse Engineering activities of UIs. Prototyping, evaluation and specification of user interfaces are enabled by using our suite of tools: GuiLayout++, PureXML, reTaskXML, and AcauiXML.

## Author Keywords

Model-based, user interfaces, legacy applications, reverse engineering, UIDL.

## General Terms

Measurement, Design, Human Factors, Languages.

## Categories and Subject Descriptors

D.2.2 [Software Engineering]: Design Tools and Techniques – *user interfaces*. H.5.2 [Information Interfaces and Presentation]: User Interfaces, Graphical User Interfaces (GUI).

## INTRODUCTION

Engineering is the process of designing, manufacturing, assembling and maintaining products and systems. There are two types of engineering, forward engineering and reverse engineering. Forward engineering is the traditional process of moving from high-level abstractions and logical designs to the final implementation of a system. The process of duplicating an existing part, subassembly, or product, without drawings, documentation, or a computer model is known as reverse engineering.

Reverse engineering is now widely used in numerous applications, such as manufacturing, industrial design and jewelry design and reproduction. For example, when a new car is launched on the market, competing manufacturers may buy one and disassemble it to learn how it was built and how it works. In software engineering, good source code is often a variation of other good source code.

In this last scenario, in software development, designers give shape to their ideas by using high and low prototypes by using paper and pencil, interactive paper prototypes,

programmed façades, and prototype-oriented languages, but a set of models are needed to develop a final application. The fidelity of the prototypes previously described ranges from low to high. Does fidelity affect a prototype's usefulness as a testing tool? Most usability experts agree that low-fidelity prototypes are very useful in the early stages of design.

Reverse engineering provides a solution to this problem, because the final user interface model may be the source of information for other models, for instance, concrete user interface, abstract user interface or another. All those models are considered in traditional model-based and model-driven user interface development environments.

In this paper reverse engineering, prototyping and model-based user interface development techniques are used jointly in order to provide facilities for user interface development of legacy application. Following are some of the reasons to use reverse engineering in user interface development:

- The original source code and developers no longer exists.
- The original developers of an application no longer maintain the product, and the software may become obsolete.
- The software application design documentation may be lost or never existed.
- We want to eliminate or modify some bad features of an application.
- Exploring new alternatives to improve software performance and features.
- Documentation

This paper is organized as follows. Section 2 describes and provides an overview of related works with Reverse Engineering. Section 3 introduces the main contribution of this paper, a software tool for supporting reverse engineering by using prototyping and user interface description languages. With this tool presentation and navigation models associated with a model-based user interface development environment can be achieved. Finally, a section related to discussion and conclusions is included.

## RELATED WORK

Reverse Engineering (RE) has been an active research topic for some years now. RE has been applied to many different fields. Regarding Computer Science, it has been applied to both hardware and software systems. First, to focus our work, the term RE will be discussed. The most widely-known taxonomy for RE is [2]. In this work the authors present a taxonomy that illustrates the different paths that software development can follow. Reverse engineering is the process of analyzing a subject system to: (1) Identify the system's components and the interrelationships and (2) Create representations of the system in another form or at a higher level of abstraction. Thus, RE is supposed to happen from implementation to design and from design to requirements. Furthermore, RE can also happen at the same development stage. In this case, some different operations are identified: *Restructuring*, *Reengineering*, *Redocumentation* and *Design Recovery*. *Restructuring* is the transformation from one representation form to another at the same relative abstraction level, while preserving the subject system's external behavior (functionality and semantics).

It does not involve modifications because of new requirements. On the other hand, *Redocumentation* is the creation or the revision of a semantically equivalent representation within the same relative level of abstraction. Such as the pretty printers for code. *Reengineering* is the examination and alteration of the subject system to reconstitute it in a new form and the subsequent implementation of the new form. It generally includes some form of reverse engineering followed by some form of forward engineering or restructuring. Finally, in *Design Recovery* domain knowledge, external information, and deduction or fuzzy reasoning are used to identify meaningful higher level abstractions beyond those obtained directly by examining the system itself. In [4] a multi-path development process for user interfaces was introduced. In this process, user interface design could begin at different entry points in the process, not necessarily starting from the first or last stages. RE was also considered in this approach.

When working with legacy systems user interfaces all three RE operations are very useful. *i.e.* *Restructuring* could be used to port the user interface from one target platform to another. This issue was addressed in VAQUITA [1]. In VAQUITA HTML contents were reverse engineered to be ported to another target platform. The main limitation of VAQUITA was its dependency on HTML. An on-line version of VAQUITA, namely ReversiXML, is also available [10]. WebRevEng [8] also supports HTML Reverse Engineering. Nevertheless, WebRevEng obtains the corresponding CTT task model. User interface *Reengineering* goes one step beyond, as it considers making changes in the functionality. An example would be using one of the tools aforementioned supporting *restructuring*, *i.e.*, VAQUITA, and then add some extra functionalities before porting to another target platform.

Obviously, to support actual RE, more than the user interface should be reversed engineered. This includes functional core and databases. *i.e.*, it has been applied to migrate legacy databases [7].

Model driven techniques applied to user interface design look promising for RE, because of the clear separation of concerns that it exhibits. When applying RE the different facets of an application must be processed. Having a clear separation of concerns can help in this process. Furthermore, in the goals of these model-driven approaches for user interface design *restructuring* and *reengineering* are two purposes usually considered. Starting from a set of models, the aim is generating the user interface for different target platforms, supporting *restructuring* to accommodate the user interface to the peculiarities of each platform (an in general each context of use). The original set of models could be modified afterwards, for instance, to remove obsolete functionalities.

The creation of the models required in any model-driven approach is an interesting issue. A visual syntax can greatly improve the usability of a notation. Directly creating the models, *i.e.* by using an XML syntax, can be error prone and slow. Different tools have been used so far for visually designing these models, including *IdealXML* [5], *SketchiXML* [3] or CTTE (<http://giove.isti.cnr.it/ctte.html>). *IdealXML* support the automatic generation of abstract user interfaces out of task models, enabling the user the rapidly create user interface prototypes. This idea was combined with some automatic evaluation techniques in *GuiLayout++* [6]. In this tool, some RE support was already included. The designer loads a screenshot of the user interface, and then he draws on it the different widgets and areas identified in the screenshot. Finally, these elements were used to automatically generate an abstract user interface.

In this work we use an approach similar to the one we used on *GuiLayout++*, but in this case we aim at reverse engineering the user interface to generate the concrete user interface, including both the widgets, layouts and navigation.

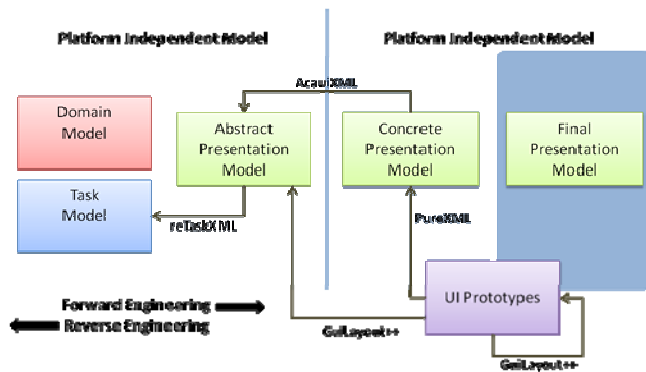
## REVERSE ENGINEERING USER INTERFACE FOR LEGACY APPLICATION

Aiming at supporting RE for user interface design a suite of tools has been developed. These tools support the reverse transformation between the usual models used in user interface development.

Currently, these tools work individually, and they interchange the models they used by means of an XML specification expressed in UsiXML. Nevertheless, we are working in a unified tool to make easier their use. In Fig. 1 the tools developed and the models they manipulate are illustrated.

Fig. 1 shows the main models linked to user interface design. Reading the figure from left to right a forward development of user interfaces can be observed (forward engineering). On the other hand, reading the figure from right to left reverse engineering can be observed.

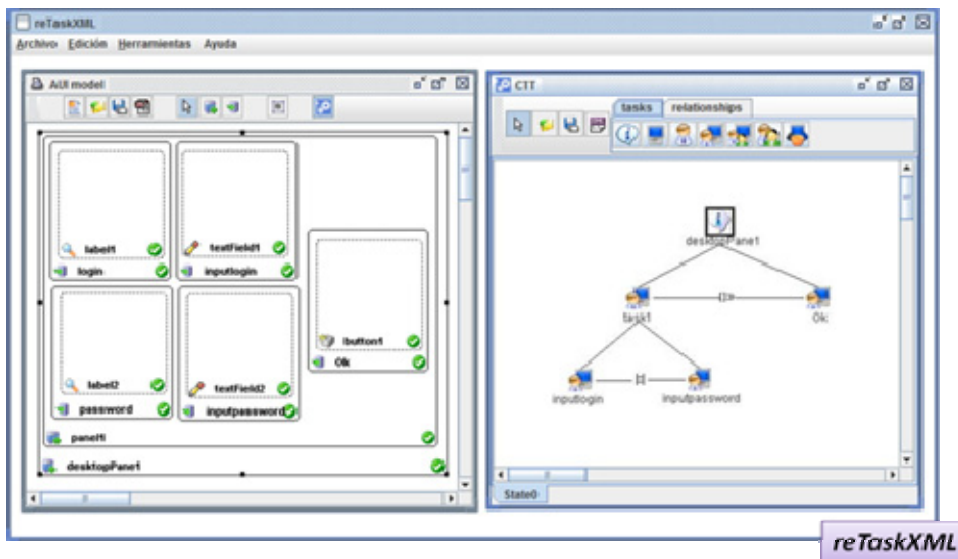




**Figure 1. Suite of tools and reverse engineering of UI process.**

The suite of tools currently developed is:

- *reTaskXML*: is tool that tackles reverse engineering an abstract specification of a user interface to obtain the corresponding task model (see Fig. 2).
- *AcauiXML*: is a tool that supports reverse engineering a concrete user interface model to obtain a corresponding abstract user interface. UsiXML syntax is used for this purpose (see Fig. 3).
- *PureXML*: is a tool that supports loading user interface screenshots, from a user interface design or an existing application, and creating prototypes for those screenshots. These prototypes are then used to obtain a concrete user interface expressed in terms of UsiXML (see Fig. 5).
- *GuiLayout++* [6]: is a tool that supports the creation and evaluation of user interface prototypes (both low and high fidelity). The prototypes created are then used to generate the corresponding abstract user interface specification in UsiXML language (see Fig. 4).



**Figure 2. reTaskXML: from abstract user interface to task model.**

### The reverse engineering process

As aforementioned, different situations can arise promoting the use of reverse engineering techniques. Legacy applications are one of the most usual situations when one would use reverse engineering. i.e., the designer would like to move the user interface of a legacy application from a character-based screen to a graphical one, or we could require retargeting the legacy application from standard desktop PC to the web.

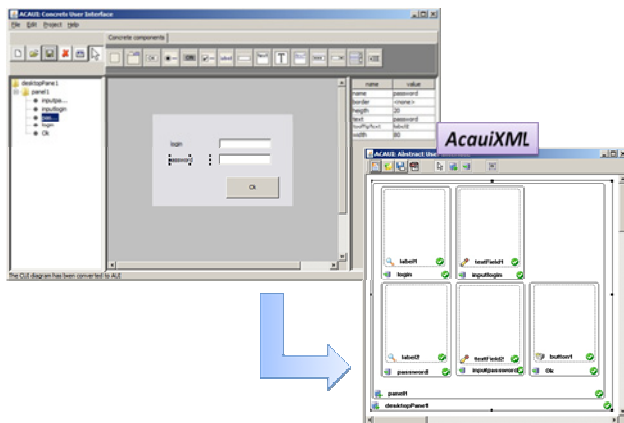
This process can be supported by our suite of tools to help the developer in the reverse engineering process.

In Fig. 1, the process is depicted. In the beginning the developer has only a screenshot of the legacy application. This screenshot can be loaded into *GuiLayout++*, where the designer can assess some metrics in the screenshot. i.e., the uniformity in the alignment in the layout of the components or the percentage devoted to each task can be analyzed. The different areas used for each purpose and its layout are specified by drawing boxes of the corresponding type. The different types of boxes are marked in different colors (see Fig. 4a). Then, the designer can change the screenshot to address the issues detected by *GuiLayout++*. An example of one of the metrics computed by *GuiLayout++* is shown in Fig. 4b. The graph in this figure displays the results for the metric to assess the amount of space devoted to each purpose in a web page. This is one of the metrics proposed in [11]. Alternatively, the designer can directly load a screenshot in *PureXML* (see Fig. 5).

In *PureXML*, the designer loads a screenshot. In the example used in this paper, a well-known printing dialogue is used. Then the designer used the widget palettes provided (see the right part in Fig. 5) to add widgets. These widgets are organized in tabs, so they can be more easily found. Their *Look&Feel* takes inspiration from Balsamiq [9]. The designer uses *Drag&Drop* to

drag the corresponding widget into the screenshot, and then he adjusts its size to the real one in the screenshot. i.e., when the designer finds a dropdown list in the screenshot, a dropdown list is dragged from the *List, Trees and Tables* tab of the widget palette, and then its size is adjusted. The designer proceeds for all widgets in the screenshot. Notice how a tree is created simultaneously to represent the widgets added to the screenshot (see the top left part in Fig. 5). One important issue that *PureXML* addresses is navigation. A user interface is usually composed of more than one window, and there-

fore more than one screenshot.



**Figure 3. AcauiXML: from concrete user interface to abstract user interface.**

Thus, in PureXML the designer can load many screenshots. They are shown in the same place as the tree in the left part of the window. The designer can toggle both views by using the tabs *Design* and *Navigation*.

In the example in the bottom left part there is a window where the designer has modeled a transition from *Properties* button to the *Properties* dialogue window. In the properties of the transition the designer can specify the events that trigger the transition. Thus, the designer is supported in modeling which widgets the user interface is composed of, and how the user navigates from one window to another.

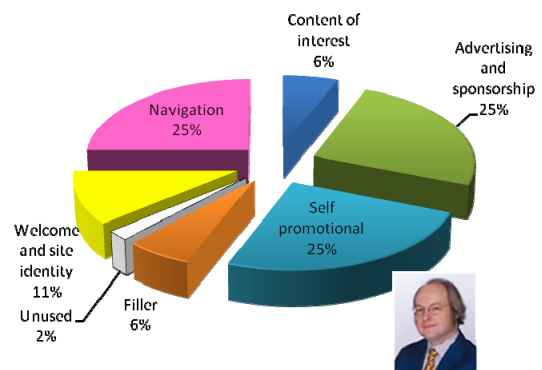
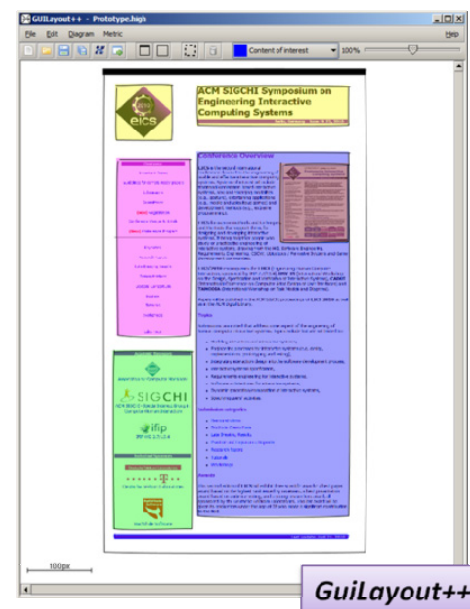
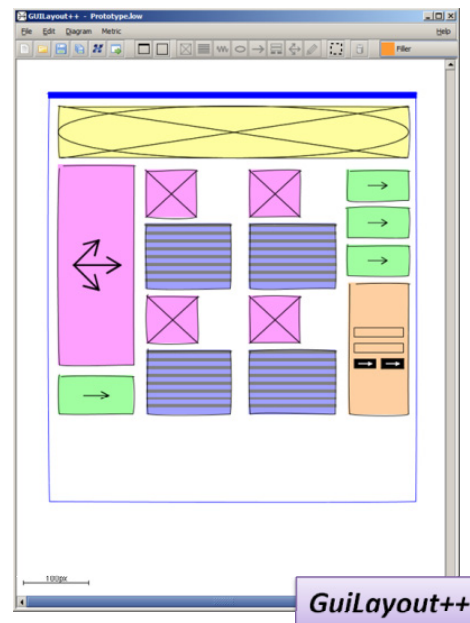
By selecting any widget added, the designer can specify some properties of the widget. The set of properties supported is taken from the current UsiXML 2.0 concrete user interface draft. Thus, the designer can specify attributes such as, the fonts, the colors, etc. All the project can be saved in UsiXML 2.0 concrete user interface XML syntax. This output can be then used in AcauiXML.

AcauiXML takes a concrete user interface specification and it obtains an abstract user interface (see Fig. 3). The visual syntax currently used is the same one we designed for *IdealXML*.

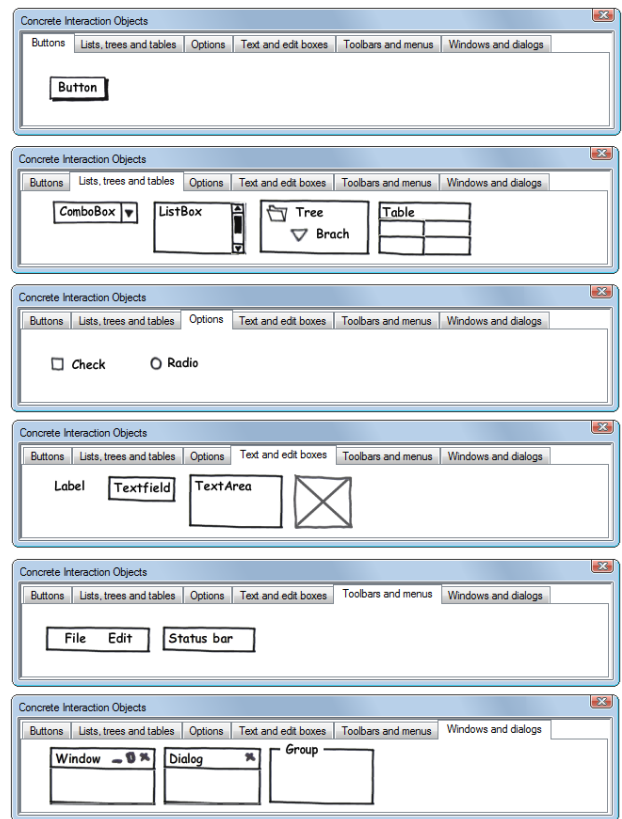
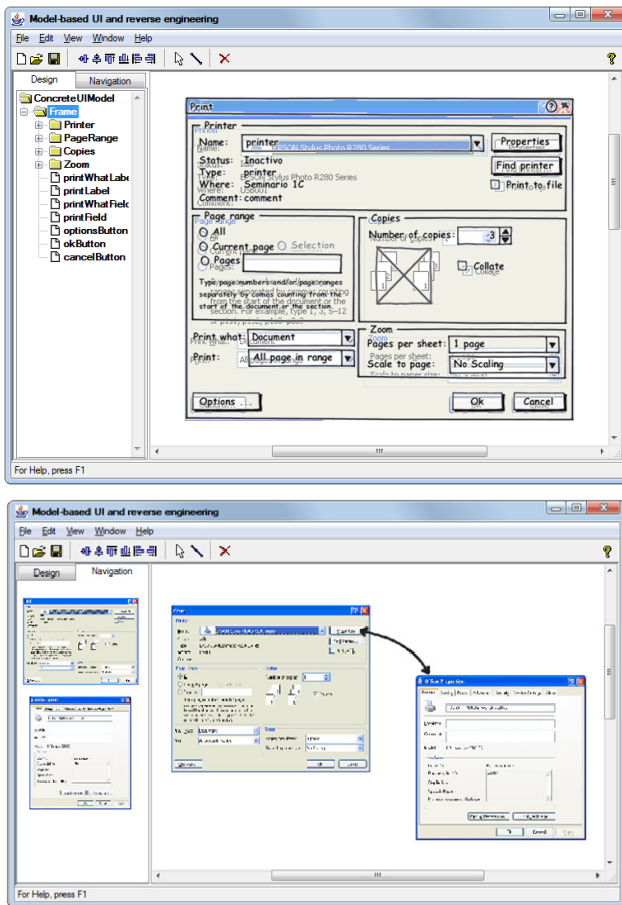
Finally, the abstract user interface generated by AcauiXML can be used to obtain a task model. The task model is a perfect starting point for reengineering an application, and starting from the task a forward engineering process to produce a new application out of the starting legacy one. In this task model unused features could be removed, and some extra features could be added.

## DISCUSSION AND CONCLUSION

Currently, user interface description languages present many challenges and difficulties, among these challenges are the following: effective, efficiency and satisfactory software tools for supporting prototyping, evaluation and developing. But, many times developing must be done following a reverse path.



**Figure 4. GuiLayout++: prototyping and assessing the user interface: (a) Prototyping, (b) Assessing.**



**Figure 5. PureXML: from final user interface to concrete user interface: (a) The editor window tabs (Design and Navigation), (b) The tabs for the UI component palette.**

In this paper a suite of tools is introduced. These tools are related to evaluation and reverse engineering. Prototyping is used in some of these software tools also. We identified a good joint-venture in the combination of prototyping and UI description languages. Prototyping is a very good common ground to communicate stakeholders involved in a development, between them, for instance, designers, final users, clients and developers.

Moreover, prototyping and evaluation are key elements in any user-centered design technique. *GuiLayout++* is useful in this scope, we can specify applications and we can evaluate what space distribution is used. In a similar scenario, *PureXML* is introduced. By using *PureXML*, concrete specifications of user interfaces can be built.

We identified an agile tendency in the development of user interfaces and software products in general. Under this point of view, several activities in the development process can be done simultaneously. Prototyping, specification and evaluation activities should be supported by available tools. In this paper a suite of tools with this premise in mind was introduced.

Additional improvements in our suite of tools can be considered. Previous software tools were done independently and integration efforts are considered in this moment.

A previous version of *UsiXML* was used for developing some previous software tools, namely 1.8, but a new release and review of this language will be made soon officially available and modifications and updates will be done. Therefore, some of our tools require some updating efforts

Reverse engineering is an interesting path in the specification, modification and optional development of software tools. In this sense, additional efforts will be done in the forward engineering direction. That is, suite of tools introduced in this paper will be integrated with *idealXML*[5] in order to provide a new version from previously non-documented or available software products.

## ACKNOWLEDGMENTS

This research has been partially funded by Spanish Ministry of Science and Innovation grant TIN2008-06596-C02-01 and the grant PEI09-0054-9581 from the Regional Government of Castilla-La Mancha. We would like to thank also Miguel Oliver, Abraham Martínez y Francisco Javier Muñoz for collaborating in the implementation of the tools.

## REFERENCES

1. Bouillon, L., and Vanderdonckt, J., Retargeting Web Pages to other Computing Platforms with VAQUITA. In *Proc. of IEEE Working Conf. on Reverse Engineering WCRE'2002* (Richmond, 28 October-1 November 2002). A. van Deursen, E. Burd (Eds.). IEEE Computer Society Press, Los Alamitos (2002), pp. 339-348.
2. Chikofsky, E.J., and Cross, J.H., II. Reverse engineering and design recovery: a taxonomy. *IEEE Software* 7, 1 (Jan. 1990), pp. 13-17
3. Coyette, A., Kieffer, S., and Vanderdonckt, J. Multi-fidelity Prototyping of User Interfaces. In *Proc. of 11th IFIP TC 13 Int. Conf. on Human-Computer Interaction INTERACT'2007* (Rio de Janeiro, 10-14 September 2007), Lecture Notes in Computer Science, vol. 4662, Springer-Verlag, 2007, pp. 149-162.
4. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., and Víctor López Jaquero. UsiXML: a Language Supporting Multi-Path Development of User Interfaces. In *Proc. of 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems EHCDVIS'2004* (Hamburg, July 11-13, 2004). Lecture Notes in Computer Science, vol. 3425. Springer, Berlin (2004), pp. 200-220.
5. Montero, F., López Jaquero, V. IdealXML: An Interaction Design Tool and a Task-Based Approach to User Interface Design. 6th International Conference on Computer-Aided Design of User Interfaces (CADUI 2006), Bucharest, Romania, 6-8, June, 2006.
6. Montero, F., López-Jaquero, V. Guilayout++: Supporting Prototype Creation and Quality Evaluation for Abstract User Interface Generation. Proceedings of the 1st Workshop on User Interface eXtensible Markup Language Workshop UsiXML'2010 (Berlin, June 20, 2010). Thalès, Paris, pp. 39-44.
7. Pérez, J. Anaya, V., Cubel, J.M., Domínguez, F., Boronat, A., Ramos, I., Carsí, J.A. Data Reverse Engineering of Legacy Databases to Object Oriented Conceptual Schemas, Software Evolution Through Transformations: Towards Uniform Support throughout the Software Life-Cycle Workshop (SET'02), Barcelona (Spain), 2002.
8. WebRevEnge - Tool for Reverse Engineering: From HTML to CTT Task Models. <http://giove.isti.cnr.it/tools/WebRevEnge/home>
9. Balsamiq. Balsamiq mockups. <http://balsamiq.com/>
10. Bouillon, L., Limbourg, Q., Vanderdonckt, J., and Michotte, B., Reverse Engineering of Web Pages based on Derivations and Transformations. In *Proc. of 3rd Latin American Web Congress LA-Web'2005* (Buenos Aires, October 31-November 2, 2005), IEEE Computer Society Press, Los Alamitos, 2005, pp. 3-13.
11. Nielsen, J., Tahir, M. Homepage Usability: 50 Websites Deconstructed, 2001.

# UsiXML Concrete Behaviour with a Formal Description Technique for Interactive Systems

Eric Barboni, Célia Martinie, David Navarre, Philippe Palanque, Marco Winckler

Institute of Research in Informatics of Toulouse, University of Toulouse

Interactive Critical Systems (ICS) team

118, route de Narbonne, F-31042 Toulouse Cedex 9 (France)

{barboni, martinie, navarre, palanque, winckler}@irit.fr

## ABSTRACT

In the last years User Interface Description Languages (UIDL) such as UsiXML appeared as a suitable solution for developing interactive systems. So far, there have been several attempts for exploring the potential of UsiXML as a language for describing user interface components for multi-target platforms. In this paper we are concerned by the behavioural aspect of interactive system built using UsiXML. In order to implement reliable and efficient applications, we propose to employ a formal description technique called ICO (Interactive Cooperative Objects) that have been developed to cope with complex behaviours of interactive systems including event-based and multimodal interaction. Our approach offers a bridge between UsiXML descriptions of the user interfaces components and a robust technique for describing behaviour using ICO modelling. Beyond that, this paper highlights how it is possible to take advantage from the two approaches to make possible to provide a model-based approach for prototyping interactive systems. The approach is fully illustrated by a case study using the ARINC 661 specification for User Interface components embedded into interactive aircraft cockpits.

## Author Keywords

Behavioural modelling, Interactive systems, User Interface Description Languages (UIDLs), UsiXML, ARINC 661.

## General Terms

Design, Reliability, Human Factors, Theory.

## Categories and Subject Descriptors

D.2.2 [Software Engineering]: Design Tools and Techniques – *User interfaces*. H.5.2 [Information Interfaces and Presentation]: User Interfaces – *User-centered design*.

## INTRODUCTION

In the last years User Interface Description Languages (UIDLs) appeared as a suitable solution for developing interactive systems [7][**Error! Reference source not found.**][18]. In this scenario UsiXML [10] appears as an emergent candidate for describing interactive system, in particular those sought to be deployed in different platforms [21]. It is widely agreed that a UIDL must cover three different aspects of the User Interface (UI): to de-

scribe the static structure of the user interfaces (i.e. presentation part which ultimately includes the description of user interface elements, e.g. widgets, and their composition), to describe the dynamic behaviour (i.e. the dialog part, describing the dynamic relationships between components including event, actions, and behavioural constraints) and to define the presentation attributes (i.e. look & feel properties for rendering the UI elements). Among the models involved in User Interface (UI) development, dynamic behaviour is one of the most misunderstood and one of the most difficult to exploit [6][23]. Dialog models play a major role on UI design by capturing the dynamic aspects of the user interaction with the system which includes the specification of: relationship between presentation units (e.g. transitions between windows) as well as between UI elements (e.g. activate/deactivate buttons), events chain (i.e. including fusion/fission of events when multimodal interaction is involved) and integration with the functional core which requires mapping of events to actions according to predefined constraints enabling/disabling actions at runtime.

These problems related to the description of behavioural aspects of interactive systems have been discussed in detail in [15]. Among the techniques presented, it is worth of mention the Interactive Cooperative Objects (ICO) formalism which is a formal description technique designed to the specification, modelling and implementation of interactive systems. ICO has been demonstrated efficient for describing several techniques including 3D, multimodal interaction techniques and dynamic reconfiguration of interactive systems [16]. ICO models are executable and fully supported by the CASE tool PetShop [4] which has been shown effective for prototyping interactive techniques [14].

In this paper we propose a model-driven approach to integrate behaviour described using ICO models and user interface components described with UsiXML. By using ICO models is possible to run the Petshop environment to control the execution of the application. This approach has already been demonstrated efficient to model the behaviour of user interface components based on the standard ARINC 661 for interactive aircraft cockpits [1][2][3][13]. In section 2 we present an overview of behavioural aspects in UsiXML and how these issues have been treated by the re-



search community. Section 3 introduces the standard ARINC 661 and how user interface components described by this standard can be implemented using UsiXML. Section 4 introduces the case study. Section 5 is devoted to the specification of the behaviour of user interface components. In section 6 we present a proposal for extending the concrete behavioural description within UsiXML. Finally, section 7 presents conclusions and future work.

#### UsiXML AND BEHAVIOURAL DESCRIPTIONS

UsiXML (USer Interface eXtensible Markup Language) is defined in a set of XML schemas where each schema corresponds to one of the models containing attributes and relationships in the scope of the language [10]. UsiXML schemas are used to describe at a high level of abstraction the constituting elements of the UI of an application including: widgets, controls, containers, modalities, interaction techniques, etc.

The UsiXML language is structured according to the four levels of abstractions as proposed by the framework Cameleon [7], as follows: *task models*, *abstract user interfaces (AUI)*, *concrete user interface (CUI)* and *final user interface (FUI)*. Several tools [12] exist for editing specification using UsiXML at different levels of abstraction. Notwithstanding, developers can start using UsiXML schemas at the abstraction level that better suits their purposes.

As far as the behaviour is a concern, there are some dedicated schemas in UsiXML. At the task level, behaviour is covered by task models featuring operators in a similar way as it is done by CTT [11]. At the AUI and CUI levels several schemas allow to describe basic elements of the dialog behaviour including *events*, *triggers*, *conditions*, and *source* and *target* components. These elements can be refined at the FUI level to reach final constructs implemented by the target platform.

So far there is limited support for UsiXML schemas related to behavioural aspect of interactive systems beyond the task model level. Some extensions have been proposed to describe high level dialog behaviours such as those implemented by transitions between windows [22] and between states of workflow-based applications [9]. However, all these extensions are more or less related to task models.

The description of fine-grained behaviour in UsiXML is awkward as the behavioural aspect and the user interface composition are interleaved in a single description. So that, the description of events, triggers and actions is scattered along the components of the user interface with makes extremely difficult to visualize the behaviour of the current state of the application being modelled. Another conceptual issue with dialog modelling with UsiXML is related to the different levels of abstraction; whilst abstract containers can be easily mapped to windows, it is not so easy to envisage abstract behaviour and how to refine them into more concrete actions on the user interface.

A few works [17][23] have addressed the behaviour aspect of interactive system described with UsiXML. Schaefer, Bleul, and Mueller (2006) [17], propose an extension of UsiXML by the means of a dedicated language called *Dialog and Interface Specification Language* (DISL). The main contribution of that work is to propose clear separation between presentation, user interface composition and dialog parts of the interface. Winckler et al (2008) [23] suggest there is no need of new dialog language as UsiXML can be coupled with existing dialog modelling techniques such as StateWebCharts (SWC) [24] to deal with the behaviour of interactive systems. Those authors propose a set of mappings that allows SWC specification to be used as running engine for the behaviour of UsiXML specifications. Notwithstanding, the work was limited to navigation between web pages in Web-based user interfaces.

#### ARINC 661 SPECIFICATION AND UsiXML

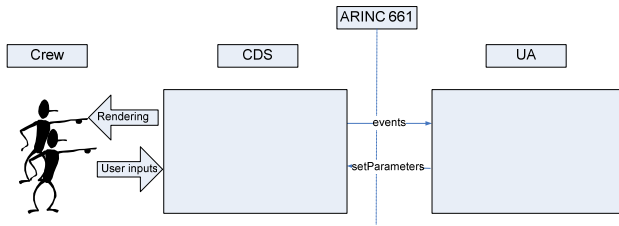
Even if the main topic of this contribution is to make a bridge between a description of the user interface using UsiXML and an external behavioural description, we firstly propose an overview of a similar work done on an aircraft standard for interactive application. Making a parallel with this previous work, we then highlight the basic bricks making possible to enhance UsiXML with a behavioural description. As illustrated in the next paragraphs, services offered by the ARINC 661 widgets and the definition of User Application (UA) are very close to UsiXML Concrete User Interface model.

The Airlines Electronic Engineering Committee (AEEC) (an international body of airline representatives leading the development of avionics architectures) formed the ARINC 661 Working Group to define the software interfaces to the Cockpit Display System (CDS) used in all types of aircraft installations. The standard is called ARINC 661 - Cockpit Display System Interfaces to User Systems [1][2]. In ARINC 661, a user application is defined as a system that has two-way communication with the CDS:

- Transmission of data to the CDS, possibly displayed to the flight deck crew.
- Reception of input from interactive items managed by the CDS.

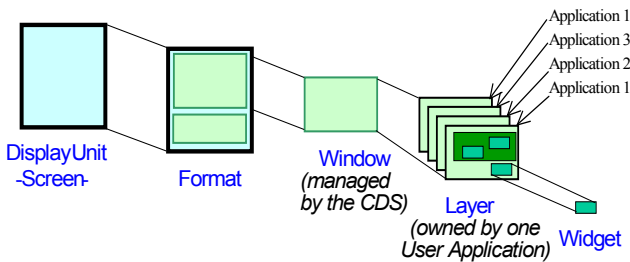
According to the classical decomposition of interactive systems into three parts (presentation, dialogue and functional core) defined in [5], the CDS part (in Figure 1) may be seen as the presentation part of the whole system, provided to the crew members, and the set of UAs may be seen as the merge of both the dialogue and the functional core of this system. ARINC 661 then puts on one side input and output devices (provided by avionics equipment manufacturers) and on the other side the user applications (designed by aircraft manufacturers). Indeed, the consistency between these two parts is maintained through the communication protocol defined by ARINC 661.





**Figure 1. Abstract architecture and communication protocol between Cockpit Display System and a User Application.**

The ARINC 661 Specification uses a windowing concept which can be compared to a desktop computer windowing system, but with many restrictions due to the aircraft environment constraints (see Figure 2).



**Figure 2. ARINC 661 Specification windowing architecture.**

The windowing system is split into 4 components:

- The display unit (DU) which corresponds to the hardware part,
- The format on a Display Unit (DU), consists of a set of windows and is defined by the current configuration of the CDS,
- The window is divided into a set of layers (with the restriction of only one layer activated and visible at a time) in a given window,
- The widgets are the smallest component on which interaction occurs (they corresponds to classical interactors on Microsoft Windows system such as command buttons, radio buttons, check buttons, among others).

In ARINC 661, a widget is defined with an identifier (widget type, widget identifier and widget parent), states (informal description of the relationship between these states) and some other descriptions:

- A **definition section** provides general information on the widget such as the categories it belongs to, a functional description of its behaviour and restrictions (if any) with respect to ARINC 661 principles.
- A **parameter table** provides the list of the widget parameters (position, size, availability...).
- A **creation structure table** presents the parameters required for the instantiation of the widget (kind, restrictions...).

- An **event structure table** presents the event notification structure. It describes the parameters that may be held by the events.
- A **run-time modifiable parameter table** presents the sets of parameters that may be changed at run-time.

For instance, a *PushButton* is defined as followed (only a subpart of the entire description is provided hereafter):

Categories:

Graphical representation, Interactive, Text string.

Description:

A PushButton widget is a momentary switched button, which enables a crew member to launch an action. A PushButton has only one inner state, so there is no need for an inner state parameter.

Restriction:

None.

PushButton event structure:

Event structure	Size(bits)	Value/Description
EventId	16	A661_EVT_SELECTION

PushButton Runtime Modifiable Parameters:

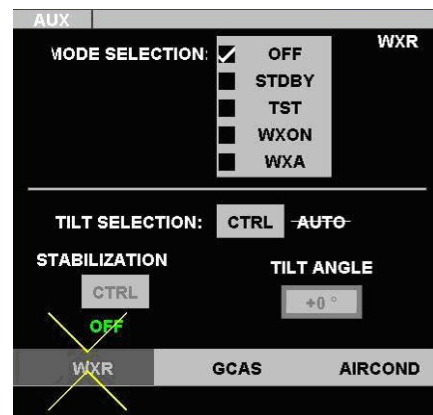
Parameter	Type	Size	Parameter Ident	Type of structure
Enable	Uchar	8	A661_ENABLE	...
Visible	Uchar	8	A661_VISIBLE	...
...				

...

In ARINC 661, a UA communicates with the CDS asking for modification of widgets parameters and receiving events from them. On the CDS side, the set of widgets is created and their layout is related to the use of the User Application Definition File (UADF). The content of this file, as well as the description of widgets is really close to the UsiXML model for a Concrete User Interface (even if it is not expressed using an XML-based format).

#### INFORMAL DESCRIPTION OF THE CASE STUDY

In order to illustrate our approach, we briefly introduce the MPIA application (which stands for Multi-Purpose Application) that we employ as case study (see Figure 3).



**Figure 3. WXA User Interface of the MPIA application.**

The MPIA is an application embedded into aircraft cockpits (see Figure 4) and it aimed for handling several flight parameters. It is made up of three pages (called WXR, GCAS and AIRCOND) between which a crew member is allowed to navigate. WXR page is in charge managing weather radar information; GCAS is in charge of the Ground Anti Collision System parameters while AIRCOND deals with settings of the air conditioning. Due to space reasons, we only focus on the WXR page. For the same reasons, we only on the ARNC 611 component *PushButton* that is used to build the buttons *WXR*, *GCAS* and *AIRCOND* as shown in the bottom-side of Figure 3.



Figure 4. The MPIA application in aircraft cockpit.

#### BEHAVIOURAL DESCRIPTION OF ARINC 661 WITH ICO

Such as UsiXML CUI model, ARINC 661 does not provide an explicit description of both the application and widgets behaviour. Previous works based on the ICO formal description technique [15] have been done in order to enhance ARINC 661 specification. In [13] we provide the basis for mapping parts of the ARINC 661 Specification into ICO constructs used to describe the behaviour of both widgets and UA. In [3] we present architecture to explicit rendering concerns based on SVG [19]. In [16] we improve the previous architecture to support both multimodal interaction and reconfiguration of input and output devices. In this section, we present an overview of this work.

##### The ICO formalism

The Interactive Cooperative Objects (ICO) formalism is based on concepts borrowed from the object-oriented approach (i.e. dynamic instantiation, classification, encapsulation, inheritance, and client/server relationships) to describe the structural or static aspects of systems, and uses high-level Petri nets to describe their dynamics or behavioural aspects. In the ICO formalism, an object is an entity featuring five components: a cooperative object (CO), an available function, a presentation part and two functions (the activation function and the rendering function) that correspond to the link between the cooperative object and the presentation part.

The **Cooperative Object** (CO) models the behaviour of an ICO. It states (by means of a high-level Petri net) how the object reacts to external stimuli according to its inner state.

Figure 5 shows the concepts of the Cooperative Object models including: *places* (i.e. used as variables for tracking the system state), *transitions* (i.e. elements processing changes in the system state) and *arcs* (i.e. connecting places and transitions in a graph). *Arcs* can indicate input/output for tokens circulating in the graph; notice that an input arc (i.e. *InputArc*) can be extended to feature preconditions such as testing the availability of tokens in a place (i.e. *TestArc*) or preventing the movement of token accordingly to special conditions (i.e. *InhibitorArc*).

The variables associated to an arc are expressed by the concept *EString*. Tokens can hold values of any class in the system. The types of tokens that can circulate in a given place are denoted through the relationship with the concept *EClass*.

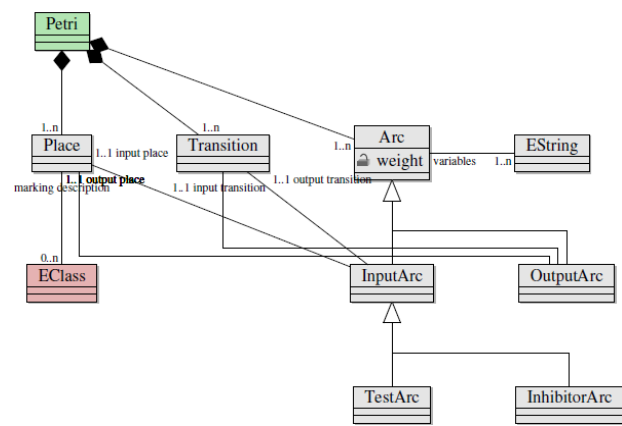


Figure 5. The Cooperative Objects meta-model.

The **presentation part** describes the external appearance of the ICOs. It is a set of widgets embedded into a set of windows. Each widget can be used for interacting with the interactive system (user interaction → system) and/or as a way to display information about the internal state of the object (system → user interaction).

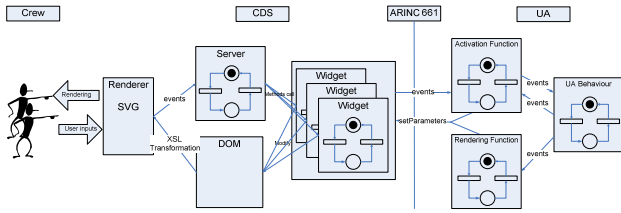
The **activation function** (user inputs: user interaction → system) links users' actions on the presentation part (for instance, a click using a mouse on a button) to event services.

The **rendering function** (system outputs: system → user interaction) maintains the consistency between the internal state of the system and its external appearance by reflecting system states changes through functions calls.

Additionally, an **availability function** is provided to link a service to its corresponding transitions in the ICO, i.e., a service offered by an object will only be available if one of its related transitions in the Petri net is available.

##### Architecture

The architecture presented in Figure 6 proposes a structured view on the findings from of a project dealing with formal description techniques for interactive applications compliant with the ARINC 661 specification.



**Figure 6. Detailed architecture compliant with ARINC 661 specification.**

The ICOs notation is exploited to model the behaviour of all the components of an interactive application compliant with ARINC 661 specification.

This includes each interactive component (i.e. widgets), the user application (UA) and the entire window manager (responsible for the handling of input and output devices, and the dispatching of events (both those triggered by the UAs and by the pilots) to the recipients (the widgets or the UAs). The two main advantages of the architecture presented in Figure are:

- Every component that has an inner behaviour (server, widgets, UA, and the connection between UA and widgets, e.g. the rendering and activation functions) is fully modelled using the ICO formal description,
- The rendering part is delegated to a dedicated language and tool (such as SVG, *Scalable Vector Graphics* [19]), thus making the external look of the user interface independent from the rest of the application, providing a framework for easy adaptation of the graphical aspect of cockpit applications. In this architecture the basic principle is to associate a document object model (DOM) to the set of widgets and to produce a SVG document using an XSLT transformation [26].

### Overview of the formal description using ICO

As illustrated by the above architecture, ICO is used to model several parts of the entire interactive system. In this section, we present the modelling of a simple widget and its link to the SVG rendering, then we briefly present the classical modelling of a user application, and finally we present parts of the server. The purpose here is to present a brief extracts to show all bricks of the modelling.

#### Modelling ARINC 661 widgets

For each widget in ARINC 661 specification document, we model:

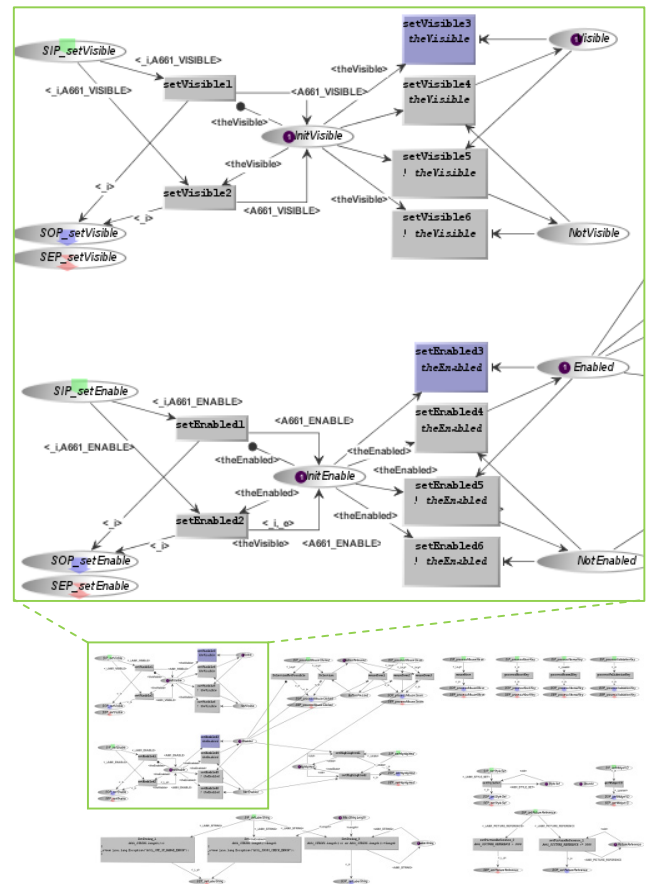
- Its behaviour using a Petri net.
- Its states (by the distribution of tokens in the places of the Petri net).
- The transition between the states.
- The rendering and activation function (which links the behaviour to the presentation part).

Modelling a widget follows the following process:

- Extract from ARINC 661 specification document the list of all the parameters
- Extract from ARINC 661 specification document the list of all the events it raises
- Build a software interface that exposes its run-time modifiable parameters, by providing an accessor for each parameter (i.e. a setXXX method for each XXX run-time modifiable parameter)
- Edit the Petri net model for which a skeleton has been generated from the previous information.

By applying this process, we modelled 12 widgets (from classical buttons, to complex containers such as a *Tabbed\_Panel\_Group*). Hereafter we present the modelling of a widget called *Picture\_Push\_Button* as an example. A *Picture\_Push\_Button* is a widget that is made up of 5 run-time modifiable parameters (*Enable*, *Visible*, *StyleSet*, *LabelString* and *PictureReference*) and raises 1 event (*A661\_EVT\_SELECTION*).

The upper side of the Figure 7 presents a zoom on the behaviour of this widget that handles the modification of the two parameters *Visible* and *Enable*. The bottom part of Figure 7 shows the connections of this widget and model describing the whole behaviour of the WXR application.



**Figure 7. Behaviour model of the *PicturePushButton*.**

Figure 8 presents the rendering function associated to the widget *PicturePushButton*. The third column presents the DOM attribute modified when the inner state of the button changes (e.g. when the state of the Petri net changes). An XSLT transformation is then used to produce the SVG document that renders the widget.

ObCSNode	ObCS event	Modified DOM attribute
Visible	<i>token_enter</i>	Visible = true
Visible	<i>token_remove</i>	Visible = false
Enabled	<i>token_enter</i>	Enabled = true
Enabled	<i>token_remove</i>	Enabled = false
...		

**Figure 8. Rendering function of the *PicturePushButton*.**

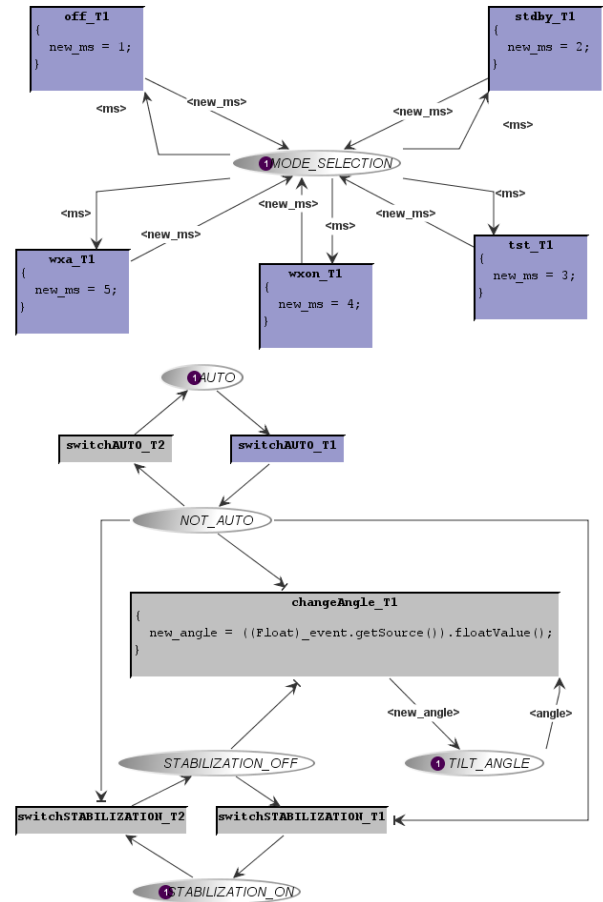
#### Modelling User Applications

Modelling a user application using ICO is quite simple as ICO has already been used to model such kind of interactive applications. Indeed, UAs in the area of interactive cockpits correspond to classical WIMP-based user interfaces (WIMP stands for Window, Icon, Menu, Pointing device). Figure 9 shows the entire behaviour of page WXR which is made up of two non-connected parts:

- The upper part aims at handling events from the 5 *CheckButtons* and the modification implied of the *MODE\_SELECTION* that might be one of five possibilities (OFF, STDBY, TST, WXON, WXA). Value changes of token stored in place *Mode-Selection* are described in the transitions while variables on the incoming and outgoing arcs play the role of formal parameters of the transitions.
- The lower part concerns the handling of events from the 2 *PicturePushButton* and the *EditBoxNumeric*. Interacting with these buttons will change the state of the application, allowing changing the tilt angle of the weather radar.

Figure 10 shows an excerpt of the activation function for page WXR, which describes the link between events availability and triggering and the behaviour of the application. For instance, the first line represents the link between the event *A661\_EVT\_SELECTION* produced by the button *auto\_PicturePushButton* and the event handler *switch* from the behavioural model of WXR (see Figure 9). If the event handler is available, the corresponding event producer (the button) should be enabled.

From this textual description, we can derive the ICO model as presented in [3]. The use of Petri nets to model the activation function is made possible thanks to the event communication available in the ICO formalism. As this kind of communication is out of the scope of this paper, we do not present the models responsible in the registration of events-handlers needed to allow the communication between behaviour, activation function and widgets.



**Figure 9. Behaviour of the page WXR**

Widget	Event	Event Handler
<i>auto_PicturePushButton</i>	<i>A661_EVT_SELECTION</i>	<i>switchAUTO</i>
<i>stab_PicturePushButton</i>	<i>A661_EVT_SELECTION</i>	<i>switchSTABILIZATION</i>
<i>tiltAngle_EditBox</i>	<i>A661_STRING_CHANGE</i>	<i>changeAngle</i>
...		

**Figure 10. Activation Function of the page WXR**

Figure 11 shows an excerpt of the rendering function, which describes how state changes within the WXR behaviour lead to rendering changes. For instance, when a token (*<float a>*) enters (i.e. *token\_enter*) the place *TILT\_ANGLE*, it calls the rendering method *showTiltAngle(a)* which displays the angle value into a textbox.

ObCSNode name	ObCS event	Rendering method
<i>MODE_SELECTION</i>	<i>token_enter &lt;int m&gt;</i>	<i>showModeSelection(m)</i>
<i>TILT_ANGLE</i>	<i>token_enter &lt;float a&gt;</i>	<i>showTiltAngle(a)</i>
...		

**Figure 11. Rendering Function of the page WXR**

The modelling of the rendering function into Petri nets works the same way as for the activation function, i.e. for each line in the rendering function, there is a pattern to ex-



press that in Petri nets (the interested reader may find more details in [3]).

#### Modelling User Interface Server

An important part of the above architecture is the user interface server that manages the set of widgets and the hierarchy of widgets used in the User Applications. More precisely, the user interface server is responsible in handling:

- The creation of widgets.
- The graphical cursors of both the pilot and his co-pilot.
- The edition mode.
- The mouse and keyboard events and dispatching it to the corresponding widgets.
- The highlight and the focus mechanisms.
- ...

As it handles much functionality, the complete model of such a server is complex and difficult to manipulate without an appropriate tool, and cannot be illustrated with a figure. In previous works [16], this server has been improved to support reconfiguration policies for both input and output devices and it has been enhanced too to support multiple mice interaction.

#### A PROPOSAL FOR CONCRETE BEHAVIOURAL DESCRIPTION WITHIN UsiXML

Beyond the obvious link that exists between the domain model of UsiXML and the behavioural description of an ICO, the work presented in the previous sections shows that there are common concerns between UsiXML CUI model and ARINC 661 specification (such as description of high level widgets and user interface, independent from implementation), and it shows that it is possible to enhance such descriptions with behavioural aspects.

With respect to the UsiXML architecture, the work done with ARINC 661 may be divided into two distinct parts, making possible to ease the design path from the concrete user interface to the final user interface.

#### An architecture making the bridge between ICO and UsiXML

As stated when discussing the architecture of Figure 6, it is possible to clearly separate behavioural aspects from rendering aspects. Figure 12 presents a first proposal for making UsiXML and ICO cooperate.

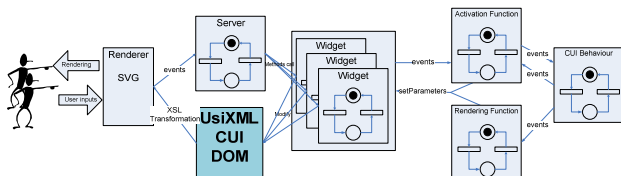


Figure 12. Detailed architecture compliant with UsiXML CUI.

As with ARINC 661, the main idea is to explicitly introduce behavioural models and make a clear link with the graphical representation. A successful integration should then lead to a UsiXML-based prototyping approach, inheriting from the prototyping capability of ICO.

#### Introducing behaviour at CUI level

Mapping state changes described using ICO description technique with UsiXML model attributes can be done easily. We illustrate the principle of introducing behavioural aspects at the CUI level with the example of the WXR application. These illustrations provide the key features allowing integration of ICO and UsiXML.

Figure 13 introduces a subpart of the CUI model of the WXR application, showing only a classical text box and a button:

- The *inputText* element *txt\_tiltAngle* aims at containing a number representing a tilt angle. In order to include such as information into the description of the user interface built using UsiXML we propose the inclusion of an attribute “*text*” that does not exist in the current version of UsiXML. Thus attribute “*text*” is used to host the corresponding rendering function as shown by Figure 15.
- When clicked, the button *btn\_switchAUTO* produces an event “*switchAUTO*”. Both the availability of this event and its occurrence are related to the behaviour of the application (as stated by the next paragraphs).

```
<cuiModel id="WXR-cui_1" name="WXR-cui">
  <window id="window_component_0"
    name="window_component_0" width="456"
    height="416">
    <inputText id="txt_tiltAngle"
      name="txt_tiltAngle" isVisible="true" isEna-
      bled="true" textColor="#000000" maxLength="50"
      numberOfColumns="15" isEditable="true" text=""/>
    <button id="btn_switchAUTO" name="btn_switchAUTO"
      isVisible="true" isEnabled="true" textCol-
      or="#000000">
    <behavior>
      <event id="switchAUTO" eventType="action"
        eventContext=""/>
    </behavior>
  </button>
  ...
</window>
</cuiModel>
```

Figure 13. Part of the CUI model of the WXR application.

Making the link between the behaviour of the application expressed using ICO (as illustrated by Figure 9) is quite easy as there can be a direct mapping of the event produced

by the button (“*switchAUTO*”) and the available event handler of the behaviour of WXR (“*switchAUTO*”), as shown in Figure 14.

Widget	Event	Event Handler
<i>btn_switchAUTO</i>	<i>switchAUTO</i>	<i>switchAUTO</i>
...		

**Figure 14. Activation Function of the page WXR.**

When the event handler is enabled, the attribute enabled of the button is thus set to “*true*”, “*false*” otherwise. Describing the rendering of the application is linked to attribute modification of the CUI DOM such as described by Figure 15.

ObCSNode name	ObCS event	CUI attribute
<i>TILT_ANGLE</i>	<i>token_enter</i> <float <i>a</i> >	“text” <i>txt_tiltAngle</i>
...		

**Figure 15. Rendering Function of the page WXR.**

When the token enters the place *TILT\_ANGLE*, the attribute “text” of the *inputText* element of the CUI is modified with the value hold by the token.

#### An executable CUI as a prototype for FUI

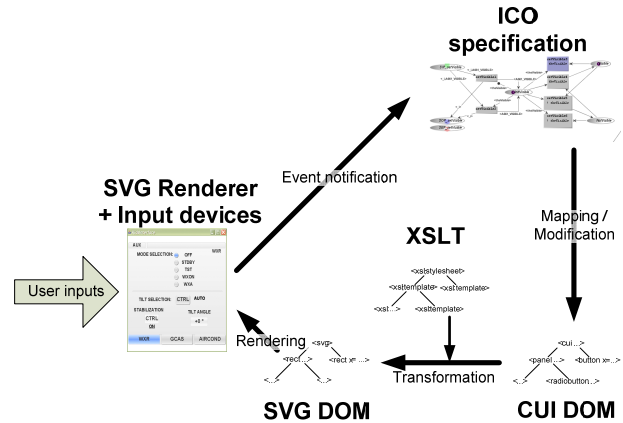
Thanks to the possibility of executing Petri nets, ICO allows prototyping when connected to the graphical representation of an application [14]. For instance, the MPIA application (from which WXR is extracted) has been fully modelled and can be executed on the CDS modelled using the ICO formalism. However, it has also been connected on a CDS developed on an experimental test bench as shown in Figure 4.

Providing a graphical representation of the CUI makes possible to build a prototype based our approach. Associating ICO and the CUI model has been discussed in the previous section, but it is possible too, in a similar way, to do this association at widget level, while proposing a way to render a CUI model based on a previous work integrating SVG [3]. Such a work should then allow the prototyping of the final UI (FUI) based on the bridge between ICO and a CUI model, shortening the design path to the FUI.

#### Rendering based on SVG

As stated in the previous section, any state change of the application is rendered via the modification of the CUI DOM, based on the mapping described by both the rendering and activation function. Figure 16 illustrates the run-time architecture that supports FUI prototyping based on the association of UsiXML and ICO. To provide a rendering to each CUI element, we propose the use of declarative descriptions of the graphical part that support transformations from conceptual models to graphical representations. The approach exploits both the SVG language [19]

for graphical representation, and the XSLT language for transformation (called a “*stylesheet*”).



**Figure 16. The run-time architecture.**

In order to write a stylesheet, one has to design the rendering of a particular widget, using Illustrator for example. When ready, the textual description of the widget is included in the stylesheet.

In our case, the source is the CUI DOM, built at start-up time, together with the instantiation of the ICOs components. Before running the application, the system must compile the stylesheet to an XSLT transformer. While running the application, every time the state of a CUI DOM variable changes, it is transformed into a DOM SVG tree, which in turn is passed to the SVG renderer and displayed.

#### Introduction of behaviour for widgets

To go further with a precise prototyping of the FUI, it is necessary to describe each widget, including its behaviour (such as already done with ARINC 661). As illustrated by Figure 17 and Figure 18, it is possible to describe the fine grain behaviour of a widget and the link of its inner state changes with rendering.

In its current state, UsiXML, via the CUI model, describes widgets as a type and a set of attributes (a button is defined by an id, a name...), making it abstract enough to be independent from the targeted platform for the FUI. But when considering prototyping, it may be interesting to provide a finer description of the kind of widget that is expected, and a less coarse grain description of the widgets attributes (for instance, it is possible to introduce rendering for any inner state of a button: armed, pressed...). Another interesting point when dealing with widget is the introduction of new widgets that may request a precise description of how it should work on the targeted platforms.

One possible way to allow such description within UsiXML could be to enhance the current platform model of the context model with a precise widget description. Even if no effort has already been put on it, this way is an important part of our future works.



## DISCUSSION AND OUTLOOK

Most of the recent work on UsiXML have been focused on mapping UsiXML schemas between several levels of abstraction [11][20] or proving automatic user interface generation of components to multi-target devices [12][21]. Indeed, very few works have focused on the behavioural aspect of interactive systems modelled with UsiXML.

This paper has presented a bridge between an already existing formal description technique for behavioural aspects of interactive systems and an approach for describing the presentation part of such system. Beyond that, it highlights how it is possible to take advantage from the two approaches to make possible to provide a model-based approach for prototyping interactive systems.

Such as highlighted by the Arch architecture, this approach allows a clear separation between graphical aspects, behavioural aspects and functional aspects. It allows too a clear separation with tasks such as with the work done in [4]. Such a separation is necessary as, depending on the functional part of the interactive system, constraints independent from task concerns can appear. In the example used in this paper, the value of the tilt angle must meet the system requirements and the dialogue is thus specially designed to support this constraint. If the functional part changes, the dialog part must be modified, but not the user's tasks.

It is noteworthy that the use of ICO models to describe the behaviour of user interfaces allows overcoming of some of the limitations of other UIDL languages such as SCXML [25] XUL [27] such as the easier management of infinite states, the encapsulation of variables as objects of any kind and dynamic instantiation of objects. Moreover, properties of UI descriptions can be formally assessed using the underlying Petri Net formalism.

Three ways of improvement for this work could be:

1. As presented in the previous section, a possible extension of our work is to introduce a notation or to enhance the current context model of UsiXML with a precise widget description, including its behaviour, making possible to build prototypes of the FUI.
2. As presented with classical widgets, such an approach can be used to precisely describe new interactive components.
3. A link from task models and abstract UI to concrete UI could be done based on the work we have already done about putting into correspondence task models and system model [4]

To make this work more "concrete" a particular effort has to be performed to integrate already existing tool support or to point out new developments. These issues are currently being addressed by our team at the IRIT (*Institute of Research in Informatics of Toulouse*) and the CNES (*Centre National d'Etudes Spatiales*) in a recently started Research & Technology project called ALDABRA.

## ACKNOWLEDGEMENTS

This work is supported by the Research & Technology Project (RT) ALDABRA (IRIT-CNES).

## REFERENCES

1. ARINC 661, Prepared by Airlines Electronic Engineering Committee. Cockpit Display System Interfaces to User Systems. ARINC Specification 661. (2002).
2. ARINC 661-2, Prepared by Airlines Electronic Engineering Committee. Cockpit Display System Interfaces to User Systems. ARINC Specification 661-2; (2005).
3. Barboni, E., Conversy, S., Navarre, D., Palanque, P. Model-Based Engineering of Widgets, User Applications and Servers Compliant with ARINC 661 Specification. In *Proceedings of the 13th conference on Design Specification and Verification of Interactive Systems DSVIS'2006*. LNCS, Springer Verlag.
4. Barboni, E., Ladry, J-F, Navarre, D., Palanque, P., Winckler, M. Beyond Modelling: An Integrated Environment Supporting Co-Execution of Tasks and Systems Models. In *Proc. of the ACM SIGCHI conference Engineering Interactive Computing Systems EICS'2010* (Berlin, June 19-23, 2010). ACM Press, New York (2010), pp. 143-152.
5. Bass L. et al. A metamodel for the runtime architecture of an interactive system: the UIMS tool developers workshop. *SIGCHI Bulletin* 24, 1 (1992), pp.32-37.
6. Book, M., Gruhn, V.: Fine-Grained Specification and Control of Data Flows in Web-based User Interfaces. *Journal of Web Engineering* 8, 1 (2009), pp. 48-70.
7. Calvary, G., Coutaz J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J. A. Unifying Reference Framework for Multi-Target User Interfaces. *Interacting With Computers* 15, 3 (2003), pp. 289-308, 2003.
8. Guerrero-García, J., González-Calleros, J.M., Vanderdonckt, J., and Muñoz-Arteaga, J. A Theoretical Survey of User Interface Description Languages: Preliminary Results. In *Proc. of Joint 4<sup>th</sup> Latin American Conference on Human-Computer Interaction-7th Latin American Web Congress LA-Web/CLIHIC'2009* (Merida, November 9-11, 2009). E. Chavez, E. Furtado, A. Moran (Eds.). IEEE Computer Society Press, Los Alamitos (2009), pp. 36-43.
9. Guerrero-García, J., Vanderdonckt, J., and González-Calleros, J.M. FlowiXML: a step towards designing workflow management systems. *International Journal of Web Eng. Technol.* 4, 2 (2008), pp. 163-182.
10. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., and Víctor López Jaquero. UsiXML: a Language Supporting Multi-Path Development of User Interfaces. In *Proc. of 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verifica-*

- tion of Interactive Systems EHCI-DSVIS'2004 (Hamburg, July 11-13, 2004). Lecture Notes in Computer Science, vol. 3425. Springer, Berlin (2004), pp. 200-220.
11. Montero, M., López-Jaquero, V., Vanderdonckt, Gonzalez, and P. Lozano. Solving the Mapping Problem in User Interface Design by Seamless Integration in IdealXML. In *Proc. of DSV-IS'2005* (Newcastle upon Tyne, 13-15 July 2005). S.W. Gilroy, M.D. Harrison (Eds.). Lecture Notes in Computer Science, vol. 3941. Springer-Verlag, Berlin (2005), pp. 161-172.
  12. Michotte, B., and Vanderdonckt, J. GrafiXML, A Multi-Target User Interface Builder based on UsiXML. In *Proc. of 4<sup>th</sup> Int. Conf. on Autonomic and Autonomous Systems ICAS'2008* (Gosier, 16-21 March 2008). IEEE Computer Society Press, Los Alamitos (2008), pp. 15-22.
  13. Navarre, D., Palanque, P., and Bastide, R. A Formal Description Technique for the Behavioural Description of Interactive Applications Compliant with ARINC 661 Specifications. In *Proc. of HCI-Aero'04* Toulouse, France, 29 September-1st October 2004
  14. Navarre, D., Palanque, P., Bastide, R., and Sy, O. A Model-Based Tool for Interactive Prototyping of Highly Interactive Applications. In *Proc. of 12th IEEE, International Workshop on Rapid System Prototyping*; Monterey (USA). IEEE; 2001.
  15. Navarre, D., Palanque, P., Ladry, J.F., Barboni, E. ICOs: A model-based user interface description technique dedicated to interactive systems addressing usability, reliability and scalability. *ACM Transactions on Computer-Human Interaction* 16, 4 (November 2009).
  16. Navarre, D., Palanque, P., Ladry, J.F., Basnyat, S. An Architecture and a Formal Description Technique for User Interaction Reconfiguration of Safety Critical Interactive Systems. In *Proc. of 15th Int. Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'2008* (Kingston, July 16-18, 2008). Lecture Notes in Computer Sciences, vol. 5136. Springer, Berlin (2008).
  17. Schaefer, R., Bleul, S., Mueller, W. 2006. Dialog modeling for multiple devices and multiple interaction modalities. In *Proceedings of the 5th international conference on Task models and diagrams for users interface design TAMODIA'06*. Karin Coninx, Kris Luyten, and Kevin A. Schneider (Eds.). Springer-Verlag, Berlin, Heidelberg, 39-53.
  18. Shaer, O., Green, M., Jacob, R.J.K, and Luyten, K., User Interface Description Languages for Next Generation User Interfaces. In *Proc. of Extended Abstracts of CHI'08*, ACM Press, New York (2008), pp. 3949-3952.
  19. SVG W3C 2003: Scalable Vector Graphics (SVG) 1.1 Specification <http://www.w3.org/TR/SVG11/>
  20. Tran, V., Vanderdonckt, J., Kolp, M., Wautelet, Y., Using Task and Data Models for User Interface Declarative Generation. In *Proc. of 12th International Conference on Enterprise Information Systems ICEIS'2010* (Funchal, 8-10 June 2010), J. Filipe, J. Cordeiro (Eds.), Vol. 5, SciTePress, 2010, pp. 155-160.
  21. Trindade, F. M., and Pimenta, M. S. RenderXML - A Multi-platform Software Development Tool. In *Proc. of TAMODIA 2007*. LNCS, vol. 4849. Springer, Berlin (2007), pp. 293-298.
  22. Vanderdonckt, J., Limbourg, Q., Florins, M., Deriving the Navigational Structure of a User Interface. In *Proc. of 9th IFIP TC 13 Int. Conf. on Human-Computer Interaction INTERACT'2003* (Zurich, 1-5 September 2003). M. Rauterberg, M. Menozzi, J. Wesson (Eds.). IOS Press, Amsterdam (2003), pp. 455-462.
  23. Winckler, M., Trindade, F.M., Stanciulescu, A., Vanderdonckt, J., Cascading Dialog Modeling with UsiXML. In *Proc. of 15th Int. Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'2008* (Kingston, July 16-18, 2008). Lecture Notes in Computer Sciences, vol. 5136. Springer, Berlin (2008), pp. 121-135.
  24. Winckler, M.; Palanque, P. StateWebCharts: a Formal Description Technique Dedicated to Navigation Modeling of Web Applications. In *Proc. of Int. Workshop on Design, Specification and Verification of Interactive Systems DSVIS'2003* (Funchal, June 2003).
  25. World Wide Web Consortium. State Chart XML (SCXML): State Machine Notation for Control Abstraction. Working Draft 26 April 2011 Available at: <http://www.w3.org/TR/2011/WD-scxml-20110426/>
  26. XSL Transformations (XSLT). Version 1.0. W3C Recommendation 16 November 1999. Available at: <http://www.w3.org/TR/xslt>
  27. XUL (XML User Interface Language). Available at: <http://www.mozilla.org/projects/xul/> (August 10, 2011).

# An Abstract User Interface Model to Support Distributed User Interfaces

A. Peñalver, J. J. López-Espín, J. A. Gallud, E. Lazcorreta, F. Botella

Operations Research Center University Institute

Miguel Hernandez University of Elche, Spain

{a.penalver, jlopez, jgallud, enrique, Federico}@umh.es

## ABSTRACT

Recently, the traditional concept of user interface has been changing significantly. The development of new devices supporting new interaction mechanisms have changed traditional way in which people interact with computers. In this environment of strong technological growth, the increasing use of different displays managed by several users has improved user interaction. Combining fixed displays with wearable devices allows interaction and collaboration among users. Traditional user interfaces are evolving towards “distributed” user interfaces according to the new technological advances, allowing one or more interaction elements are distributed among many different platforms in order to support interaction with one or more users. In this new scenario, the Abstract User Interface model has been reviewed and modified to include specific characteristics from the Distributed User Interface point of view. This paper proposes a new Abstract User Interface that takes into account the possibility of distribution. Before presenting this new AUI model, the paper introduces the definition of the DUI concept and its foundations using a formal notation.

## Author Keywords

Disabled, elderly people, user interface description language, UsiXML, user model editor, virtual user.

## General Terms

Design, Human Factors, Theory.

## Categories and Subject Descriptors

D.2.2 [Software Engineering]: Design Tools and Techniques – *User interfaces*. H.5.2 [Information Interfaces and Presentation]: User Interfaces – *User-centered design*.

## INTRODUCTION

Distributed User Interfaces are gaining attention in many research groups due to the recent advances in the field of display and mobile technologies, among others. This evolution has affected definitively the concept of user interface, which has to be called now Distributed User Interface (DUI). Nowadays, a user can interact with computers by using new interaction ways thanks to the use of different displays, platforms and the total connectivity. A typical example of DUI could be AttachMe/DetachMe [8], where one single user can distribute UI elements across several platforms at run-time in order to accomplish a given task.

For example, the painter’s palette of this application can be migrated from a desktop PC to a mobile device such as a PDA in order to maximize the available screen space for painting. This way, the interaction elements of the palette (e.g. brushes, color palettes, pencils) would be available in the PDA, while the drawing would be showed in the desktop PC.

This paper presents a new Abstract User Interface model that takes into account the properties and features of the DUIs. Before introducing our proposal, we review the definition of user interface to give a more comprehensive approach. This is justified by the fact that the Graphical User Interfaces (GUI) no longer has much emphasis in the proposed new interaction devices.

The use of formal models for user interface design can help us to ensure coherency across designs for multiple platforms and prove properties such as consistency, reachability and completeness [1]. Previous efforts dedicated to specify user interfaces (UI) [3] must be revisited and redefined in order to consider this new interaction environment provided by distributed user interfaces, including for User Interface Description Languages (UIDLs) [12].

According to [13], DUI concerns to the allocation of one or many elements of one or many user interfaces to support one or many users carrying out one or many tasks on one or many domains in one or many contexts of use, each context of use consisting of users, platforms and environments. Authors explain that where UI distribution is supported, UI federation is needed. UI federation supports the concentration of UI elements from distributed sources.

Other previous studies have addressed, not always from a formal point of view, the specification of the essential properties of DUI’s, as well as reference model proposals [2,4,5,6,11]. In this paper, we define a user interface as a set of elements (input, output and control) that allows users to interact with different types of devices. The above definition of DUI is the starting point of the proposal described in this article. DUI adds the term *distribution* to the user interface concept.

We use the specification of Distributed User Interfaces (DUIs) that can be found in [10]. This formal view covers a wide range of descriptions from the most abstract to the implementation-oriented. Formal description techniques provide a means for producing unambiguous descriptions of complex interactions that occur in DUIs (distribution of

elements, including communication and distributed interaction), more precise and understandable than descriptions using the natural language. In addition, formal description techniques provide the foundation for analysis and verification of the descriptions provided. The analysis and formal verification can be applied to specific or abstract properties. Natural language is a good complement to the formal notation to get a first idea of the purpose of description.

The work is organized as follows. First we briefly describe the field of Distributed User Interfaces. Next section shows the characterization of the DUI concept. Section 3 presents the set of basic definitions that are the foundation of our concept of DUI. Section 4 presents our AUI model and its characteristics. Last section details the conclusions and future work.

### DUI CHARACTERIZATION

The definition of *Distributed User Interface* (DUI) is based on the definition of User Interface (UI). In this section we are reviewing each term included in the former definition of DUI: user interface elements, UI, user, task, domain, contexts of use. Some authors have proposed a new definition of UI concept by using the term Human User Interface (HUI) [7] to underline the fact that the interface concept is becoming an element that is “closer” to the user than the computer.

In the traditional definition, the interface is “closer” the computer or part of the computer indeed. This is, a UI is a set of elements that allows users to interact with computers. These elements can be categorized into input data elements, output data elements and control elements. This definition of UI supports all kind of technologies and interaction mechanisms.

The task can be defined as the set of actions the user performs to accomplish an objective. A DUI system is an application or set of applications that make use of DUIs, since these applications share the user interface. A DUI system can be implemented by means of several kinds of devices, hardware and software platforms. So, for the purposes of this paper, there is no need to maintain the difference between device and platform.

If we consider the former definition of DUI, we can define the following essential properties: portability, decomposability (and composability), simultaneity and continuity. The next paragraphs are devoted to describe each of the properties.

#### Portability

This property means that the UI as a whole, or elements of the UI, can be transferred among platforms and devices through easy user actions. For example, a user could be running a graphic editor in his/her desktop computer and then transfer the color palette panel (UI element) to another platform (a portable device) with a simple action.

#### Decomposability

A DUI system is decomposable if given a UI composed by a number of elements, one or more elements of that UI can be executed independently as a UI without losing their functionality. For example, a calculator can be decomposed in two UI elements, the display and the numeric keyboard. This property could be used along with Portability in order to allow the keyboard being executed in a smartphone meanwhile the display could be showed in a public display. These two UI elements could be also joined in a unique UI (composability).

#### Simultaneity

A DUI system is said simultaneous if different UI elements of the same DUI system can be managed in the same instant of time on different platforms. For example, two or more users could be using the same DUI system, each one interacting with one of the different platforms at the same time. This does not imply that all DUI systems are multiuser as we shall see later.

#### Continuity

A DUI system is said continuous if an element of the DUI system can be transferred to another platform of the same DUI system maintaining its state. For example, a user could be on a call in his/her mobile phone while walking down the street and transfer the call to the TV when he/she get home without interruption.

### BASIC DEFINITIONS AND SPECIFICATION OF ESSENTIAL PROPERTIES OF DUIS

In this section a set of concepts with the objective of obtaining a formal definition of a DUI is presented.

#### Definition 1: Interaction Element

An *Interaction Element*  $e \in E$  is defined as an element which allows a user  $u$  to carry out an interaction through a platform  $p$  (denoted by  $u \sim^e p$ ). An element can be defined as an input-data element  $u \sim^e, \rightarrow p$ , an output-data element  $u \sim^e, \leftarrow p$  or a control element  $u \sim^e, cp$ . In this work the generic notation  $e$  is used to enclose the three kinds of elements.

#### Definition 2: Functionality

Two elements of interaction  $e$  and  $e'$  have the same *functionality* if a user can perform the same action using them in his/her interaction with the device (denoted by  $e \stackrel{F}{=} e'$ ). In this sense, a button in a “Graphic Interface Unit” has the same functionality than a hand movement, if the computer receives the same order. In the same way, a sound has the same functionality as an audible alert if the user receives the same information as an answer to any interaction.

#### Definition 3: Target

A set of elements of interaction  $E_0 \subset E$  have the same *Target* ( $e \in {}^T E_0$ ) if  $\forall e \in E_0$ , a user  $u \in U$  obtains, through the functionality of  $e$ , an action of the task whose goal is to reach that target.

**Definition 4: User Interface**

A *User Interface* (UI)  $i \in UI$  is a set of interaction elements such as  $i = \{e \in E / e \in T_i\}$ , i.e., the user interface  $i$  is defined by the target for which that elements were chosen. From definitions exposed above it is possible to define a User Interface as a set of interaction elements which let a user to carry out a task in a specific context. After introducing the concepts of interaction elements, functionality and target, we can say that a user interface is simply a set of interaction elements that allow a user to perform a task in a context.

**Definition 5: Platform**

An interaction element  $e \in E$  exists in a platform  $p \in P$  (denoted by  $\sim^e p$ ), if  $e$  can be implemented, supported or executed on  $p$ . Thus, this definition also includes the existence of a framework that supports the interaction element  $e$ . A user interface  $i \in UI$  is supported on  $p \in P$  (denoted by  $u \sim^i p$ ) if  $\forall e \in i$  then  $u \sim^e p$  being  $u \in U$ . In addition,  $i \in UI$  is supported on a set of platforms  $P_0 \subset P$  ( $u \sim^i P_0$ ) if  $\forall e \in i$  then  $u \sim^e p$   $\forall p \in P_0$  being  $u \in U$ .

Essential properties explained in the aforementioned sections can be formalized following the proposed notation.

**Portability**

A user interface  $i \in UI$  /  $u \sim^i p$  being  $u \in U$  and  $p \in P$ , is *portable* if exists  $E_0 = \{e \in E / e \in i\} \subset i$  such as  $u \sim^{E_0} p'$  and  $u \sim^{\bar{E}} p$  being  $p, p' \in P$  reaching the same target than  $i$  (This property can be extended to more than one user).  $i \in UI$  has been *ported* if  $i$  is portable and this property is fulfilled.

**Decomposition**

A user interface  $i \in UI$  is *decomposable* if exists  $E_0 \subset i$  such as  $E_0 = \{e \in i / e \in T_{E_0}\}$  and  $\bar{E} = \{e \in i / e \in T_{\bar{E}}\}$  obtains the same target than  $i$ . Thus, if through  $i$  the target  $T$  is reached, then  $T$  and  $T'$  are two subtarget of  $T$  which can be reached through  $E_0$  and  $\bar{E}_0$  respectively. Note that from the definition of UI we can deduce that  $E_0$  and  $\bar{E}$  are two user interfaces (denoted by *User Subinterface* as it is shown in the next definition).

**Definition 6: User Subinterface**

Let us suppose that  $i \in UI$  is a user interface that allows a user  $u \in U$  to reach a target  $T$  on a platform  $p \in P$ , i.e.  $u \sim^T p$ . If  $T'$  is a subtarget of  $T$ , then the set  $i' = \{e \in T_i / e \in T_{i'}\}$  is a *User Subinterface* of  $i$ , and  $u \sim^{T'} p$ .  $i \in UI$  has been *decomposed* if it is decomposable and this property has been fulfilled.

**Definition 7: Distributed User Interface**

A *Distributed User Interface*  $di \in DUI$  is defined as a user interface which has been decomposed and ported. Then, a *Distributed User Interface*  $di \in DUI$  is defined as

$$di = \bigcup_{k=1}^N E_k = \bigcup_{k=1}^N \{e_{kj} \in i_k, j = 1 \dots N_k, e_{kj} \in T_k E_k\}$$

such as there exist  $n_p > 1$  platforms  $\{p_s \in P / s = 1 \dots n_p\}$  such as

$$di = \bigcup_{s=1}^{n_p} \{e_{sj} \in E / u \sim^{e_{sj}} p_s, j = 1 \dots n_{p_s}, e_{sj} \in T_{di}\}$$

for a user  $u \in U$  being  $T_k$  a subtarget of  $T \forall 1 \leq k \leq N$ .

Thus, a distributed user interface is a collection of interaction elements that forms a set of user interfaces, as well as the subinterfaces of the user distributed interface. These user subinterfaces are distributed in platforms without losing their functionality and their common target.

Using this new notation it is possible to express the interaction of a user through traditional UIs as  $u \sim^i p$ , being  $i \in UI$ , the interaction of a user through DUIs as  $u \sim^{di} p$  being  $di \in DUI$ , and the interaction of some users through some platforms through DUIs as  $\{u / u \in U\} \sim^{di} \{p / p \in P\}$ .

**Definition 8: State of a User Interface**

The *Status* of a user interface  $i \in UI$ , denoted by  $S(i)$ , is defined as the temporal point in which  $i$  lies after the user has used part of its elements with the goal of reaching the target associated to  $i$ . A state of  $i$  is the *Initial State* ( $S_0(i)$ ) either none of the elements have been used or some elements have been used, but they do not contribute to reach the target of  $i$ . The *Final State* of  $i$  ( $S_F(i)$ ) is reached when the target of  $i$  is reached. It is said that this target is achieved in  $n$  steps or states, if through the sequence  $S_0(i), \dots, S_n(i)$  the target of  $i$  is fulfilled. Note that moving from the state  $S_j(i)$  to  $S_{j+1}(i)$  requires to use the appropriate interaction element  $e \in i$ . Other elements used do not change the state. Furthermore, there exists elements which move from state  $S_j(i)$  to  $S_{j-1}(i)$ , to  $S_F(i)$  or to  $S_0(i)$ .

**Definition 9: State of a Distributed User Interface**

The *State* of a Distributed User Interface  $di \in DUI$ , denoted by  $\mathcal{S}(di) = (S(i_1), \dots, S(i_n))$ , is defined as a  $n$ -tuple where each element corresponds to the state of each user interface in which  $di$  has been decomposed. Note that  $\mathcal{S}(di)$  depends on the decomposition of  $di$  in subinterfaces and those that have been ported to different platforms.

We say that  $di$  is in an initial state if  $\mathcal{S}_0(di) = (S_0(i_1), \dots, S_0(i_n))$ , and we also say that  $di$  is in a final state if  $\mathcal{S}_F(di) = (S_F(i_1), \dots, S_F(i_n))$ . The number of states required to reach the target of  $di$  is the product of the number of states required to reach each subtarget in each ported user subinterface in which  $di$  have been divided.

**Simultaneity**

An distributed user interface  $di \in DUI$  is *simultaneous* in  $p_0, p_1, \dots, p_n \in P$  with  $n > 1$  for  $u_k \in U$  with  $k = 1, \dots, n_u$  ( $n_u \geq 1$ ) users, if  $di = \bigcap_{j=1}^N i_j$  with  $i_j \in UI$ , and  $u_k \sim^i p_s$  in the same temporal point, with  $j = 1 \dots N$  and  $s = 1 \dots n$  and  $k = 1 \dots n$ .

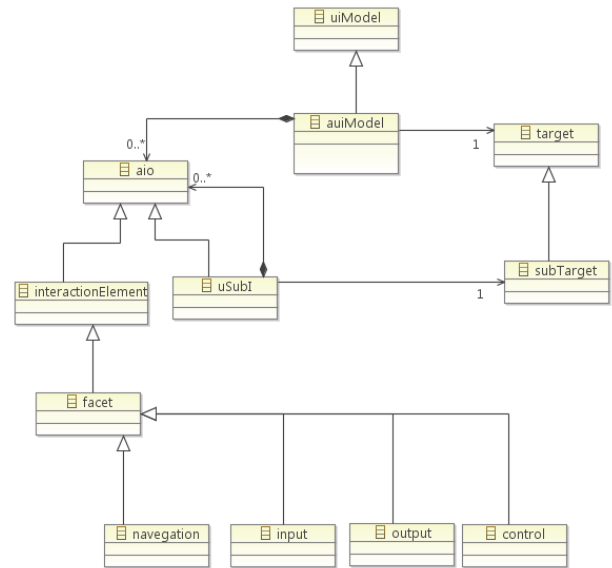
An distributed user interface  $di \in DUI$  is *continuous* in  $p_0, p_1 \in P$  if  $\forall e \in di, u \sim^e p_0$  and  $u \sim^e p_1$  maintaining the state of  $di$ , i.e., being  $S_j(di)$  the state of  $di$ , in both cases  $S_i(i)$  is reached (being able to be  $t=0, j+1, j-1, F$ ). We can illustrate these definitions using a simple example: the calculator. A calculator application consists of two main sets of interaction elements: the calculator screen or display and the calculator key pad. These two sets can be considered as a user subinterface (a container of other interaction elements). The key pad is also a set of interaction elements, being each numeric button an interaction element itself. In this case, the common global user's goal (target) for calculator is to allow the user to perform computations. The calculator screen has its own target: to show the operands, numbers and results. The key pad has its own target or goal: to allow the user to introduce the operations. We cannot consider each button as its own target because it would not make sense.

If we consider the distribution of the calculator application across two different platforms (for instance, a situated large display and a mobile device), a possible solution could be to distribute the calculator display on the large situated display and the key pad to the mobile device. The distributed calculator application accomplishes the *decomposition* property since we can divide the main user interface into two UsubI, each one with its own target. The other three properties need a real implementation in order to test whether the distributed application verifies or not each property. A detailed verification of real applications can be found in [10].

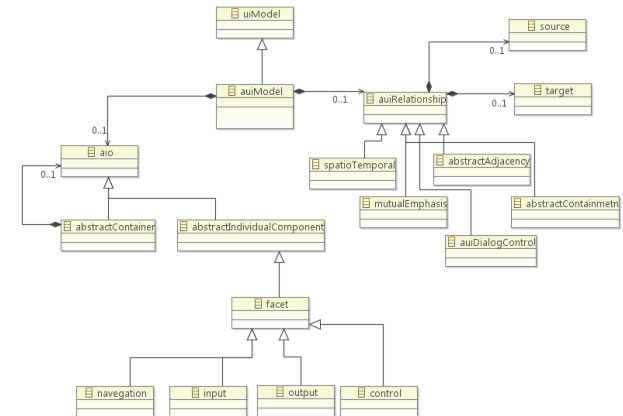
## AN AUI MODEL TO SUPPORT DUIS

In this section, the AUI model that supports our concept of Distributed User Interfaces is presented. As it has been showed in the previous section, the concept of Distributed User Interfaces is based on the concept of User Interface. In fact, according of the perspective of this work, a DUI is a set of interaction elements distributed across different platforms. In order to consider a set of interaction elements as a UI, it is required that all these interaction elements have a common goal (target). This common goal is connected with the user's task, i.e., a UI is the resource to the user to reach the result of the task. This result appears in the model as *target*.

Figure 1 shows the abstract user interface model proposed in this paper. A User Interface is composed of an abstract interaction object (aio) that can be either an interaction element or a sub user interface (uSubI). The interaction element can be used to express input, output, navigation or control actions. In this model, the terms "target" and "sub-target" are present in the model to remark the importance of the common goal which is connatural to the concept of user interface. Figure 2 shows the abstract user interface model that is proposed in the UsiXML V1.8 [9].



**Figure 1. Abstract User Interface with the DUI perspective.**

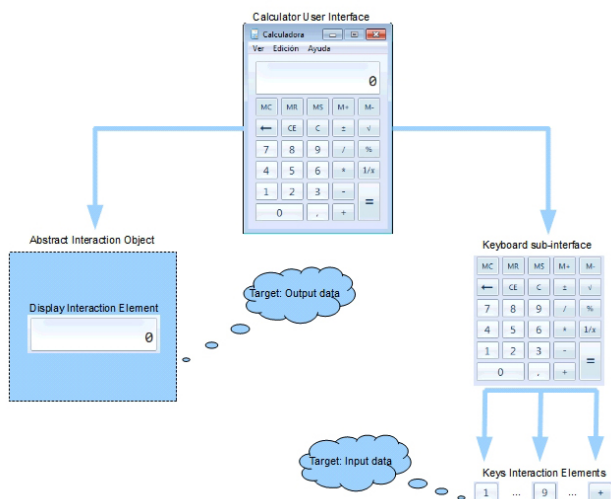


**Figure 2. Abstract User Interface proposed in UsiXML.**

After analyzing the showed models, it must be noted that the term "target" is used in Figure 2 in the sense of user's goal or objective. Target in Figure 2 does not represent user's goal. Comparing both models, the *interaction element* of Figure 1 is equivalent to the *abstractIndividualComponent* of Figure 2, the *uSubI* element is equivalent to the *abstractContainer* and the *uSubI* element is linked with the *subTarget* element to support the explained properties.

There is a significant difference between these two AUI models and the introduction of the *target* and *subTarget* terms in the model. As it has been explained, a DUI is a set of distributed interaction elements that support a common goal (target). Before distributing a UI, it is possible to check if a set of interaction elements can be separated and transferred to other platform (portability and decomposability properties) with a simple verification. We have to test if the selected interaction elements have associated a sub-target or not.





**Figure 3. Decomposing the calculator interface.**

To illustrate this proposal, a simple example based in the calculator is explained. The calculator, as a whole, is a UI in order to allow the user to perform mathematical calculations which is the final objective (“target” in our model). The calculator is composed of an abstract interaction object which also contains an interaction element (the display) and of a user sub-interface (the keyboard). The display has a subtask related to the main goal: show the calculations to the user.

The sub-interface keypad is composed of a set of interaction elements (buttons) with a common sub-objective: Allow user input. The display is framed within the output interaction elements, while each of the keys can be considered as an input element. The “key” interaction elements share a common sub-objective: to allow data input, so it makes no sense to be distributed separately, but framed within its sub-interface since they share the same input subgoal. To separate or distribute interaction elements that do not share a sub-objective linked to the main purpose of the UI can jeopardize the attainment of this objective. Figure 3 shows the discussed concepts.

### CONCLUSIONS AND FUTURE WORK

This work presents a new AUI model to support the Distributed User Interface (DUI) perspective. The introduction of the “target” hierarchy associated to every user interface supports the distribution of user interface elements across different devices maintaining the coherence with the user’s task goal. This new AUI model is based on the characterization of the essential properties of Distributed User Interfaces: decomposability, portability, simultaneity and continuity. We have employed a formal notation to describe these properties. With this formal notation we will be able to build a solid base for developing distributed user interfaces.

This approach will allow us to define an automatic or semi-automatic way to determine whether a user interface can be

distributed or not, depending on the degree of achievement related to the target hierarchy. Provided that all user interfaces and subInterfaces are linked with a target (in the sense of user’s goal), we can easily detect when an isolated (not linked to a user’s goal) interaction element is going to be distributed on a different platform, which can have an adverse effect in the behavior of the distributed user interface.

### REFERENCES

1. Bowen, J. and Reeves, S. Using formal models to design user interfaces: a case study. In *Proceedings of the BCS-HCI'2007* (Swinton, 2007). (2007), pp. 159–166.
2. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J. A Unifying Reference Framework for Multi-Target User Interfaces. *Interacting with Computers* 15, 3 (2003), pp. 289–308.
3. Chi, U. Formal specification of user interfaces: A comparison and evaluation of four axiomatic approaches. *IEEE Transactions on Software Engineering* 11, 8 (August 1985), pp. 671–685.
4. Demeure, A., Calvary, G., Sottet, J.-B., Ganneau, V., and Vanderdonckt, J. A Reference Model for Distributed User Interfaces. In *Proc. of 4<sup>th</sup> Int. Workshop on Task Models and Diagrams for user interface design TAMODIA'2005* (Gdansk, 26-27 September 2005). ACM Press, New York (2005), pp. 79–86.
5. Demeure, A., Sottet, J.S., Calvary, G., Coutaz, J., Ganneau, V., and Vanderdonckt, J. The 4C Reference Model for Distributed User Interfaces. In *Proc. of 4<sup>th</sup> Int. Conf. on Autonomic and Autonomous Systems ICAS'2008* (Gosier, 16-21 March 2008). D. Greenwood, M. Grotke, H. Lutfiyya, M. Popescu (Eds.). IEEE Computer Society Press, Los Alamitos (2008), pp. 61–69.
6. Florins, M., Montero, F., Vanderdonckt, J., Michotte, B., Splitting Rules for Graceful Degradation of User Interfaces. In *Proc. of 8<sup>th</sup> Int. Working Conference on Advanced Visual Interfaces AVI'2006* (Venezia, 23-26 May 2006). ACM Press, New York (2006), pp. 59–66.
7. Gallud, J.A., Villanueva, P.G., Tesoriero, R., Sebastian, G., Molina, S., and Navarrete, A. Gesture-based interaction: Concept map and application scenarios. In *Proceedings of the 3<sup>rd</sup> Int. Conf. on Advances in Human-Oriented and Personalized Mechanisms, Technologies and Services CENTRIC'2010*. IEEE Computer Society Press, Los Alamitos (2010), pp. 28–33.
8. Grolaux, D., Vanderdonckt, J., and Van Roy, P. Attach me, Detach me, Assemble me like You Work. In *Proc. of 10<sup>th</sup> IFIP TC 13 Int. Conf. on Human-Computer Interaction INTERACT'2005* (Rome, 12-16 September 2005). Lecture Notes in Computer Science, vol. 3585. Springer-Verlag, Berlin (2005), pp. 198–212.

9. Limbourg, Q. and Vanderdonckt, J. UsiXML: A User Interface Description Language Supporting Multiple Levels of Independence. In *Engineering Advanced Web Applications*, M. Matera, S. Comai, S. (Eds.). Rinton Press, Paramus (2004), pp. 325-338.
10. Lopez-Espin, J.J. Formal specification of distributed user interfaces. In: Proc. of DUI'2011 Workshop, University of Castilla-La Mancha (2011).
11. Reichart, D.: A.: Task models as basis for requirements engineering and software execution. In *Proc. of TAMODIA'2003*. (2003), pp. 51–589.
12. Souchon, N., and Vanderdonckt, J. A review of XML-compliant user interface description languages. In *Proc. of Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'2003*. J.A. Jorge, N. Jardim Nunes, and J. Falcão e Cunha (Eds.). Lecture Notes in Computer Science, vol. 2844. Springer, Berlin (2003), pp. 391–401.
13. Vanderdonckt, J. Distributed User Interfaces: How to Distribute User Interface Elements across Users, Platforms, and Environments. In *Proc. of XIth Congreso Internacional de Interacción Persona-Ordenador Interacción'2010* (Valencia, 7-10 September 2010). J.L. Garrido, F. Paterno, J. Panach, K. Benghazi, N. Aquino (Eds.). AIPO, Valencia (2010), pp. 3-14.

# A Graphical UIDL Editor for Multimodal Interaction Design Based on SMUIML

Bruno Dumas<sup>1</sup>, Beat Signer<sup>1</sup>, Denis Lalanne<sup>2</sup>

<sup>1</sup>WISE Lab, Vrije Universiteit Brussel  
Pleinlaan, 2 – B-1050 Brussels (Belgium)  
{bdumas, bsigner}@vub.ac.be

<sup>2</sup>DIVA Group, Université de Fribourg  
Boulevard de Pérolles, 90  
CH-1700 Fribourg (Switzerland)  
denis.lalanne@unifr.ch

## ABSTRACT

We present the results of an investigation on software support for the SMUIML multimodal user interaction description language. In particular, we introduce a graphical UIDL editor for the creation of SMUIML scripts. The presented graphical editor is fully based on SMUIML for the representation of the underlying data as well as for the dialogue modelling. Due to the event-centered nature of SMUIML, the representation of the multimodal dialogue modelling in the graphical SMUIML dialogue editor has been realised via a state machine. The editor further offers a real-time graphical debugging tool. Compared to existing multimodal dialogue editors, the SMUIML graphical editor offers a dual graphical and textual editing as well as a number of operators for the temporal combination of modalities.

## Author Keywords

Multimodal Interaction, UIDL, Graphical Editor, SMUIML, HephaisTK.

## General Terms

Design, Human Factors, Theory

## Categories and Subject Descriptors

H.5.2 [Information Interfaces and Presentation]: User Interfaces – *Graphical User Interfaces (GUI), Prototyping, Theory and Methods.*

## INTRODUCTION

Multimodal interfaces aim to improve the communication between humans and machines by making use of concurrent communication channels or modalities. They have been shown to increase comfort and offer better expressivity to users. Nevertheless, multimodal interfaces are difficult to realise due to a number of reasons. First, multimodal interfaces are typically composed of a number of state of the art recognition technologies, such as speech recognition or pattern matching-based gesture recognition. Typically, developers have to master a number of these state of the art recognisers for different modalities in order to create advanced multimodal interfaces. Second, a combination of input for the same modality can lead to ambiguous interpretations based on factors such as the ordering of input events, the delay between events, the context of use or specific user profiles. Fusion algorithms that take adaptation into account are therefore required. Last but not least, multimodal human-machine dialogue modelling is desirable in

order to facilitate the development of complex multimodal interfaces.

The challenges introduced by multimodal interaction design can potentially be addressed by using a modelling language in combination with a multimodal framework and development environment. A multimodal User Interface Description Language (UIDL) forms the key element of such an approach. A UIDL [17] is used to define the behaviour of the multimodal framework, to perform the dialogue modelling and as the underlying format for the GUI development environment. A multimodal user interface description language is typically situated at the Abstract User Interface (AUI) layer. Furthermore, software support for the UIDL is provided for the definition, modelling or interpretation of user interface descriptions.

We present our explorations of such a language-based approach in the context of the Synchronized Multimodal User Interfaces Modelling Language (SMUIML) and the corresponding software support. In particular, we present a graphical UIDL editor for SMUIML and discuss its support for designing multimodal interactions. The graphical editor offers an alternative to the purely text-based editing of scripts in our XML-based language, which is often tedious and can easily lead to errors. This graphical editor furthermore focuses on how to express complex temporal relations between input modalities. We start by discussing related work in the context of modelling languages as well as graphical editors for multimodal interaction design. We then introduce the SMUIML language and some results from our research on the language design for modelling multimodal interaction. This is followed by a description of the different supportive software components for the SMUIML language with a particular focus on the graphical UIDL editor. After an overview of the planned future work, we provide some conclusions.

## RELATED WORK

Over the last decade, there have been a number of formal language approaches for the creation of multimodal interfaces. Some of these approaches are positioned in the context of a *multimodal web*, propagated by the World Wide Web Consortium's (W3C) Multimodal Interaction Activity and its proposed multimodal architecture<sup>1</sup>. This theoretical

---

<sup>1</sup> <http://www.w3.org/TR/mmi-arch/>

framework describes the major components involved in multimodal interaction, as well as potential or existing markup languages to be used to relate these components. Many elements described in this framework, such as the W3C EMMA markup language<sup>2</sup> or modality-focused languages including VoiceXML<sup>3</sup>, EmotionML and InkML<sup>4</sup>, are of practical interest for multimodal HCI practitioners.

The W3C framework inspired Katsurada *et al.* [9] for their work on the XISL XML language. XISL focuses on the synchronisation of multimodal input and output, as well as dialogue flow and transition.

Araki *et al.* [1] propose the Multimodal Interaction Markup Language (MIML) for the definition of multimodal interactions. A key characteristic of MIML is its three-layered description of interaction, focusing on interaction, tasks and platform.

Ladry *et al.* [11] use the Interactive Cooperative Objects (ICO) notation for the description of multimodal interaction. This approach is closely bound to a visual tool enabling the editing and simulation of interactive systems, while being able to monitor system operations at a low level.

Stanciulescu *et al.* [18] followed a transformational approach for the development of multimodal web user interfaces based on UsiXML. Four steps are necessary to get from a generic model to the final user interface. One of the main features of their work is a strong independence from the available input and output channels. A transformational approach is also used in Teresa XML by Paterno *et al.* [13]. DISL [14] was created as a language for specifying a dialogue model which separates multimodal interaction and presentation components from the control model. Finally, at a higher level of modelling, NiMMiT [6] is a graphical notation associated with a language used to express and evaluate multimodal user interaction. An analysis of multimodal interaction modelling languages can also be found in [16].

Graphical editors for the definition of multimodal dialogues can broadly be separated into two families. These two families differ in the way how a dialogue is represented, which is often driven by the underlying architecture. On the one hand, stream-based architectures favour a direct representation of data streams, with building blocks consisting of processing algorithms that are applied to the streams in a sequential manner. In the past few years, there has been a trend for graphical editors for stream-based multimodal architectures. Petshop for ICO [11], Squidy [10] or Skemmi [12] for OpenInterface are examples of these types of graphical editors for stream-based architectures. On the

other hand, event-driven architectures result in a state machine-based representation of the multimodal human-machine dialogue. In this category, fewer examples exist for the representation of multimodal interaction, the most prominent one being IMBuilder from Bourguet [3]. Note that the graphical editors introduced in this section have all been built from scratch and they are not based on a previously defined formal language, with Petshop for ICO forming the only exception.

## THE SMUIML LANGUAGE

SMUIML stands for *Synchronized Multimodal User Interaction Modelling Language*. As the name implies, SMUIML aims to offer developers a language to describe multimodal interaction and define the used modalities in an easy-to-read and expressive way. The language can further be used to describe the recognisers associated with a given modality, the human-machine dialogue modelling, the various events associated with these dialogues and the way these different events can be temporally synchronised<sup>5</sup>. SMUIML was designed to be as simple as possible and is targeting usability. In order to minimise the verbosity of SMUIML, we decided not to rely on existing standard multimodal interaction languages.

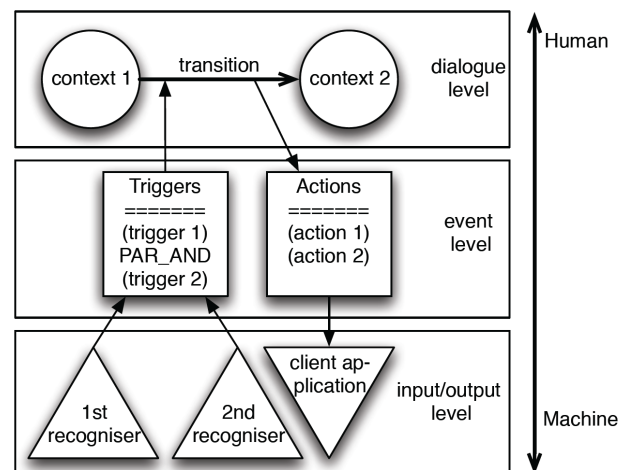


Figure 1. The three levels of SMUIML.

The SMUIML language is divided into the three abstraction layers shown in Figure 1. The lowest level details the different modalities which are then used in the context of an application, as well as the particular recognisers to be used to access the different modalities. The middle level addresses input and output events. Input events are called *triggers* and output events *actions*. Triggers are defined per modality which means that they are not directly bound to specific recognisers and they can express different ways to

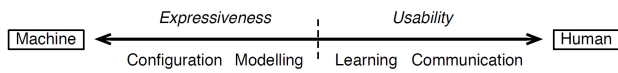
<sup>2</sup> <http://www.w3.org/TR/emma/>

<sup>3</sup> <http://www.w3.org/TR/voicexml20/>

<sup>4</sup> <http://www.w3.org/TR/InkML/>

<sup>5</sup> An XML Schema as well as some SMUIML examples can be found at: <http://sourceforge.net/projects/hephaistk/>

trigger a particular event. For example, a speech trigger can be defined in such a way that the words “clear”, “erase” and “delete” will all lead to the same event. Actions are the messages that the framework sends to the client application. The top level of abstraction describes the actual human-machine dialogue by means of defining the contexts of use and interweaving the different input events and output messages between those contexts. The resulting human-machine dialogue description is a series of “contexts of use”, with transitions between these different contexts. Therefore, the description of the multimodal human-machine dialogue in SMUIML has an implicit representation as a state machine, similar to Bourguet’s IMBuilder [3]. The combination of modalities is defined based on the CARE properties [5] as well as on the (non-)sequentiality of input triggers. As shown in Listing 1, the three abstraction levels are directly reflected in the basic structure of the language.



**Figure 2. Four modelling language purposes (from machine-oriented to human-oriented) with respect to expressiveness and usability.**

The spectrum of multimodal dialogue description language users, on a scale from usability to expressiveness, was presented in [8]. Through various workshops, informal discussions with colleagues and students and a study of the current state of the art, we envisioned three types of approaches for a description language: a highly formal language approach that perfectly fits for configuring a tool, a less formal language approach which is good for communicating the details of an application and a “middle” approach focussing on the modelling. Along these three approaches, a formal language can also be used as a learning tool (see Figure 2) helping teachers in communicating the features of a particular application domain to their students.

In [8] we presented 9 guidelines for a multimodal description language. These guidelines should be used as design tools or as language analysis criteria:

- Abstraction levels
- Modelling the human-machine dialogue
- Adaptability to context and user (input and output)
- Control over fusion mechanism
- Control over time synchronicity
- Error handling
- Event management
- Input and output sources representation
- Finding the right balance between usability and expressiveness

## SOFTWARE SUPPORT FOR SMUIML

SMUIML enables the definition of a full model of multimodal human-machine events and dialogues by providing modelling capabilities as well as a reflection basis. However, the language shows its true potential when linked to a range of different supportive software solutions. In the following, we briefly introduce the software support within SMUIML for interpretation and then discuss the latest software addition in the form of a graphical editor for designing multimodal human-machine dialogues.

### The HephaisTK Framework

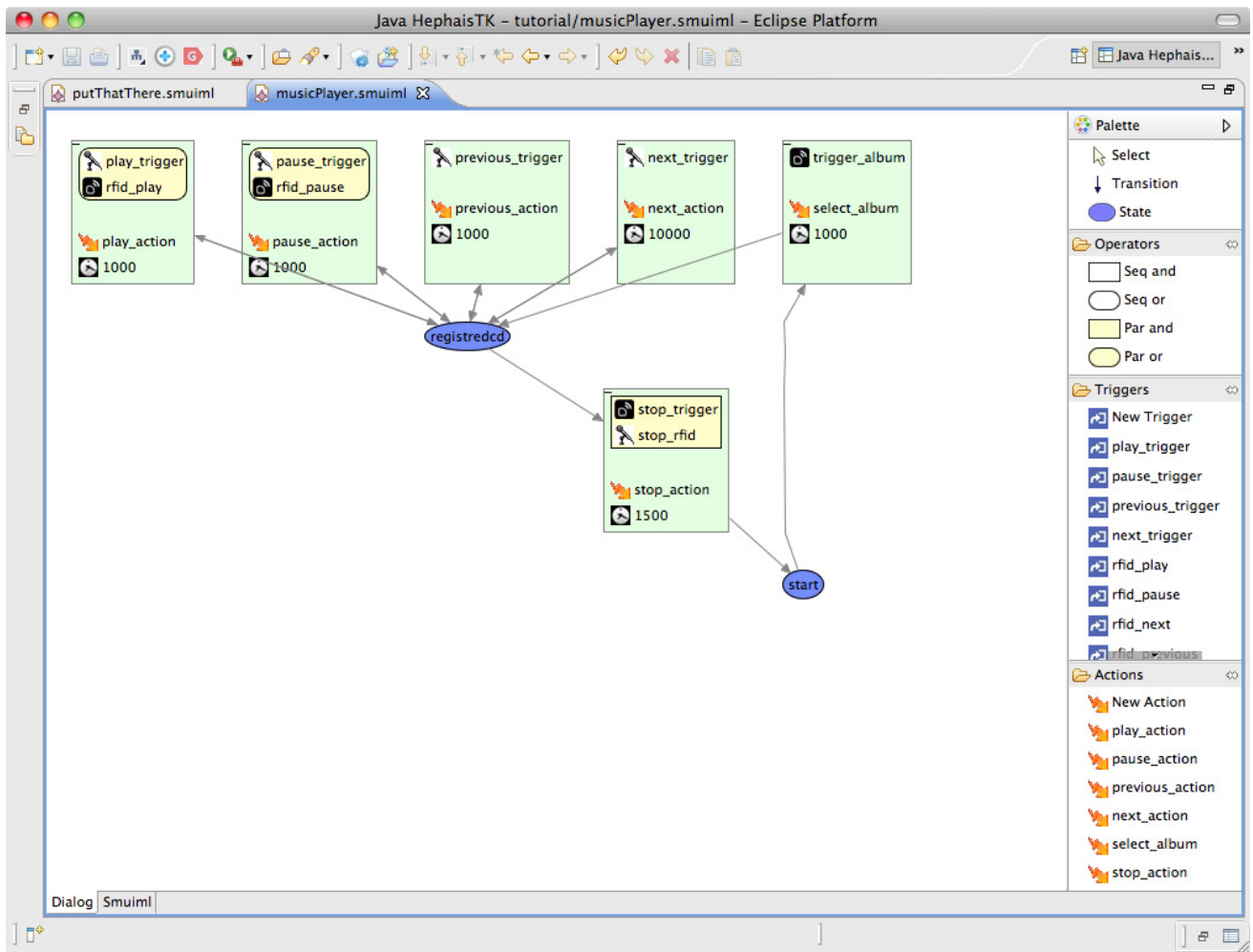
The HephaisTK framework which supports the creation of multimodal interfaces based on the SMUIML scripting language has been developed in our research lab. A description created in SMUIML, with the structure shown in Listing 1, is used to configure the HephaisTK framework. The `<recognizers>` part indicates which recognisers have to be loaded by the framework. It further provides some high-level parameters such as whether a speech recogniser is able to recognise different languages. The `<triggers>` are directly bound to the different fusion algorithms provided by HephaisTK. The `<actions>` part defines the semantics to be used when communicating fusion results to a client application. Last but not least, the SMUIML `<dialog>` part is used for a number of specific goals in HephaisTK.

```
<?xml version="1.0" encoding="UTF-8"?>
<smuiml>
  <integration_desc client="client app">
    <recognizers>
      <!-- ... -->
    </recognizers>
    <triggers>
      <!-- ... -->
    </triggers>
    <actions>
      <!-- ... -->
    </actions>
    <dialog>
      <!-- ... -->
    </dialog>
  </integration_desc>
</smuiml>
```

**Listing 1. Basic layout of a SMUIML script.**

First and foremost, by providing a description of the human-machine dialogue flow, the HephaisTK Dialog-Manager agent stays in a consistent state with the client application. The clear separation of the SMUIML `<dialog>` into transitions and contexts allows the different triggers to be enabled or disabled depending of the current context. Since only a subset of triggers has to be considered in a given context, the load on the recognisers is reduced and the overall recognition rate is improved.





**Figure 3. The SMUIML graphical editor with an example dialogue defining the behaviour of a music player application.**

The `<dialog>` part of SMUIML also helps with the instantiation of the different fusion algorithms present in HephaistK. In the case of the Hidden Markov Model-based fusion algorithm that is integrated in HephaistK, the definition of the human-machine dialogue in SMUIML is also used to generate a set of all expected trigger input sequences. This set of expected sequences is then injected into a series of Hidden Markov Models (one per context of use) in order to have the fusion engine ready to be used when launching the HephaistK framework.

The SMUIML language is applied at multiple levels in the context of the HephaistK framework: at the multimodal dialogue description level, at the recogniser launch and parameterisation level as well as the fusion engine instantiation level. SMUIML is typically used during the later stages of multimodal interface development, including the *system design* and *runtime* stages. Note that it is out of the scope of this paper to provide a full description of HephaistK but further details can be found in [7].

#### *The SMUIML Graphical Editor*

The SMUIML language is derived from the XML metalanguage and a standard text editor is sufficient for creating SMUIML documents. Even if the language has been proven to be expressive in a qualitative study [8], the editing of “raw” XML documents can easily lead to errors that are only identified when interpreting a SMUIML script at runtime. Other issues with the text-based editing of SMUIML scripts include the lack of an explicit representation of the relationships between different elements as well as the difficulty to produce and maintain an accurate mental model of complex dialogue scenarios. Furthermore, the necessity of having to learn a new language may represent a major challenge for some users. In order to overcome these shortcomings, we have developed a graphical editor for the definition of new SMUIML scripts.

The goal of our SMUIML graphical editor was to provide developers, who are not fully proficient with multimodal interfaces, a usable and expressive tool for creating



SMUIML scripts. The dialogue editor offers a graphical representation of SMUIML-encoded multimodal human-machine dialogues. Furthermore, it supports the creation of sets of actions and triggers and can be used to generate a Java configuration with all the calls related to the SMUIML script. The graphical representation of a multimodal dialogue follows the SMUIML logic presented in the previous section. The SMUIML graphical editor has been created based on the Eclipse open development platform (<http://www.eclipse.org>). Eclipse is widely used among development teams and provides a set of well-known interface elements. The SMUIML graphical tool itself was developed using the Graphical Editing Framework (GEF - <http://www.eclipse.org/gef/>) and the Eclipse Modeling Framework (EMF - <http://www.eclipse.org/modeling/emf/>).

The main window of the graphical editor is shown in Figure 3. The central part of the tool is dedicated to the actual dialogue representation. As stated earlier, the multimodal human-machine dialogue in SMUIML is represented via a state machine. A graphical representation of this state machine is used to depict the multimodal dialogue in the graphical editor. Note that the editor also provides access to a textual version of the SMUIML script that is currently edited. Any changes that are done either in the graphical or the textual representation are immediately reflected in the other representation. For both, the graphical and textual representation, there exists real-time error checking.

On the right-hand side of the window are a set of toolboxes and most of them are related to the different parts of a typical SMUIML file. The Palette toolbox presents the basic building blocks for creating the dialogue state machine, in particular states and transitions. The selection tool also forms part of the Palette toolbox. The Operators toolbox offers some operators to combine different modalities as defined in the SMUIML specification. These operators are tightly linked to the CARE properties [6]. Seq and corresponds to sequential-constrained complementarity, Par and to sequential-unconstrained complementarity, the Seq or operator to equivalence and Par or to redundancy. The next toolbox is devoted to input triggers and contains a list of all triggers defined for a given application, as well as a New trigger button to create new triggers. Last but not least, the Actions toolbox lists all actions that have been defined for a given application and also provides a New action button. Triggers and actions are added to these toolboxes when they are defined as part of a multimodal dialogue.

Figure 4 shows the graphical representation of Bolt’s “put that there” example [2] in the graphical editor with states (contexts) visualised as blue ellipses. The corresponding textual SMUIML specification is shown in Listing 2. Based on the actions taken by users, HephaisTK might stay in the same context or switch to another context of use. In the “put that there” example, there is only as single start

context with a transition starting and pointing to it. This transition contains the overall description of the “put that there” action. It asks for five different input triggers in order that the action will be fired. Namely, three speech triggers (“put”, “that” and “there”) as well as two pointing event triggers. Furthermore, three temporal combination operators are used in this example. The main transition uses a Seq and operator asking for a “put” speech trigger to be followed by a “that” and “there” sub-event. The two sub-events use a Par and combination operator, meaning that there should be speech and pointing triggers but without any sequential constraint. This implies that a user can perform the commands in different orders, such as “put that” [point1] [point2] “there” or “put” [point1] “that there” [point2] and both sequences will be correctly recognised. Finally, the transition specifies a time window of 1500 milliseconds for the whole command as well as an action (message to be sent the client application) to be performed if the command has been successfully recognised. In our example, the transition then proceeds to the same start context it originated from.

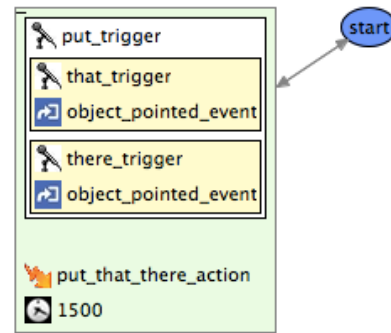
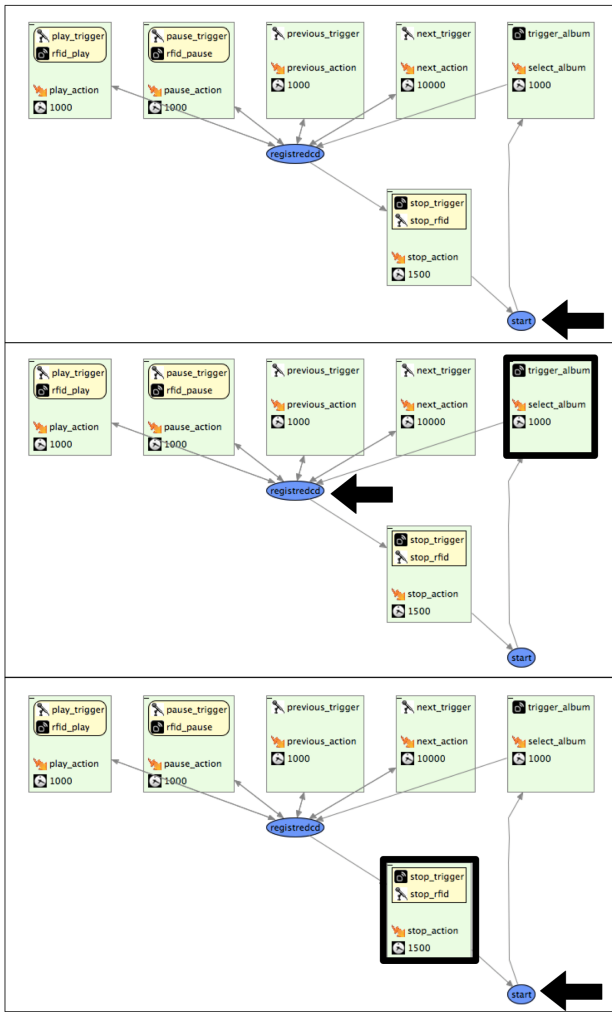


Figure 4. Graphical description of “put that there”.

```
<context name="start">
<transition leadtime="1500">
  <seq and>
    <trigger name="put trigger"/>
    <transition>
      <par and>
        <trigger name="that trigger"/>
        <trigger name="object pointed event"/>
      </par and>
    </transition>
    <transition>
      <par and>
        <trigger name="there trigger"/>
        <trigger name="object pointed event"/>
      </par and>
    </transition>
  </seq and>
  <result action="put that there action"/>
  <result context="start">
</transition>
</context>
```

Listing 2. SMUIML description of the “put that there” example.



**Figure 5. The graphical debugging tool with three different steps going from the start context to registeredcd and back again.**

The SMUIML graphical editor has been presented to two expert users in order to achieve a small expert review. This review by experts lead to a number of changes to improve the editor’s usability. The modality of each trigger is now indicated by means of an icon. The start context which is the only mandatory context in a given SMUIML script is visualised in a slightly different colour to denote its special status compared to other contexts. Finally, users have the possibility to change the colour of connections, contexts or transitions in order to best suit their preferences.

The graphical editor also contains an integrated debugging tool. This debugging tool is launched with the client application and provides a real-time visualisation of the context the HephaisTK framework is currently in. It also highlights the transition leading from the previous context to the current one. In the example illustrated in Figure 5, the application starts in the start context. A Radio Frequency Identification (RFID) reader that is connected to the framework

detects a tagged music album and transmits the information. Based on the trigger\_album trigger, a transition is fired and the application moves to the registeredcd state and starts playing the music album. The user then executes a stop command and, at the same time, holds a “stop” labelled RFID tag close to the RFID reader. This simultaneous action fires the transition going from the registeredcd context back to the start context. As illustrated in this example, the graphical debugging tool allows developers to visually analyse the application behaviour in real-time.

## FUTURE WORK

While the presented SMUIML graphical editor looks quite promising and offers some features not available in other graphical editors for multimodal interfaces, we plan to perform a detailed evaluation of the presented solution in the near future. First, we are going to evaluate the usability of the presented graphical editor by asking developers to express a number of multimodal interaction use cases via the tool. In addition, we plan to evaluate the expressiveness of the presented approach. It is not enough to guarantee an effective and simple definition of multimodal interactions based on the graphical editor. We also have to ensure that the editor and the underlying SMUIML language are expressive enough to describe multimodal interactions of arbitrary complexity. This study could be a starting point for tackling a more general question: *to what extent do finite state machine-based approaches or event stream-based approaches best represent multimodal human-machine dialogues?*

Another important future direction is to support the flexible adaptation of multimodal interfaces [19,20]. The idea is to no longer have a fixed combination of modalities, but rather provide a context-dependent adaptation of multimodal user interfaces. This can either be achieved by extending the SMUIML language with the necessary concepts or by introducing another language for the adaptation of the multimodal interaction. In this view, the abstract user interface definition would rely on SMUIML while the concrete, context-dependant user interface specification would require the definition of a new language. The final user interface could be realised by HephaisTK [4].

This new language for flexible multimodal interface adaptation could then be used to provide innovative document interfaces. Today’s document formats often provide no access to specific semantic subparts or embedded media types [15]. However, if we would be able to get access to these document subparts, specific embedded media types could be associated with different modalities of interaction. Within the MobiCraNT<sup>6</sup> project we are currently investigating innovative mobile cross-media applications. As part of this research effort, we are developing a new fluid cross-

<sup>6</sup> <http://soft.vub.ac.be/mobicrant/>

media document model and investigate how SMUIML, in combination with a context-dependant user interface specification language, could be used to provide multimodal access to such a fluid document model.

## CONCLUSION

We have presented our exploration on software support for multimodal UIDL based on the SMUIML multimodal dialogue modelling language. Thereby, we focussed on two particular software components: the HephaisTK framework which is used to interpret the SMUIML language (at the final, runtime stage) and the SMUIML graphical editor for the graphical design of multimodal interaction dialogues (at the system design stage). The SMUIML graphical editor aims to provide a user-friendly way to create multimodal applications based on HephaisTK and SMUIML. Compared to other graphical dialogue editors, our solution supports temporal constraints and a number of operators for the combination of multiple modalities. While these concepts already form part of the underlying SMUIML language, the graphical editor makes these concepts accessible via a user-friendly interface. Users further have the possibility to freely switch between the graphical and textual dialogue representation. The presented SMUIML graphical editor further addresses a number of usability-oriented issues such as automatic layouting, the clear identification of input modalities via specific icons as well as the possibility to customise various features of the graphical editor. Last but not least, the SMUIML graphical editor offers an integrated debugging tool supporting developers in analysing the real-time application behaviour.

## ACKNOWLEDGMENTS

The authors wish to thank Saïd Mechkour for his work on the SMUIML graphical editor. The work on HephaisTK and SMUIML has been funded by the Hasler Foundation in the context of the MeModules project and by the Swiss National Center of Competence in Research on Interactive Multimodal Information Management via the NCCR IM2 project. Bruno Dumas is supported by MobiCraNT, a project forming part of the Strategic Platforms programme by the Brussels Institute for Research and Innovation (Innoviris).

## REFERENCES

1. Araki, M., and Tachibana, K. Multimodal Dialog Description Language for Rapid System Development. In *Proc. of the 7<sup>th</sup> SIGdial Workshop on Discourse and Dialogue* (Sydney, July 2006), pp. 109–116.
2. Bolt, R. A. “Put-that-there”: Voice and Gesture at the Graphics Interface. In *Proc. of the 7<sup>th</sup> Annual Conference on Computer Graphics and Interactive Techniques SIGGRAPH’80* (Seattle, July 1980), pp. 262–270.
3. Bourguet, M.-L. A Toolkit for Creating and Testing-Multimodal Interface Designs. In *Adjunct Proc. of the 15th Annual Symposium on User Interface Software and Technology UIST’2002* (Paris, October 2002).
4. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J. A Unifying Reference Framework for Multi-Target User Interfaces. *Interacting with Computers* 15, 3 (2003), pp. 289–308.
5. Coutaz, J., Nigay, L., Salber, D., Blandford, A., May, J., and Young, R.M. Four Easy Pieces for Assessing the Usability of Multimodal Interaction: The CARE Properties. In *Proc. of the 5<sup>th</sup> Int. Conf. on Human-Computer Interaction Interact’1995* (Lillehammer, June 1995).
6. De Boeck, J., Vanacken, D., Raymaekers, C., and Coninx, K. High-Level Modeling of Multimodal Interaction Techniques Using NiMMiT. *Journal of Virtual Reality and Broadcasting* 4, 2 (September 2007).
7. Dumas, B., Lalanne, D., and Ingold, R. HephaisTK: A Toolkit for Rapid Prototyping of Multimodal Interfaces. In *Proc. of 11th International Conference on Multimodal Interfaces ICMI’2009* (Cambridge, November 2009). ACM Press, New York (2009), pp. 231–232.
8. Dumas, B., Lalanne, D., and Ingold, R. Description Languages for Multimodal Interaction: A Set of Guidelines and its Illustration with SMUIML. Special Issue on the Challenges of Engineering Multimodal Interaction, *Journal on Multimodal User Interfaces* 3, 3 (February 2010), pp. 237–247.
9. Katsurada, K., Nakamura, Y., Yamada, H. and Nitta, T. XISL: A Language for Describing Multimodal Interaction Scenarios. In *Proc. of the 5th Int. Conf. on Multimodal Interfaces ICMI’2003* (Vancouver, Canada, November 2003). ACM Press, New York (2003), pp. 281–284.
10. König, W.A., Rädle, R., and Reiterer, H. Squidy: A Zoomable Design Environment for Natural User Interfaces. In *Proc. of the 27<sup>th</sup> Int. Conf. on Human Factors in Computing Systems CHI’2009* (Boston, April 2009).
11. Ladry, J.-F., Palanque, P., Basnyat, S., Barboni, E., and Navarre, D. Dealing with Reliability and Evolvability in Description Techniques for Next Generation User Interfaces. In *Proc. of the 26<sup>th</sup> ACM Int. Conf. on Human Factors in Computer Systems CHI’2008* (Florence, April 2008). ACM Press, New York (2008).
12. Lawson, J.-Y. L., Al-Akkad, A.-A., Vanderdonckt, J., and Macq, B. An Open Source Workbench for Prototyping Multimodal Interactions Based on Off-the-Shelf Heterogeneous Components. In *Proc. of the 1<sup>st</sup> ACM Symposium on Engineering Interactive Computing Systems EICS’2009* (Pittsburgh, July 2009). ACM Press, New York (2009), pp. 245–254.
13. Paterno, F., Santoro, C., Mäntyjärvi, J., Mori, G., and Sansone, S. Authoring Pervasive Multimodal User Interfaces. *International Journal of Web Engineering and Technology* 4, 2 (2008), pp. 235–261.

14. Schaefer, R., Bleul, S., and Mueller, W. Dialog Modeling for Multiple Devices and Multiple Interaction Modalities. In *Proc. of the 5th International Workshop on Task Models and Diagrams for User Interface Design TAMODIA'2006* (Hasselt, October 2006). Springer-Verlag, Berlin (2006), pp. 39–53.
15. Signer, B. What Is Wrong with Digital Documents? A Conceptual Model for Structural Cross-Media Content Composition and Reuse. In *Proc. of the 29<sup>th</sup> International Conference on Conceptual Modeling ER'2010* (Vancouver, November 2010). Lecture Notes in Computer Science, Springer-Verlag, Berlin (2010), pp. 391–404.
16. Sottet, J.-S., Calvary, G., Coutaz, J., Favre, J.-M., Vanderdonckt, J., Stanciulescu, A., and Lepreux, S. A Language Perspective on the Development of Plastic Multimodal User Interfaces. *Journal on Multimodal User Interfaces 1*, (2007), pp. 1–12.
17. Souchon, N., and Vanderdonckt, J. A Review of XML-Compliant User Interface Description Languages. In *Proc. of 10th Int. Conf. on Design, Specification, and Verification of Interactive Systems DSV-IS'2003* (Madeira, 4-6 June 2003). J. Jorge, N.J. Nunes, J. Cunha (Eds.). Lecture Notes in Computer Science, vol. 2844. Springer-Verlag, Berlin (2003), pp. 377–391.
18. Stanciulescu, A., Limbourg, Q., Vanderdonckt, J., Michotte, B., and Montero, F. A Transformational Approach for Multimodal Web User Interfaces Based on UsiXML. In *Proc. of the 7th International Conference on Multimodal Interfaces ICMI'2005* (Trento, October 2005). ACM Press, New York (2005), pp. 259–266.
19. Vanderdonckt, J., Calvary, G., Coutaz, J., and Stanciulescu, A. Multimodality for Plastic User Interfaces: Models, Methods, and Principles. In *Multimodal User Interfaces, Signals and Communication Technology*, Springer, Berlin (2008), pp. 61–84.
20. Vanderdonckt, J. Model-Driven Engineering of User Interfaces: Promises, Successes, and Failures. In *Proc. of 5<sup>th</sup> Annual Romanian Conf. on Human-Computer Interaction ROCHI'2008* (Iasi, September 18-19, 2008), S. Buraga, I. Juvina (Eds.). Matrix ROM, Bucarest (2008), pp. 1–10.

# FlexiXML, A Portable User Interface Rendering Engine for UsiXML

José Creissac Campos<sup>1</sup>, Sandrine Alves Mendes<sup>2</sup>

<sup>1</sup> Departamento de Informática/CCTC  
Universidade do Minho  
Campus de Gualtar, 4710-057 Braga, Portugal  
jose.campos@di.uminho.pt

<sup>2</sup> ALERT Life Sciences Computing, S.A.  
Rua Daciano Baptista Marques, 245  
4400-617 Vila Nova de Gaia, Portugal  
sandrine.mendes@alert.pt

## ABSTRACT

A considerable amount of effort in software development is dedicated to the user interaction layer. Given the complexity inherent to the development of this layer, it is important to be able to analyse the concepts and ideas being used in the development of a given user interface. This analysis should be performed as early as possible. Model-based user interface development provides a solution to this problem by providing developers with tools that enable both modeling, and reasoning about, user interfaces at different levels of abstraction. Of particular interest here, is the possibility of animating the models to generate actual user interfaces. This paper describes FlexiXML, a tool that performs the rendering and animation of user interfaces described in the UsiXML modeling language.

## Author Keywords

Tool Support, User Interface Description Languages (UIDLs), UsiXML.

## General Terms

Design, Human Factors, Theory

## Categories and Subject Descriptors

D.2.2 [Software Engineering]: Design Tools and Techniques – *User interfaces*. H.5.2 [Information Interfaces and Presentation]: User Interfaces – *User-centered design*.

## INTRODUCTION

User interface development is a complex process. In the long run, the success of an interactive system hinges on having considered both the users of the system appropriately, as well as the technologies for its development. Model-based development provides a solution to manage such complexity. In this paradigm, declarative models are created that range from abstract concerns with domain and task knowledge, through the design of the intended dialog, down to the concrete interaction styles and execution platforms to be used.

A model-based approach encourages a more sustainable development process. In particular it allows capturing a rigorous description of the design, thus facilitating the construction of prototypes via the animation of the models. In turn, the development of prototypes fosters a better understanding of a systems design, and facilitates the participation of users in the development process. These prototypes

can be repeatedly tested and adapted to the users' needs. Prototyping and testing does not guarantee the absence of errors, but minimizes the likelihood of such errors in more advanced stages of the project, making it possible to assess the users' reactions to the interface design under development.

This paper describes FlexiXML, a new user interfaces rendering tool for the UsiXML modeling language. FlexiXML acts as a renderer and animator, enabling users to interact with an interface expressed in UsiXML. The rest of the paper is organized as follows. First related work is discussed. Then the UsiXML modeling language is described. Next the technology used to implement the FlexiXML tool is described, followed a description of the tool itself, and an example of application. The paper ends with conclusions and an outline of future work.

## RELATED WORK

As stated above, the core language for this project is UsiXML (User Interface eXtensible Markup Language) [14]. UsiXML is a User Interface Description Language (UIDL), and will be discussed in the next section. Besides UsiXML, other UIDLs [17] dealing with different aspects of a graphical interface have been put forward. Examples include: AUIML (Abstract User Interface Markup Language) [3], that focuses on enabling user interfaces to be deployed to different device types; UIML (User Interface Markup Language) [2], a markup language standardized by OASIS; XIML (eXtensible Interface Markup Language) [1]; IBM's WSXL (Web Service Experience Language) [2]; or Mozilla's XUL (XML User Interface Language) [9]. For an extensive survey of XML-compliant languages for user interface description see the paper by Guerrero *et al.* [12] and [17].

The selection of UsiXML was a result of its broad scope, which allows the specification of the different models needed during interface development, and a result of the fact that there is an active community supporting the development of the language. The UsiXML language allows for the interface specification to be made at different levels of abstraction, and provides transformation mechanisms between them. Of particular interest in this context are models that deal with the design of the concrete user interface (the components that make up the interface, their lay-

out, and behaviour), and the execution context.

To support user interface modeling and manipulation of the models, UsiXML has a set of tools, which can be divided into two categories:

- *Editors* – Tools for the creation of UsiXML descriptions. The way the descriptions are created varies with the type of tool. Examples of this type of tool include, SketichXML [5] that uses as input hand-drawn interfaces (sketches), GrafiXML [8] where the interface is created by direct manipulation of components on the screen, VisiXML [5] where the interface design is done in Microsoft Visio, among other tools.
- *Interpreters* – Engines to generate graphical user interfaces described in UsiXML. Each interpreter generates interfaces with a set of specific characteristics. Examples of this type of tool include: FlashiXML [15] that generates vectorial interfaces, HaptiXML [7] that generates graphical user interfaces in 3D with interaction by touch, QtkiXML [6] setting interfaces for multiple platforms, InterpiXML [15] that allows simultaneous interpretation of various UsiXML descriptions.

Although the UsiXML community already has several tools focusing on different aspects of the language, ongoing development of the language means that it does not have an updated animation tool to facilitate a process of rapid prototyping and analysis. The tool that more closely approximates the desired outcome is FlashiXML. However this tool is not compatible with the current version of UsiXML, and not easily upgradable. Currently UsiXML is in version 1.8, and FlashiXML is compatible with version 1.4.6 only.

This project aims to create a completely new interpreter tool that supports the current version of UsiXML, providing a number of additional features when compared with FlashiXML. Specifically, the main goals are to provide a tool that:

- Is capable of both interpreting the latest version of UsiXML, and adapting to new versions;
- Is implemented in a platform independent technology enabling web access.

Additionally, we intend to take the opportunity to create a generic and expandable application, as exposed in the FlexiXML Platform section.

Regarding the technologies to implement the tool, the choice was made to use the Adobe Flex software development toolkit. The use of Flex and ActionScript 3 provides a robust solution, with better performance, and a more modular architecture, than what can be achieved by using Flash and ActionScript2.

The use of the Adobe's AIR runtime environment allows FlexiXML to be relatively independent of the computing platform where it runs. The only requirement is that there is an AIR runtime for the target platform.

## USiXML

UsiXML (USeR Interface eXtensible Markup Language) is an XML-compliant markup language for user interfaces (a XML-based User Interface Development Language – UIDL) that describes a user interface independently of programming language, computing platform and working environment. This UIDL enables description of a user interface at a high level of abstraction without requiring programming skills, enabling analysts, designers, programmers and end-user to use it during the development life cycle [18].

### Levels of abstraction

UsiXML allows for user interfaces to be modelled at several levels of abstraction. The language is inspired by the Cameleon Reference Framework (Context Aware Modeling for Enabling and Leveraging Effective interaction) [13], which defines development stages for interactive applications with multiple contexts. In the current context the relevant layers are the Concrete User Interface (CUI) layer, comprising specifications of the user interface (in terms of interaction objects and their relationships) which are independent of the computing platform; and the Final User Interface (FUI) layer, the interface that can be executed or interpreted in a context of use (a specific computing platform and a set of specific devices, using specific interaction objects). A rendering engine performs a transformation from a CUI to a FUI. That is, a Reification transformation – converting an interface model at a more abstract level to a more concrete one.

### Relevant models

The UsiXML language consists of a number of models that together address the needs of the framework described in the previous section. A detailed description of the different UsiXML models falls outside the scope of this paper. However, in order to contextualize the work, a brief presentation of the main models interpreted by FlexiXML will be made.

#### *UiModel – User Interface model*

The UiModel is the core model of the graphical interface specification. This component contains the common features to all models, such as version, author, or creation date, among others.

The UiModel aggregates a number of other models. In the specific case of the FlexiXML interpreter only the following are required: the concrete user interface model (cuiModel), the context model (contextModel) and the resources model (resourceModel). These are the models that contain information needed for constructing the final user interface.

#### *CuiModel – Concrete User Interface model*

The CuiModel specifies the concrete user interface as described previously. It defines the objects that make up the graphical interface (CIO - Concrete Interaction Objects), and the relations between them (CUIR - Concrete User In-



terface Relationships). Particularly relevant are graphical transitions, which enable the specification of control flow.

Since FlexiXML has a fixed platform (the Air runtime), only the Environment and Stereotype features can vary. At this stage Stereotype was considered the relevant feature. In the stereotype, the feature that has more interest is the language. Using it, it is possible to define the language in which the generated application will be viewed. A UsiXML model can define more than one context, allowing user to view the application in different languages.

#### *ResourceModel – Resources Model*

The ResourceModel defines values for the attributes of the graphical objects that depend on the context (e.g. location, language, culture, etc.). This model contains all kinds of content that can be attributed to an interaction object (content, tooltip, etc.).

#### **ADOBE FLEX**

As stated above the FlexiXML has been developed using the Adobe Flex<sup>7</sup> software development toolkit (hence the name *FlexiXML*).

Adobe Flex (or simply Flex) is an open-source framework for the development of cross platform Rich Internet Applications. The framework provides a library of components to build graphical user interfaces. These components can be extended to build new ones. User interface layout is declaratively defined using MXML, an XML-based user interface markup language. MXML also supports a predefined set of behaviors, such as transitions between elements. For more complex control logic, the object oriented Action Script 3 language is used. Action Script is a dialect of ECMAScript, and as such it shares its syntax and semantics with JavaScript.

Applications developed in Flex are compatible (i.e. can run) with all main browsers and operating systems. They can be run on a browser resorting to the Adobe Flash® Player plug-in, or directly on the desktop with the cross-platform Adobe AIR runtime environment.

Adobe AIR<sup>8</sup> is a cross platform runtime environment that enables Rich Internet Application to be run on the desktop to simulate native applications. Properly packages and signed applications will gain access to local resources in the host machine, bringing them closer to the flexibility and power of native applications. Runtime environments are available for most mainstream operating systems, including mobile operating systems such as Android and iOS. This enables an application developed in Flex to be run in a multitude of different platforms as either a Web or desktop application.

Besides the discussion above, other motivation to choose Adobe Flex as the implementation technology included:

- The fact that it enables access to a number of different data sources (different databases, XML files, etc.);
- The fact that it supports changing the user interface at runtime;
- The fact that it provides better performance when compared with previous versions of Flash.

#### **USIXML vs. FLEX**

One of the problems with tools such as FlashiXML is that the mapping between UsiXML and the implementation technology is hardcoded in the tool. This makes keeping the tool up-to-date with the language difficult. To avoid this pitfall we have opted for a configuration based approach when designing FlexiXML. Instead of hard coding the mapping in the tool, a configuration file is used to explicitly provide this mapping. This creates a decoupling between the tool implementation and the specific version of the language being used with the goal of easing maintenance and upgrades. It also should enable FlexiXML to support other markup language than UsiXML.

Three types of mapping were identified as being needed. A mapping between the user interface elements of the markup language and the widgets in the implementation technology; a mapping between the events in the markup language and the events supported by FlexiXML (in this case those supported by Action Script 3); and finally a mapping between window transitions in UsiXML and animation effects in Flex.

Regarding the first mapping, Figure 1 illustrates the mapping of four elements: windows, buttons, text components and checkboxes. In each case, a class in the Flex implementation is identified. In the first three cases, special purpose widgets, derived from the native widgets, are used. In the last case, a native widget is used.

```
<ComponentsMapper>
  <window component =
    "Classes.Components.FlexiXMLWindow"/>
  <button component =
    "Classes.Components.FlexiXMLButton"/>
  <textComponent component =
    "Classes.Components.FlexiXMLText"/>
  <checkBox component = "mx.controls.CheckBox"/>
</ComponentsMapper>
```

**Figure 1. Components mapping.**

Regarding the second mapping, Figure 2 illustrates how UsiXML events are mapped to events in the FUI. In this case four events are mapped: release, depress, rollOver and rollOut. These vents are mapped to corresponding mouse events: mouse up, mouse down, mouse over, and mouse out.

<sup>7</sup> <http://www.adobe.com/products/flex/> (visited 14/07/2011)

<sup>8</sup> <http://www.adobe.com/products/air/> (visited 14/07/2011)

```

<EventsMapper>
  <release event = "mouseUp"/>
  <depress event = "mouseDown"/>
  <rollOver event = "mouseOver"/>
  <rollOut event = "mouseOut"/>
</EventsMapper>

```

**Figure 2. Events mapping.**

Finally, regarding the third and last mapping, Figure 3 examples of mapping between window transitions in UsiXML, and animation effects in Flex. Hence, box in/out transitions are mapped to zoom effects, fade in/out transitions are mapped to corresponding fade effects, and close/open transitions are mapped to corresponding visibility effects.

Using these mappings, it becomes possible to easy tailor how the user interface is generated. For example, we could change how a depress or release event is detected at the interface, or specify that a fade event should be mapped to a zoom animation.

```

<ActionsMapper>
  <transition>
    <boxOut effect="Classes.Animation.Zoom"
           direction = "OUT"/>
    <boxIn effect="Classes.Animation.Zoom"
          direction = "IN"/>
    <fadeOut effect="Classes.Animation.Fade"
            direction = "OUT"/>
    <fadeIn effect="Classes.Animation.Fade"
            direction = "IN"/>
    <close effect="Classes.Animation.Visibility"
           direction= "OUT"/>
    <open effect="Classes.Animation.Visibility"
          direction = "IN"/>
  </transition>
</ActionsMapper>

```

**Figure 3. Actions mapping.**

## FLEXIXML

Put simply, the main goal of FlexiXML is to produce Final User Interfaces from Concrete User Interfaces. Context and Resources models provide additional information that shapes the generation of the generated user interface. By definition Adobe's Air runtime environment is considered as belonging to the context of use. Currently the CUI model defines the user interface, the Context model defines the available user languages, and the Resources model defines language dependent attributes.

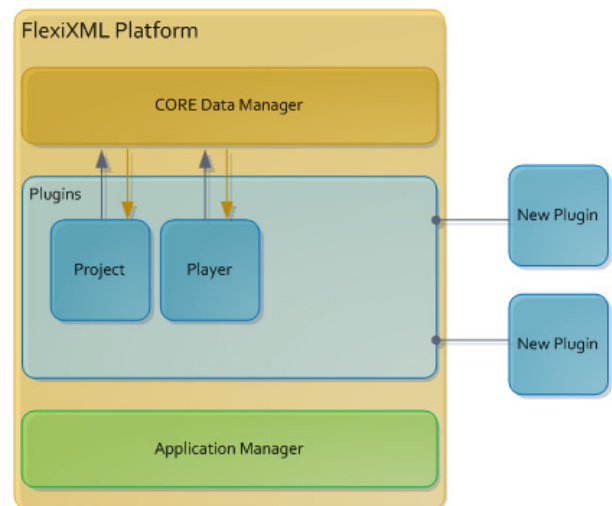
In the next sections the main features of the FlexiXML tool are introduced. An overview of the application is made, its architecture is described, and an example of use presented.

FlexiXML is structured around the concept of plugins, where each plugin implements a set of specific functions. This approach allows for constant evolution of the tool

through the integration of new plugins and other features, with no impact on existing ones.

Figure 4 presents the architecture of the tool. This architecture can be divided into 3 layers:

- *Application Manager* – this layer performs application management. Its main responsibilities include loading and coordinating available plugins, and dealing with messages localization.
- *Plugins* – This is the layer where plugins are stored. In addition to the two default plugins (Project and Player), it is possible (through the Application Manager) to integrate additional plugins into this layer.
- *CORE Data Manager* – This layer is responsible for storing and providing information that is shared by all plugins, the most relevant being the CUI model to be rendered and animated.



**Figure 4. FlexiXML platform.**

As stated, besides allowing FlexiXML to integrate new plugins, the current version provides two default plugins: Project and Player.

### Project plugin

The Project plugin is responsible for loading the project. A project file identifies two further files: a UsiXML file with the CUI model describing the interface, and an ActionScript file with the dialogue control (i.e. the event handlers associated to controls in the CUI model). This arrangement promotes reuse since different CUI models can be used with the same event handlers and vice-versa.

The Project plugin loads the two files. A parser is then responsible for interpreting the file describing the interface and for filling the core data structures with this information so that others plugins might be able to use it. The parser to be used is determined by the UIDL selected by the user in this plugin. The mapping between the parsers and the

UIDLs is defined in a configuration file. Configuration files are explained later in this article. Currently, only the UsiXML parser is available, but other can be integrated. To do this the parser must implement the `parse()` method.

### Player plugin

The Player plugin is responsible for generating the graphical interface of the loaded project. In addition to generating the interface, it allows changes to the generated interface at runtime, such as changing the style or language. This is the plugin where the user specifies the programming language to be used for user interface generation.

At the time of generation of the interface, the Player Plugin accesses the CORE Data Manager to get information about the current model. This information is then interpreted, and the relevant components and/or objects created that represent it. For each component defined in the model the corresponding graphical component, its behavior, its content and the possible transitions to other components are created. The mapping between each UsiXML element and the widgets/controls in the interface is defined in a configuration file (described later in this paper).

Presently FlexiXML includes a generator for ActionScript 3 only. However, other generators can be integrated. To do this the generator must implement the interface defined in Figure 5.

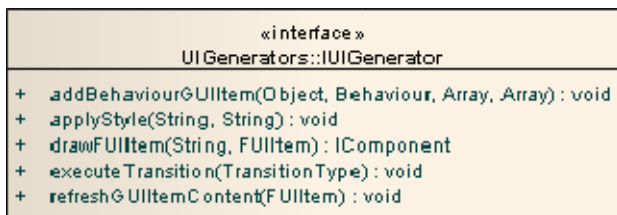


Figure 5. Generator interface.

As showed in the figure, a generator must be able to:

- Add behavior to a widget (`addBehaviourGUIItem`);
- Apply a style to the graphical user interface (`applyStyle`);
- Draw a graphic component (`drawFUIItem`);
- Play transitions between components (`executeTransition`);
- Update the contents of a widget (`refreshGUIItemContent`).

### New plugins

The list of plugins that FlexiXML provides is defined in an XML configuration file. The Application Manager reads this file during the initial process of starting the application. For each plugin, this file indicates its name, description,

and the class implementing it. It is this class that the Application Manager will load to make the plugin available.

For the integration of a plugin into the platform to be possible, the plugin must be defined as a specialization of class `PluginBase` (Figure 6).

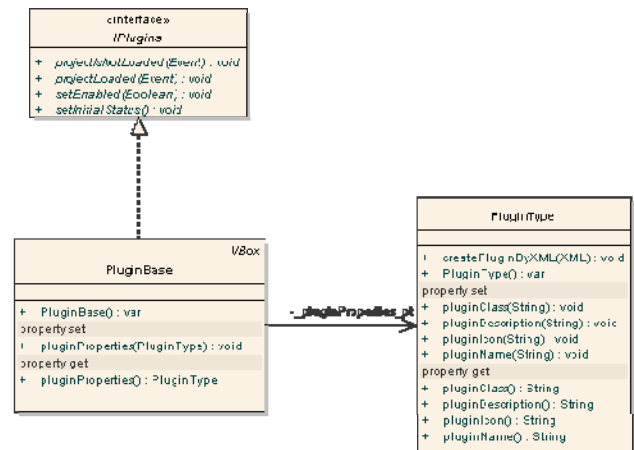


Figure 6. Class diagram of a FlexiXML plugin.

The subclasses of `PluginBase` inherit and must set a `PluginType` object, which contains all the necessary properties for plugin characterization: name, description, icon, etc. In addition to setting these properties, the class must implement the `IPlugin` interface. This interface defines all required methods for FlexiXML to be able to interact with the plugin. The methods defined in this interface are:

- **setInitialStatus()** – defines the initial state of the plugin: active or inactive;
- **setEnabled(enabled:Boolean)** – assigns a specific state to the plugin: active or inactive;
- **projectLoaded(event\_evt:Event)** – method executed whenever a new project is uploaded into the application;
- **projectIsNotLoaded(event\_evt:Event)** – method executed whenever there is no longer a project in the application.

### Configuration

As already mentioned, a set of configuration files allows changing the behavior and visual aspects of user interfaces generated by FlexiXML. The main configuration files are:

- *Messages* – All messages used in the application have an associated code. The message string associated with each code is defined in a XML configuration file. The existence of this file allows the FlexiXML to be localized without the need to recompile the code.

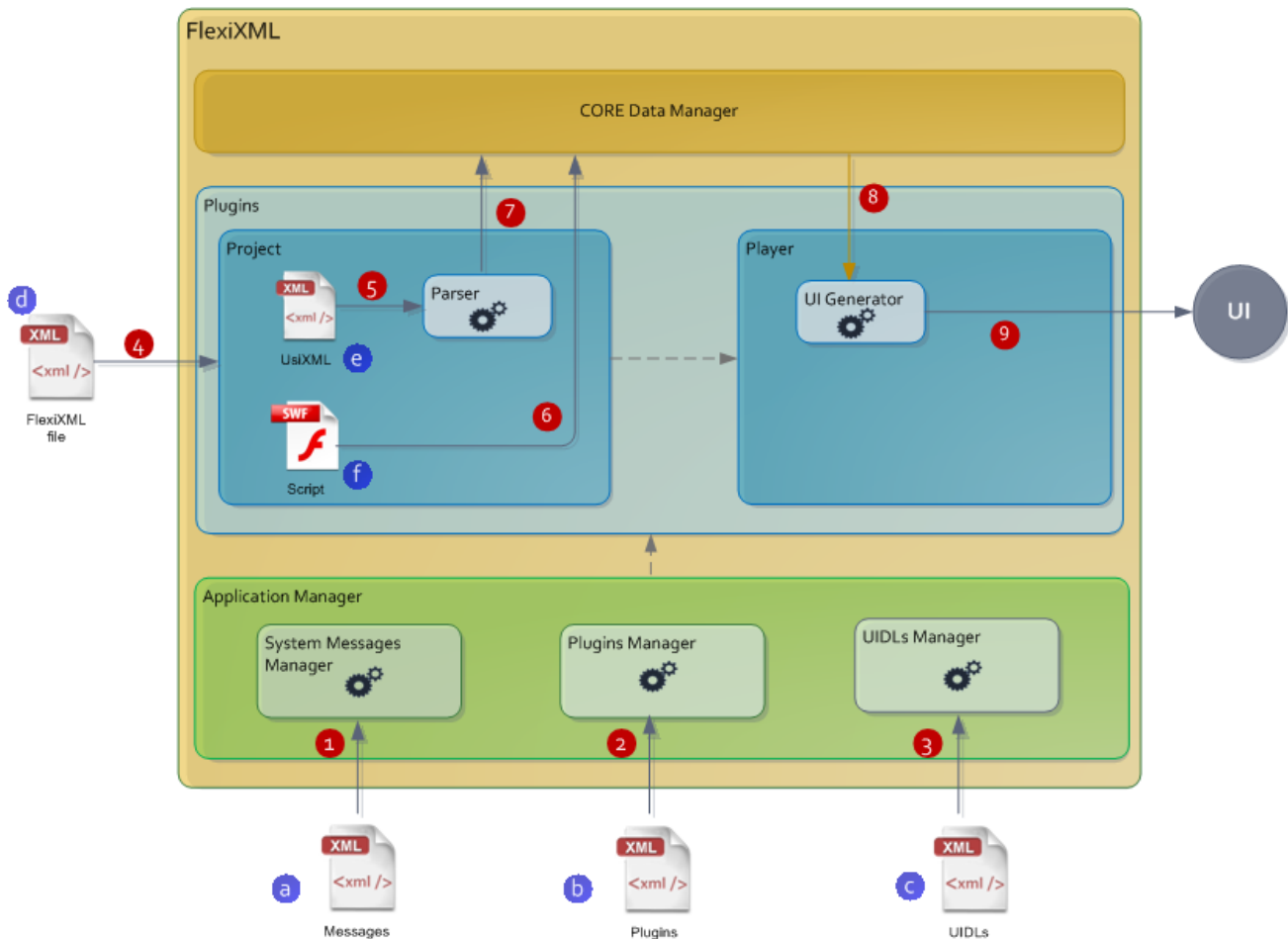


Figure 7. FlexiXML workflow.

- *Plugins* – Defines the list of available plugins. If a new plugin needs to be inserted, it is necessary to include its information in this file for the application be able to load it.
- *UIDLs* – This file defines the list of UIDLs that FlexiXML interprets. In addition to listing the available UIDLs, it defines all the characteristics needed for their integration into the application. These include: the parser for the UIDL, the available programming languages for generating the interface, the mapping between the UIDL objects and the widgets that can represent it, among others.
- *Styles* – The list of styles available in the Player plugin is defined in this XML file. Thus, whenever there is the need to insert new styles, they simply have to be added to this file.

#### Workflow of the Generation Process

Previous sections, have described the basic building blocks of the FlexiXML's architecture. The process carried out by the tool for generating a graphical interface is now described. This process is depicted in Figure 7. The figure

identifies both the inputs to the tool (labeled with letters *a* to *d*), and the flow of information (labeled with numbers *1* to *9*).

FlexiXML takes as input a set of configuration files (labels *a*, *b* and *c* in Figure 7) that are interpreted by dedicated managers (steps 1, 2 and 3 in Figure 7). These managers keep this information, which (once the tool is running) can then be accessed by any one of the plugins that has been loaded (see below). The managers are:

- *System Messages Manager* – this component is responsible for loading of the messages to be used in the application;
- *Plugins Manager* – this component is responsible for loading the listed plugins into the application;
- *UIDLs Manager* – this component is responsible for loading information of available UIDLs, and making this information available to plugins.

Once this initial processing has been done, the application becomes available, with all the plugins that have been configured.

Once the configuration of the tool is set up, the process for generating a graphical interface can start. For that, the tool needs to load the two project files: one containing the UsiXML model (label e), and the other containing dialogue control written in ActionScript (label f). The process starts (step 4) by receiving as input a project file (label d) where the location of these two files is provided. The UsiXML model is interpreted by the Project plugin (step 5), which sends the information therein to the CORE Data Manager, together with the dialogue control information (steps 6 and 7). The Project plugin then creates the entities representing the components in the CUI model, and which the Player will afterwards interpret. The CORE Data Manager centralizes all the information that can be shared by the plugins. Thus, when a plugin needs information about the current project it must request it from this manager.

Once the project is loaded into the CORE Data Manager, the Player can generate a graphical user interface (step 9) based on the information provided by the CORE Data Manager (step 8).

### AN ILLUSTRATIVE EXAMPLE

This section presents an example of a GUI generated using the FlexiXML tool. The example is an application to display and listen to music albums. Given the size of the model, only a few excerpts are presented here. The full model can be downloaded from the project's webpage<sup>9</sup>.

#### Design

The user interface is generated inside a main box (ii) in the application's main window (i). In the concrete case of the "Music Player", the user interface consists of a window that can be divided into two main areas (see Figure 8): a header box (item iii in the figure) containing the application's controls; and an area (CurrentView) for displaying information about the albums collection (item vi in the figure).

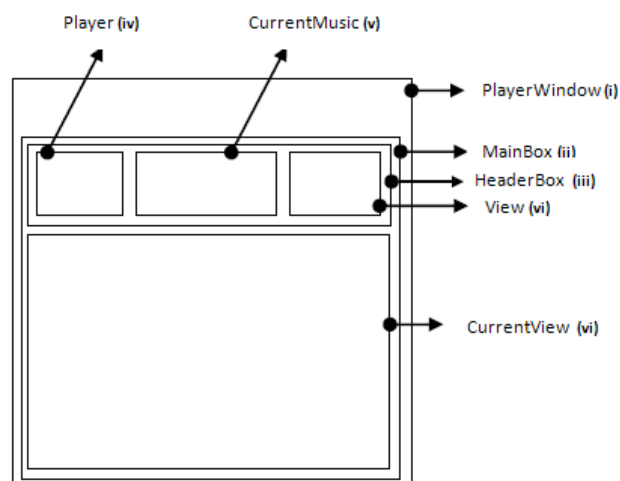


Figure 8. Decomposition of the application in areas.

```
<cuiModel id="musicPlayer-cui"
  name=" musicPlayer-cuiModel">
  <window id="playerWindow" ...> (i)
    <box id="mainBox" ...> (ii)
      <box id="headerBox" ...> (iii)
        <box id="playerBox" .../> (iv)
        <box id="currentMusicBox" .../> (v)
        <box id="viewsBox" .../> (vi)
      </box>
      <box id="currentView" ...> (vii)
        [...]
      </box>
    </window>
  </cuiModel>
```

Figure 9. CUI model of the MusicPlayer application.



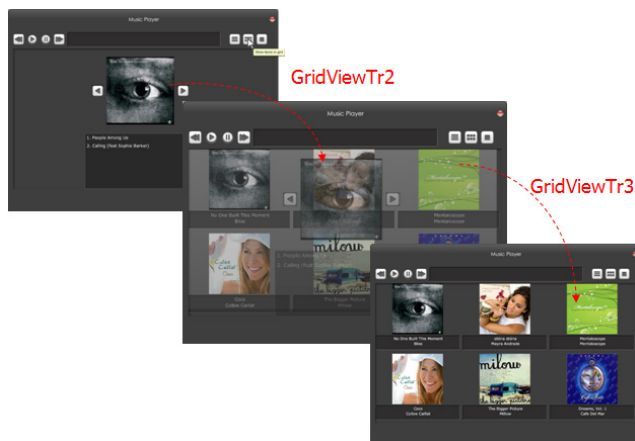
Figure 10. Music player (coverView and gridView) generated by FlexiXML.

The header box is itself subdivided into three areas:

- *Player* – the area where the buttons to control the music are placed (item iv in the figure);
- *CurrentMusic* – the area that displays information about the music currently playing (item v in the figure);
- *View* – the area where the buttons to switch between the different display formats of the albums list are placed (item vi in the figure).

<sup>9</sup> <http://FlexiXML.di.uminho.pt> (visited 14/07/2011)





**Figure 11. Graphical transitions between views.**

The View area contains three buttons for toggling between the three available display formats:

- *ListView* (to see the albums and theirs songs in list format);
- *GridView* (to see the albums in a grid);
- *CoverView* (to see the albums one at a time, by their cover).

Users may, at any time, change the view being used.

Figure 9 shows the basic structure of the model. The excerpt shown identifies the main structural components. For readability, the details of each component are omitted. The end result of the generation process is shown in Figure 10.

#### Behaviour

The above model describes the structure of the user interface. It is now necessary to model its behaviour. This will be illustrated with a concrete example.

As can be seen in Figure 10, it is possible to have different views of the album collection. Switching between views is achieved by pressing the corresponding buttons in the interface. Figure 11 illustrates the effect of pressing the "GridView" button: the "GridView" view should be displayed, and the previous view hidden.

The model specifying the behaviour of the GridView button is presented in Figure . When a depressed event happens in the button, a sequence of three transitions is fired: GridViewTr1, GridViewTr2, and GridView-Tr3. In addition, the method `updateGridView` must be invoked. This method is responsible for providing the necessary data to this view (e.g., list of albums to view).

Figure 12 defines the sequence of transitions, but does not describe what each transition actually is. That is done in Figure 13. There, it can be seen that GridViewTr1 and GridViewTr2 correspond to fade-outs of the grid and over views, respectively, while GridViewTr3 corresponds to the fade-in of the list view.

```
<button id="gridButton">
  <behavior id="gridView">

    <event id="gridViewEvt"
      eventType="depress"
      eventContext="gridButton"/>

    <action id="gridViewAct">
      <transition transitionIdRef
        ="GridViewTr1" />
      <transition transitionIdRef
        ="GridViewTr2" />
      <transition transitionIdRef
        ="GridViewTr3" />

      <methodCall methodName
        ="updateGridView"/>

    </action>
  </behavior>
</button>
```

**Figure 12. Specification of the behavior of the "GridView" button.**

```
<graphicalTransition id="GridViewTr1"
  transitionType="fadeOut">
  <source id="gridButton" />
  <target id="listView" />
</graphicalTransition>

<graphicalTransition id="GridViewTr2"
  transitionType="fadeOut">
  <source id="gridButton" />
  <target id="coverView" />
</graphicalTransition>

<graphicalTransition id="GridViewTr3"
  transitionType="fadeIn">
  <source id="gridButton" />
  <target id="gridView" />
</graphicalTransition>
```

**Figure 13. Specification of the graphical transitions triggered by the "GridView" button.**

#### Context (Language)

As already stated, FlexiXML supports localization of the interface via the definition of language contexts. To illustrate this use of context, the steps needed to make the interface available in two languages are now put forward.

Two models must be created to build the application in different languages: the ContextModel (to create the languages) and the ResourceModel (to specify the contents of the objects in each language). Figure 14 defines two languages for the "Music Player" application: English and French. Then Figure shows the specification of the application title in the two languages. Figure 15 shows the application window with the titles in both languages.

FlexiXML also allows changing the style of the generated interface at runtime. The list of styles that can be applied is defined in a configuration file. In this file, the name of the style, and the location of the style file (CSS or SWF format) are indicated.



Each style is defined in CSS (Cascading Style Sheet) format. This enables a style to contain images (skins), fonts, class selectors, among others. FlexiXML interprets this style from an external SWF (Shockwave Flash) file. The SWF files arise from the conversion of CSS files.

Figure 17 shows the “Music Player” in two different styles. In this case, the background colors of the header and song list where changed.

```
<contextModel id="playerContextModel"
  name="playerContextModel">
  <context id="playerContext_En_US"
    name="playerContext_En_US">
    <userStereotype
      id="playerContextUser_US"
      language="en_US"
      stereotypeName="playerContextUser_US"/>
    </context>
    <context id="playerContext_FR"
      name="playerContext_FR">
      <userStereotype
        id="playerContextUser_FR"
        language="FR"
        stereotype-
        Name="playerContextUser_FR"/>
      </context>
    </contextModel>
```

Figure 14. Specification of the languages.

```
<resourceModel id="playerResourceModel"
  name="playerResourceModel">
  <cioRef cioId="playerWindow">
    <resource content="Music Player"
      context-
      tId="playerContext_En_US"/>
    <resource content="Lecteur de Musique"
      contextId="playerContext_FR"/>
  </cioRef>
</resourceModel>
```

Figure 15. Specification of the application title in different languages.

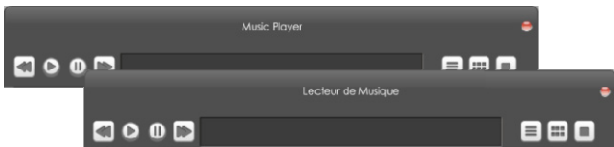


Figure 16. "Music Player" in English and French Styles.

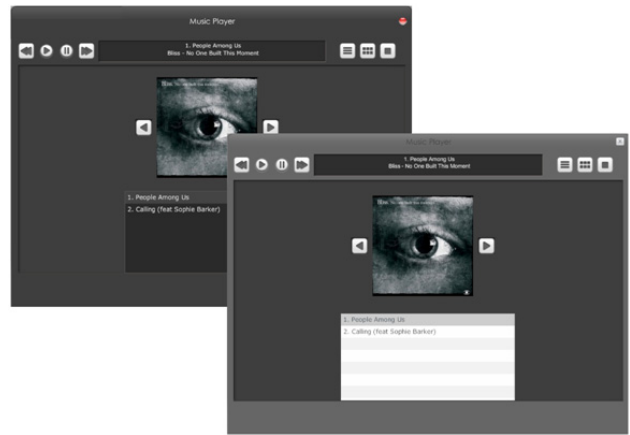


Figure 17. “Music Player” with different styles.

## CONCLUSION AND FUTURE WORK

The acceptance of an application depends largely on the quality of its graphical interface. Model-based user interface development helps ensure the quality of the solution can be assessed at an early stage [4], allowing for the problems identified to be analyzed as soon as possible. This is achieved through the creation of models at different levels of abstraction, from the domain model to the final user interface. To maximize model-based development, tools support is necessary both to enable analysis of the models, and to enable moving between levels of abstraction.

The aim of this project was the creation of a tool to support automatic generation of user interfaces from models expressed in the UsiXML language. The current version of the tool supports the UsiXML language, but is designed to allow the inclusion of other XML-based declarative languages.

In this version of the project, only a sub-set of the potential of the UsiXML language is used. For example, the possibility of defining characteristics of the computing platform in the context model was not considered. In alternative, a technology was used that enables the generated interfaces to be run in a variety of platforms. That is, the adaptation is not directly handled by FlexiXML, but by the runtime environment in which FlexiXML is executing.

The generated interfaces are created in FLEX and ActionScript 3. The tool, however, is structured so that the user can specify in which language (s)he wants the interface to be generated. These two characteristics are intended to make the tool as flexible as possible, not limiting users to particular languages (or language versions) for interface specification and generation, thus extending the number of users that can benefit from it.

The fact that the FlexiXML tool relies on the AIR runtime environment for execution makes it independent of any specific computing platform. Indeed, due to the use of the runtime environment, FlexiXML is available in two formats: Desktop and Web. Where the desktop version can be

used in any operating system that features an Air runtime environment (i.e. all major operating systems).

Looking back at the objectives initially set forth, the following features of the FlexiXML tool can be highlighted:

- Implementation in a recent technology (AIR, Flex and ActionScript3) enabling portability of the user interface (in the sense that it can be deployed in different platforms);
- An explicit, and configurable, mapping between the modeling language and the implementation technology (both regarding the structural elements of the interface, and regarding behaviour – supported events, graphical transitions);
- Support for runtime adaptation of the user interface via localization, and the use of styles to change the look of the interface;
- An architecture designed to support the integration of new plugins and new graphical interfaces modeling and/or programming languages.

At this stage, a number of future lines of work is open. Some of the possible improvements and areas for future work include:

- *Creating new plugins* – for example, a model editor;
- *Extending the widget library and the layout managers*, in order to provide a wider set of user interface representations – thus supporting the creation of more realistic user interfaces;
- *Supporting the generation of prototypes for different devices and platforms* – while the tool can be run on a number of platform due to the AIR runtime, no attempt is made at this stage to adapt the generated interface to the device being used;
- *Implementing parsers for new UIDs*, and generating the user interfaces using different programming languages and technologies.

## ACKNOWLEDGMENTS

The authors would like to thank Jean Vanderdonckt and Michael D. Harrison for their helpful comments on previous versions of this paper. Sandrine Mendes would also like to thank her employer, Alert Life Sciences Computing, for sponsoring this work.

## REFERENCES

1. Puerta, A., and Eisenstein, J. XIML: A common representation for interaction data. In *Proc. of the 7th Intl. Conf. on Intelligent User Interfaces*, ACM, 69-76.
2. Helms, J., Schaefer, R., Luyten, K., Vermeulen, J., Abrams, M., Coyette, A., and Vanderdonckt, J. Human-Centered Engineering with the User Interface Markup Language. In *Human-Centered Software Engineering*. Chapter 7, Seffah, A., Vanderdonckt, J., Desmarais, M. (Eds.), HCI Series. Springer, London (2009), pp. 141-173.
3. Argollo, M. Jr., and Olguin, C. Graphical user interface portability. *CrossTalk: The Journal of Defense Software Engineering*, 10(2):14-17, 1997.
4. Bäumer, D., Bischofberger, W.R., Lichter, H., and Züllighoven, H. User interface prototyping - concepts, tools, and experience. In *Proc. of the 18<sup>th</sup> Int. Conf. on Software Engineering ICSE '96*. IEEE Computer Society, Los Alamitos (1996), pp. 532-541.
5. Coyette, A., Kieffer, S., and Vanderdonckt, J. Multi-Fidelity Prototyping of User Interfaces. In *Proc. of 11<sup>th</sup> IFIP TC 13 Int. Conf. on Human-Computer Interaction INTERACT'2007* (Rio de Janeiro, September 10-14, 2007). Lecture Notes in Computer Science, vol. 4662. Springer-Verlag, Berlin (2007), pp. 149-162.
6. Denis, V. *Un pas vers le poste de travail unique : QTKiXML, un interpréteur d'interface utilisateur à partir de sa description*, M.Sc. thesis, Université catholique de Louvain, Belgium, September 2005.
7. Kaklanis, N., Gonzalez, J.M., Vanderdonckt, J., and Tzovaras, D. A Haptic Rendering Engine of Web Pages for Blind Users. In *Proc. of 9th Int. Conf. on Advanced Visual Interfaces AVI'2008* (Naples, May 28-30, 2008). ACM Press, New York (2008), pp. 437-440.
8. Michotte, B., and Vanderdonckt, J. GrafiXML, A Multi-Target User Interface Builder based on UsiXML. In *Proc. of 4<sup>th</sup> Int. Conf. on Autonomic and Autonomous Systems ICAS'2008* (Gosier, 16-21 March 2008). D. Greenwood, M. Grottke, H. Lutfiyya, M. Popescu (Eds.). IEEE Computer Society Press, Los Alamitos (2008), pp. 15-22.
9. Mozilla foundation. *XUL Tutorial*, [https://developer.mozilla.org/en/XUL\\_Tutorial](https://developer.mozilla.org/en/XUL_Tutorial). Last accessed on November 22, 2010.
10. Paternó, F., and Santoro, C. One model, many interfaces. In *Proc. of the 4<sup>th</sup> Int. Conf. on Computer-Aided Design of User Interfaces CADUI'2002*. Ch. Kolski, J. Vanderdonckt (Eds.). Kluwer Academics Publishers, Dordrecht (2002), pp. 143-154.
11. Silva, E. *Sistemas interactivos*. Departamento de Computação, Universidade Federal de Ouro Preto. 2006.
12. Guerrero-García, J., González-Calleros, J.M., Vanderdonckt, J., and Muñoz-Arteaga, J. A Theoretical Survey of User Interface Description Languages: Preliminary Results. In *Proc. of Joint 4<sup>th</sup> Latin American Conference on Human-Computer Interaction-7th Latin American Web Congress LA-Web/CLIHIC'2009* (Merida, November 9-11, 2009). E. Chavez, E. Furtado, A. Moran (Eds.). IEEE Computer Society Press, Los Alamitos (2009), pp. 36-43.

13. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J. A Unifying Reference Framework for Multi-Target User Interfaces. *Interacting with Computers* 15,3 (2003) 289-308.
14. Limbourg, Q., and Vanderdonckt, J. UsiXML: A User Interface Description Language Supporting Multiple Levels of Independence. In *Engineering Advanced Web Applications*, M. Matera, S. Comai, S. (Eds.). Rinton Press, Paramus (2004), pp. 325-338.
15. Goffette, Y., and Louvigny, H.-N. *Development of multimodal user interfaces by interpretation and by compiled components : a comparative analysis between InterpiXml and OpenInterface*, M.Sc. thesis, UCL, Louvain-la-Neuve, 28 August 2007
16. Vanderdonckt, J., Guerrero-Garcia, J., González-Calleros, J.M., A Model-Based Approach for Developing Vectorial User Interfaces. In *Proc. of Joint 4th Latin American Conference on Human-Computer Interaction-7th Latin American Web Congress LA-Web/CLIHIC'-2009* (Merida, November 9-11, 2009), E. Chavez, E. Furtado, A. Moran (Eds.), IEEE Computer Society Press, Los Alamitos, 2009, pp. 52-59.
17. Souchon, N. and Vanderdonckt, J. A Review of XML-Compliant User Interface Description Languages. In *Proc. of 10th Int. Conf. on Design, Specification, and Verification of Interactive Systems DSV-IS'2003* (Madeira, 4-6 June 2003). J. Jorge, N.J. Nunes, J. Cunha (Eds.). Lecture Notes in Computer Science, vol. 2844, Springer-Verlag, Berlin (2003), pp. 377-391.
18. Vanderdonckt, J. Model-Driven Engineering of User Interfaces: Promises, Successes, and Failures. In *Proc. of 5th Annual Romanian Conf. on Human-Computer Interaction ROCHI'2008* (Iasi, September 18-19, 2008), S. Buraga, I. Juvina (Eds.). Matrix ROM, Bucarest (2008), pp. 1-10.

# Model-Driven Engineering of Dialogues for Multi-platform Graphical User Interfaces

Efrem Mbaki<sup>1</sup>, Jean Vanderdonckt<sup>1</sup>, Marco Winckler<sup>2</sup>

<sup>1</sup> Louvain School of Management (LSM),  
Université catholique de Louvain,  
Place des Doyens, 1  
B-1348 Louvain-la-Neuve (Belgium) –  
{efrem.mbaki@student,  
jean.vanderdonckt}@uclouvain.be

<sup>2</sup>IRIT  
Université Paul Sabatier,  
118 Route de Narbonne  
F-31062 Toulouse CEDEX 9  
winckler@irit.fr

## ABSTRACT

This paper describes a model-driven engineering of interactive dialogues in graphical user interfaces that is structured according to the three lowest levels of abstraction of the Cameleon Reference Framework: abstract, concrete, and final user interface. A dialogue model captures an abstraction of the dialogue as opposed to a traditional presentation model that captures the abstraction of the visual components of a user interface. The dialogue modeled at the abstract user interface level can be reified to the concrete user interface level by model-to-model transformation, which in turn leads to code by model-to-code generation. Five target markup and programming languages are supported: HTML V4.0, HTML for Applications (HTA), Microsoft Visual Basic for Applications V6.0 (VBA), and DotNet V3.5 framework. Two computing platforms support these languages: Microsoft Windows and Mac OS X. Five levels of dialogue granularity are considered: object-level (dialogue of a particular widget), low-level container (dialogue of any group box), intermediary-level container (dialogue at any non-terminal level of decomposition such as a dialog box or a web page), intra-application level (application-level dialogue), and inter-application level (dialogue across different interactive applications). A Dialog Editor has been implemented that holds abstractions pertaining for expressing the dialogue at an abstract level and then producing a final user interface.

## Author Keywords

Dialogue model, event-condition-action rule, model-driven engineering, user interface description language.

## General Terms

Algorithms, Design, Languages, Theory.

## Categories and Subject Descriptors

C.2.4 [Computer-Communication Networks]: Distributed systems – *Distributed applications*. D2.2 [Software Engineering]: Design Tools and Techniques – *Modules and interfaces; user interfaces*. D2.m [Software Engineering]: Miscellaneous – Rapid Prototyping; reusable software. H5.2 [Information interfaces and presentation]: User Interfaces – *graphical user interfaces, user interface management system (UIMS)*.

## INTRODUCTION

We hereby refer to *dialogue* as being the dynamic part of a Graphical User Interface (GUI) such as the physical and temporal arrangement of widgets in their respective containers and their evolution over time depending on the user's task. The dialogue regulates the ordering of these widgets so as to reflect the constraints imposed by the user's task. The dialogue has been also referred to as *behavior*, *navigation*, or *feels* (as opposed to *look* for presentation) [1,2,12]. Here are some typical examples of dialogues: when the end user selected her native language in a list box, a dialog box is translated accordingly; when a particular value has been entered in an edit field, other edit fields are deactivated because they are no longer needed; when a validation button is pressed, the currently opened window is closed and another one is opened for pursuing the dialog.

Conceptual modeling [1], model-based design [4] or model-driven engineering [20] of the dialog has already been introduced since years [12] in order to be derived from a task model [11,17,25,31,33], perhaps combined with a domain model [30] or a service model [4], to derive its software architecture from its model [23], to analyze its properties [5,32], to foster component reuse [10], to check some dialogue or usability properties [32], to support adaptation [19], to automatically keep trace of interaction and analyze them afterwards [25]. Dialog models have been used in several domains of applications, such as web engineering [3,5], information systems [18], multi-device environments [27], multimedia applications [23,24], multimodal applications [28], and workflow systems [29,30].

Dialogue modeling has however often been considered harmful for several reasons which may impedit further research and development in this area:

1. *Choosing the modeling language paradigm is a dilemma*: an imperative or procedural language is often more suitable and convenient to represent a GUI dialogue than a declarative language. The last could introduce a verbose representation of something that could be expressed in a straightforward way in the latter. The current trend goes in favor of scripting languages.

2. *Abstracting the right concepts is complex*: finding the aspects of a dialog that should lead to abstraction is not straightforward and turning them into an abstraction that is expressive enough without being verbose is hard. A dialogue model may benefit from a reasonable level of expressiveness, but will prevent the designer from specifying complex dialogues while another dialogue model may exhibit more expressiveness, but is considered complex to use. Which modeling approach is also an open question: taking the greatest common denominator across languages (with the risk of limited expressiveness) or more (with the risk of non-support).
3. *Heterogeneity of computing platforms is difficult to handle*: Integrated Development Environments (IDEs) are often targeted to a particular programming language or markup language that is dedicated to a particular operating system or platform. Some IDEs exist (e.g., Nokia QT (<http://qt.nokia.com/products>, QtK) that address multi-platform GUIs, but they remain at the code level or their usage is still complex.
4. *Model-driven engineering of dialogue is more challenging than model-based design*. Model-based GUI design only assumes that one or many models are used to design parts or whole of a GUI, while Model-Driven Engineering (MDE) [21] imposes at least one User Interface Description Language (UIDL) [7] that should be rigorously defined by a meta-model (preferably expressed in terms of MOF language, but not necessarily). Model-based GUI design may invoke virtually any technique, while model-driven engineering imposes that everything is rigorously defined in terms of model transformations, which are in turn based on a meta-model.

This paper is aimed at addressing the aforementioned challenges by applying MDE principles to designing a dialog for GUIs belonging to different computing platforms. The remainder of this paper is structured as follows: Section 2 reports on the main trends so far in dialogue modeling, Section 3 defines the conceptual model of dialogue used in this paper, Section 4 presents an overview of the methodological approach with three views: model & language, step-wise approach, and software support. A running example is given to exemplify how this approach is executed. Section 5 motivates our software implementation with multi-level dialog model editing, model-to-model transformation, and model-to-code generation. Section 5 concludes the paper and addresses some avenues.

## STATE OF THE ART

### Overview of dialogue modeling techniques

A very wide spectrum of conceptual modeling and computer science techniques has been used over years to model a dialogue [1-5, 8-14, 16-35], some of them with some persistence over time, such as, but not limited to: Backus-Naur Form (BNF) grammars [12, 16], state-transition diagrams in

very different forms (e.g., dialog charts [1], dialog flows [3], abstract data views [10], dialog nets [9], windows transitions [33]), state charts [14] and its refinement for web applications [5], and-or graphs coming from Artificial Intelligence (e.g., function chaining graphs [18]), event-response languages, and Petri nets [2]. Some algorithms [17] have been also dedicated to support the dialog design through models, such as the Enabled Task Set [22].

Rigorously comparing these models represents a contribution that is yet to appear. Green [9] compared three dialogue models to conclude that some models share the same expressivity, but not the same complexity. Cachero *et al.* examine how to model the navigation of a web application [5]. In [9], the context model drives a dialogue model at different steps of the UI development life cycle.

So far, few attempts have been made to structure the conceptual modeling of dialogues in the same way as it has been done for presentation, the notable exception being applying StateWebCharts [34] with Cascading style sheets [35] in order to factor out common parts of dialogues and to keep specific parts locally.

### Some recent dialogue modeling techniques

The DIAMODL runtime [30] models the dataflow dialog as JFace Data Binding and includes extensions for binding EMF data to SWT widgets in order to link domain and dialogue models. Statechart logic is implemented by means of the Apache SCXML engine [36], while GUI execution utilizes an XML format and renderer for SWT.

The Multimodal Interface Presentation and Interaction Model (MIPIM) [28] could even model complex dialogues of a multimodal user interface together with an advanced control model, which can either be used for direct modeling by an interface designer or in conjunction with higher level models.

Van den Bergh & Coninx [31] established a semantic mapping between a task model with temporal relationships expressed according to ConcurTaskTrees notation and UML state machines as a compact way to model the dialog, resulting into a UML profile.

Figure 1 graphically depicts some dialogue models in families of models. Each family exhibits a certain degree of model expressiveness (i.e., the capability of the model to express advanced enough dialogues), but at the price of a certain model complexity (i.e., the easiness with which the dialogue could be modeled in terms specified by the meta-model). At the leftmost part of Figure 1 are located (E)BNF grammars since they are probably the least expressive dialogue models ever.

Then we can find respectively State Transitions Networks and their derivatives, then Event-Response Systems. Petri nets [2] are probably the most expressive models that can be used to model dialogues, but they are also the most complex to manipulate.

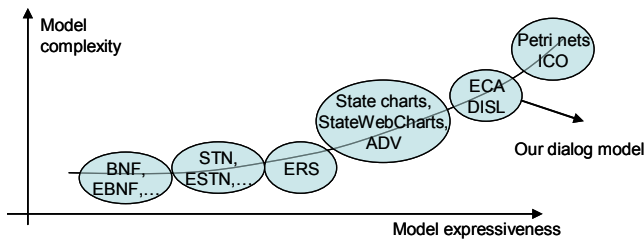


Figure 1. Expressiveness of Model Complexity.

### Dialogue modeling techniques in UIDLs

Less expressive and less complex are Event-Condition-Action (ECA) systems that are considered in several UIDLs such as DISL [27,28], UIML [15], MariaXML [22] and UsiXML [35], probably because they are convenient to describe according to a declarative paradigm, that is often predominant in defining models in UIDLs. But their expressivity is limited by the model concepts coverage.

In *UIML* [15], a dialogue is defined as a set of condition-action rules that define what happens when a user interacts with any GUI element, such as a button, a group box, a window. In *MariaXML* [22], a dialog model describes parallel interaction between a GUI and its end user through connections. A *connection* indicates what the next active presentation will be when a given interaction takes place: elementary connection, complex connection (in which a logical formula composes elementary conditions), or a conditional connection (when specific conditions are associated with it).

In *UsiXML* [35], a behavior is defined as a set of ECA rules, where: an *event* can be any UI event that is relevant to the level of abstraction (abstract or concrete), a *condition* can state any logical condition on a model, a model element, or a mapping between models, an *action* can be any operation on widgets (abstract or concrete).

### The Cameleon Reference Framework

Several UIDLs [7] are structured according to the four steps of the Cameleon Reference Framework (CRF) [6], that are now recommended to consider by W3C [7]:

- 1) *Task & Concepts* (T&C): describe the various user's tasks to be carried out and the domain-oriented concepts required by these tasks to be performed.
- 2) *Abstract UI* (AUI): defines abstract containers (AC) and individual components (AIC), two forms of Abstract Interaction Objects (AIO) by grouping subtasks according to various criteria (e.g., task model structural patterns, cognitive load analysis, semantic relationships identification). As in Guilet Dialog Model [26], a navigation scheme between the container and selects abstract individual component for each concept so that they are independent of any interaction modality. The AUI is said to be *independent of any interaction modality*.

- 3) *Concrete UI* (CUI): concretizes an abstract UI for a given context of use into Concrete Interaction Objects (CIOs) so as to define widgets layout and interface navigation. It abstracts a final UI into a UI definition that is independent of any computing platform. A CUI assumes that a chosen interaction modality, but the CUI remains *independent of any platform*.

- 4) *Final UI* (FUI): is the operational UI i.e. any UI running on a particular computing platform either by interpretation (e.g., through a Web browser) or by execution (e.g., after compilation of code in an IDE).

### CONCEPTUAL MODELING OF DIALOGUE

In order to apply MDE techniques, we need to define a dialog model that is expressive enough to accommodate advanced dialogues at different levels of granularity and different levels of abstraction, while allowing some structured design and development of corresponding dialogue. The BCHI Dialogue Editor described in this paper will rely on this conceptual model. For this purpose, our conceptual modeling consists of expanding ECA rules towards *dialogue scripting* (or *behavior scripting*) in a way that is independent of any platform. This dialogue scripting is structured according to a meta-model that is reproduced in Figure 2 that enables defining a dialogue at five levels of granularity:

1. *Object-level dialogue modeling*: this level models the dialogue at the level of any particular object, such as a CIO or a AIO. In most cases, UI toolkits and IDEs come up with their own widget set with built-in, predefined dialogue that can be only modified by overwriting the methods that define this dialogue. Only low-level toolkits allow the developer to redefine an entirely new dialogue for a particular widget, which is complex.
2. *Low-level container dialogue modeling*: this level models the dialogue at the level of any container of other objects that is a leaf node in the decomposition. Typically, this could be a terminal AC at the AUI level or a group box at the CUI level in case of a graphical interaction modality.
3. *Intermediary-level container dialogue modeling*: this level models the dialogue at the level of any non-terminal container of objects, that is any container that is not a leaf node in the container decomposition. If the UI is graphical, this could be a dialog box or the various tabs of a tabbed dialog box.
4. *Intra-application dialogue modeling*: this level models the dialogue at the level of top containers within a same interactive application such as a web application or a web site. It therefore regulates the navigation between the various containers of a same application. For instance, the Open-Close pattern means that when a web page is closed, the next page in the transition is opened.



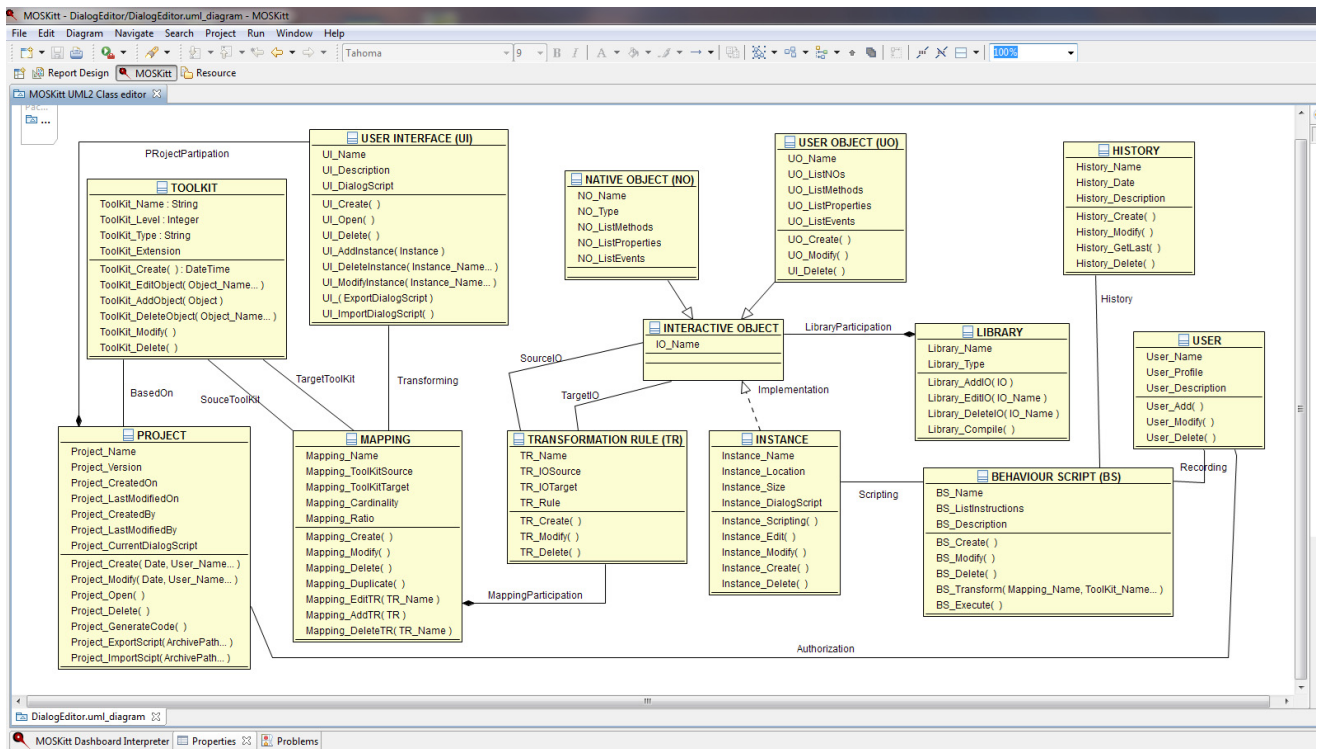


Figure 2. A Conceptual model of dialogue as a basis for model-driven engineering (implemented in Moskit).

5. *Inter-applications dialogue modeling*: since the action term of an ECA rule could be either a method call or an application execution, it is possible to specify a same dialogue across several applications by calling an external program. Once the external program has been launched, the dialogue that is internal to this program (within-application dialog) can be executed.

#### Levels of dialogue granularity

Now that these five levels are defined, we introduced the concepts used towards the conceptual modeling of dialogues that could be structured according to the five aforementioned levels of granularity. These concepts are introduced, defined, and motivated in the next sub-sections.

**Interactive Object.** An *interactive object* is the core component of the conceptual model as it consists of any object perceivable by the end user who could act on it. Interactive objects are further sub-divided into three levels of abstraction depending on the CRF [6]: *abstract*, *concrete*, and *final* (Figure 3 shows how this hierarchy is implemented in the BCHI Dialogue Editor respectively at the three levels).

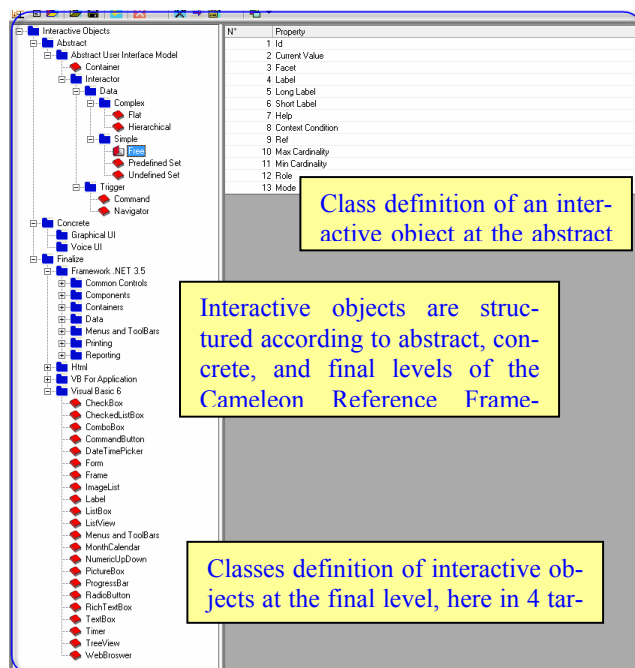
**Abstract Interactive Objects.** They describe interactive objects at the Abstract User Interface (AUI) level of the CRF. In the BCHI Dialog Editor, they are implemented as abstract classes compliant with Morfeo's Abstract UI model ([http://forge.morfeo-project.org/wiki\\_en/index.php/Abstract\\_User\\_Interface\\_Model](http://forge.morfeo-project.org/wiki_en/index.php/Abstract_User_Interface_Model)) which has been selected for the following reasons: Morfeo's AUI is one of the most

recent effort to define AUI that has been successfully implemented in the Morfeo project and has therefore been recommended as a reference model for European NESSI platform ([www.nessi.eu](http://www.nessi.eu)) through the FP7 Nexof-RA project ([www.nexofra.eu](http://www.nexofra.eu)) which promotes a reference software architecture for interactive systems, including the GUI part. Morfeo's AUI model holds two object types: an *interactor* manipulates data as input, output, or both, through *simple* interaction mechanism (e.g., a selection) or through *complex* ones (e.g., a vector, a hierarchy); a *container* could contain interactors and/or other containers. Figure 3 details the definition of the abstract class implemented for the *Free* object that serves for general-purpose input/output.

**Concrete Interactive Objects.** They describe interactive objects at the Concrete User Interface (CUI) level of the CRF. In the BCHI Dialog Editor, they are implemented as abstract classes for one modality at a time. Figure 3 shows that graphical and vocal modalities are included, but only the graphical part is the subject of this paper. Such concrete interactive objects may range from simple widget such as a push button, a slider, a knob to more complex ones such as group box, dialog box, tabbed dialog box.

If we abstract an interactive object from its various physical representations that belong to the various computing platforms and window managers, any interactive object is be characterized by its attributes and dialogue. An object may react to the end user's actions by handling events gen-

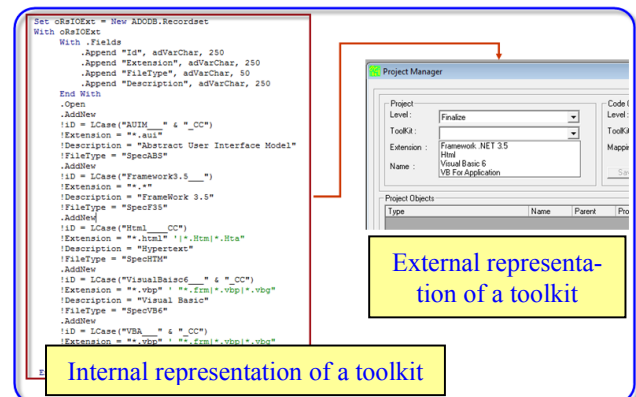
erated by this object. Therefore, a class could introduce an abstraction of object characteristics, including its *attributes* (fields or properties), its *methods* (through which a concrete interactive object could be manipulated) and its *events* (that could be generated by, or received by, a concrete interactive object). A class is hereby considered as a model of interactive objects of the same type. For example, a *TextBox* of a GUI consists of a rectangular widget for entering text, characterized by attributes including *width*, *height*, *backgroundColor*, *maxLength* or the *currentText*. Textbox operators are also associated such as *appendText*, *giveFocus*, *selectAll* or *clearEntry*. A textbox generates events such as *textBoxSelected* when the textbox has been selected by any mean (e.g., by clicking in it, by moving the tabulation until reaching the object) or *textBoxEnter* when the GUI pointer enters in the object (e.g., by moving the mouse into it or by touching it).



**Figure 3. The hierarchy of interactive objects classes as implemented in the BCHI Dialog editor.**

**Final Interactive Objects.** They describe interactive objects at the Final User Interface (FUI) level of the CRF. In the BCHI Dialog Editor, they are implemented as real classes corresponding to various toolkits supported (Figure 3 shows the four toolkits that are currently supported with the hierarchy expanded for Visual Basic V6.0). For each interactive object, only the common native dialogue is factored out and rendered as a sub-class of the toolkit. This is why final interactive objects are represented as native objects in Figure 2, while abstract and concrete interactive objects are represented as user-defined classes in Figure 2. We hereby assume that the native dialogue of any final interactive object is preserved. For defining non-native dialogues of a final interactive object, dedicated methods exist, such as the

*Interaction Object Graph* (IOG) [8]. Since defining custom dialogue at the control level requires complex and dedicated programming, it is not supported unless such a dialogue can be characterized as an interactive object.



**Figure 4. Internal and external representation of toolkits.**

**Toolkit.** In order to support GUIs for multiple computing platforms, each supported toolkit of a particular platform is characterized by its name, its level (e.g., a version), its extensions, and a series of templates describing how this toolkit implements particular dialogues. Three values are accepted depending on which level of abstraction it is considered: abstracted (AUI), concrete (CUI) or final (FUI). Figure 4 shows the correspondence of the external representation of a toolkit that is visible to the end user and the internal representation inside the BCHI Dialogue Editor.

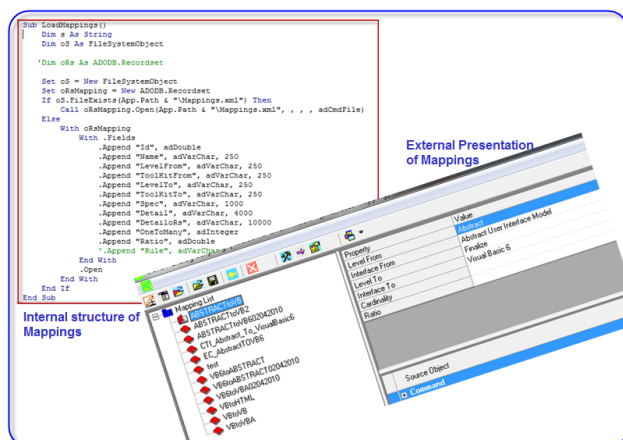
**Library.** A *library* gathers a series of particular interactive object at any level so as to refer to them as a whole, which is helpful for keeping the same definitions for one target computing platform, typically a toolkit. For the moment, HTML V4.0 is one of the supported toolkits by its corresponding library. Any newer version of HTML, e.g., V5.0, requires implementing a new library for this toolkit.

**Instance.** An *instance* is any individual object created as an instance of any interactive object class. While a class defines the type of an interactive object, any actual usage of this class is called "instance". Each class instance possesses different values for its attributes. At any *t* time, the instance state is defined by the set of its attributes values. By respecting the encapsulation i.e., the process of hiding all of the attributes of an object from any outside direct modification, object methods can be used to change an instance state. In order to have a login+password, two instances should be created that share the same definition, but with different instance states.

**User Interface.** A *User Interface* (UI) as it is considered in this conceptual model may consist of any UI at any level of abstract (i.e., abstract, concrete, or final). Therefore, such a UI consists of a set of instances each belonging to the corresponding level of abstraction.

**Project.** A *project* is considered as a set of UIs for a same case study for a particular toolkit. In a same project, one can typically find one AUI, one CUI, and one FUI. Of course, for the same AUI, different CUIs could be created that, in turn, lead to their corresponding FUIs. Actually, a project could hold as many CUIs and FUIs as model-driven engineering has been applied to the same AUI. This is achieved through the mechanism of mapping.

**Mapping.** In order to support model-driven engineering, a *mapping* is hereby referred to as any set of transformation rule from one source toolkit to a target toolkit. Note that source and target toolkits could be identical. A transformation rule is written as a PERL regular expression applied from a source class of interactive objects to a target class of interactive objects. In order to support Model-to-Model (M2M) transformation, a transformation rule may be applied from one or many classes of abstract interactive objects to one or many classes of concrete interactive objects. For Model-to-Code (M2C) generation, a transformation rule is applied from one or many classes of concrete interactive objects to one or many classes of final interactive objects (so-called *native objects*). Let us consider again the login and the password example. At the abstract level, two instances of entry fields are created to be mapped onto objects belonging to a particular toolkit. In HTML, both fields are transformed into Input objects, respectively of type Text and Password. In VB6, they are transformed into two text boxes. For the password, *IsPassword* is set to True.



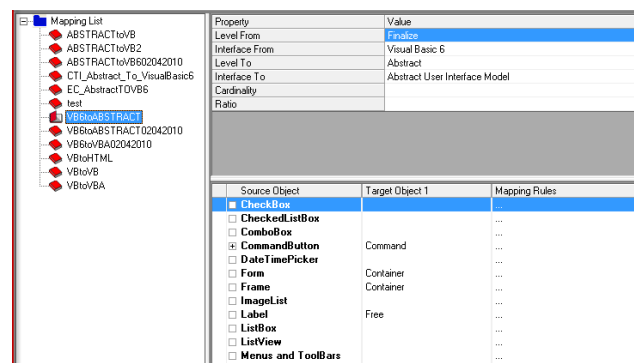
**Figure 5. Internal and external representation of mappings.**

Figure 5 shows both, the external representation of a mapping that is visible to the end user and its internal representation inside the BCHI Dialogue Editor. Note that the Cameleon Reference Framework [6] enables multiple development paths, and not just forward engineering. In *forward engineering*, transformations are supposed to transform elements of a model into elements belonging to another model whose level of abstraction is inferior (this process is referred to as *reification*). In *reverse engineering*, transformations are supposed to transform elements of a

model into elements belonging to another model whose level of abstraction is superior (this process is referred to as *abstraction*). In *lateral engineering*, transformations are applied on models belonging to the same level of abstraction, possibly the same one. Mappings as supported by the Dialogue Editor support the three types of engineering:

- (1) *Forward engineering*, where mappings transform successively the AUI model into a CUI model that, in turn, is transformed into a FUI for the four following targets: HTML V4.0, HTML for Applications (HTA), Microsoft Visual Basic for Applications V6.0 (VBA), and DotNet V3.5 framework. HTML V4.0 and HTA are running on both MS Windows and Mac OS X platforms.
- (2) *Reverse engineering*, where mappings transform something concrete into something abstract. Figure 6 depicts a mapping for reverse engineering Visual Basic V6.0 code directly into an AUI model by establishing a correspondence between native objects (Figure 2) and their corresponding user objects (Figure 2), two sub-classes of interactive objects.
- (3) *Lateral engineering*, where mappings transform model elements belonging to a same level of abstraction, but for another context of use.

Before continuing, we must emphasize that our conceptual and technical choices are guided by a desire to easily integrate our results into the usiXML environment. Indeed, conceptual model of dialogues has been implemented as UML V2.0 class diagram in Moskitt ([www.moskitt.org](http://www.moskitt.org)) (Figure 2) that gave rise to a XML Schema.



**Figure 6. Example of a mapping for reverse engineering.**

Note also that in this example, the reverse engineering does not need necessarily to work between two subsequent levels. The mapping depicted in Figure 6 goes from FUI directly to AUI without passing by the intermediary CUI level. This type of mapping is called *cross-cutting* as it represents a shortcut between two non-consecutive levels of abstraction. For example, Figure 7 depicts a mapping for forward engineering from an AUI model directly to Visual Basic V6.0 code.

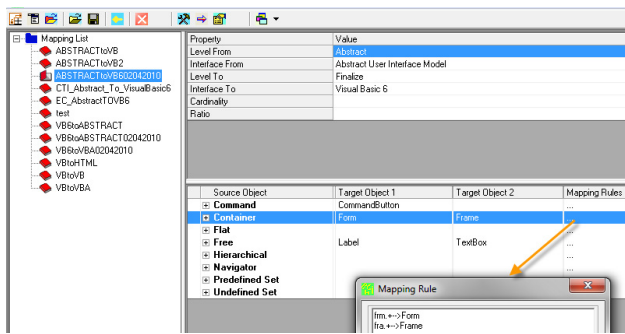


Figure 7. Example of a mapping for reverse engineering.

In the Cameleon Reference Framework, multi-target is also described in terms of different contexts of use. Therefore, any mapping that goes from one context of use to another one is referred to as *lateral engineering*. The BCHI Dialogue Editor also supports this through mappings at the same level of abstraction, but across two different contexts of use, such as between VB6 and HTML V4.0 (Figure 8).

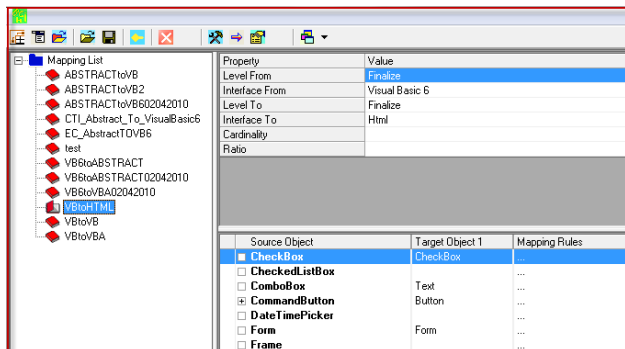


Figure 8. Example of a mapping for lateral engineering.

**Dialogue Script.** A *dialogue script* (or *behaviour script*) is a sequential text expressing the logic and conditional elements. It describes the actions to be achieved according to a given interaction scenario. An action can be the change of an attribute value, the call of a semantic function belonging to the functional core, or the opening or the closing of another user interface. Three levels of script are possible:

1. *Elementary dialogue scripts.* These scripts are related to instances found in a given project. Often, these scripts are systematically generated accordingly to a template-based approach. They can come from:
  - A change of an attribute value: for example, a read only field implies automatic database requests in its dialogue script ;
  - A layout positioning: for example, two interactive objects may be laid out in their parent according to an adaptation mechanism.
2. *User interface Scripts.* These scripts relate to the implicit or explicit data exchanges between two or several

interactive components having a common interactive ancestor. For example, an interactive object is activated or deactivated depending on the state of another object. The verification of a login+password can be initiated only after both fields are properly filled in.

3. *Project scripts.* These scripts express the data exchanges between two or many interactive objects that are independent as they do not share any parent.

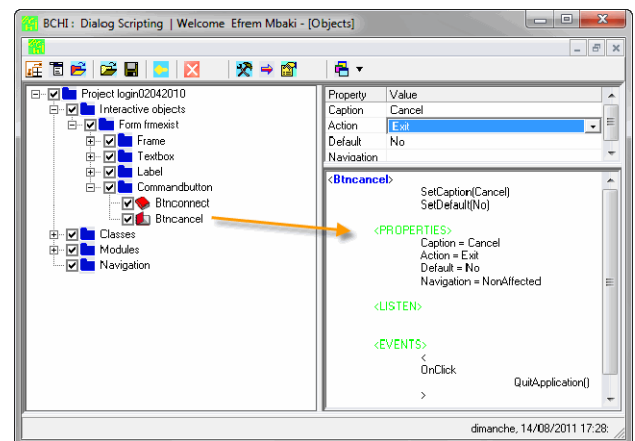


Figure 9. Definition of a dialogue script.

Any dialogue script is structured into three parts (Figure 9): a *condition* of realization, the *event* to consider and a *list of actions* to be undertaken when the event is fired and the condition is satisfied. A single script language has been defined in common for all the three types of dialogue scripts. These scripts use in harmony three models of dialogues; transition networks, grammars, and events [13]. Scripts of dialogues at the abstract and concrete levels are written with a generic language that we described using a BNF grammar. At final level, the code generator translates from generic scripting to specific language relative to a target model. It should be noted that some of these scripts are automatically deduced through some attribute values.

A simple example is to associate the exit of an interactive task with the click of a button. Such a script is generated automatically. As in useML editor [21], other scripts are derived semi-automatically. Indeed, by combining the event of an interactive object to a function call, the developer will have to make the links between the function parameters (input and output) with the attributes of interactive objects. Then, the editor automatically builds the script.

**History.** A *history* consists of a set of time-stamped operations applied to dialogue scripts over time in order to preserve the design history. In this way, some traceability of dialogue scripts (i.e., who created, retrieved, updated, deleted which dialogue script over tie in the same project) and some reusability (i.e., copy/paste an already existing dialogue script) are ensured (Figure 10). Any dialogue script definition can be validated for a particular toolkit.



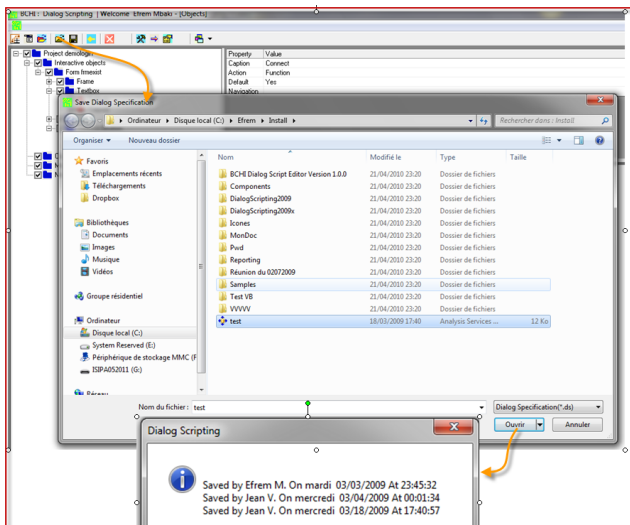


Figure 10. Recovering a previously saved history.

### MODEL-DRIVEN ENGINEERING OF DIALOGUES

#### The four main phases of model-driven engineering

In order to achieve the goal of Model-Driven Engineering of dialogues for multi-platform GUIs based on the conceptual model of Figure 2, the process supported by the BCHI Dialogue Editor (Figure 12) is decomposed into four main phases (Figure 11): (i) *project editing* includes all facilities required to create, retrieve, update, and delete any UI project during the development life cycle; (ii) *project transforming* is aimed at supporting the creation of new mappings between levels and applying them via a mapping editor (Figures 6,7,8 provide significant examples of mappings that support AUI to CUI reification or others), respectively the transformation engine; (iii) *scripting* is aimed at specifying any desired dialogue script at any time, before or after transformation; (iv) *code generating* calls the mappings corresponding to the target platform for which the code of the FUI should be produced.

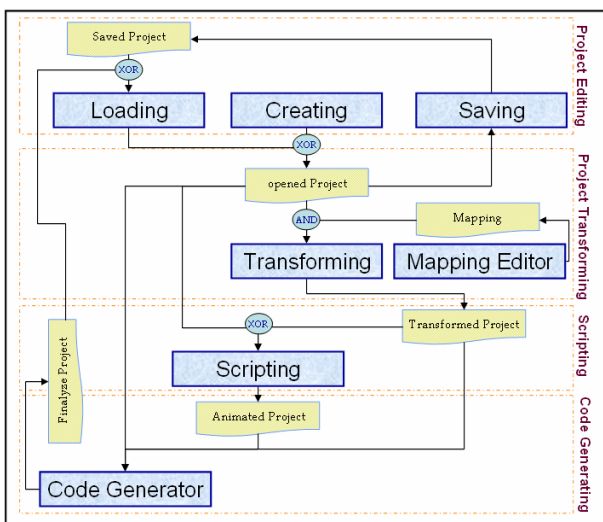


Figure 11. Four phases of dialogue model-driven engineering.

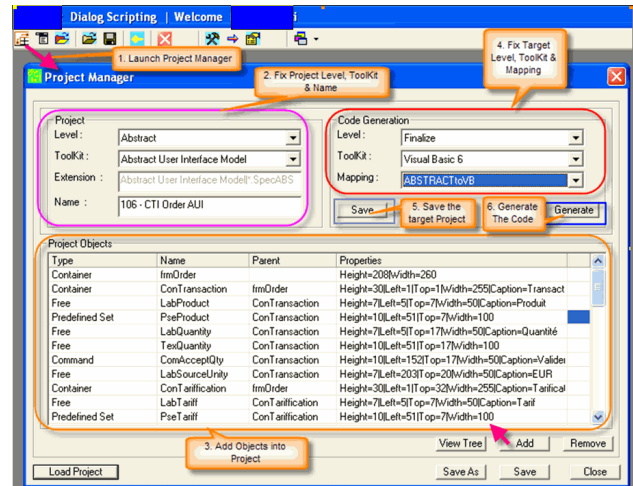


Figure12. The Dialogue Editor for model-driven engineering.

#### Applying model-driven engineering on a simple case

In this sub-section, an overview is provided of how the four main phases of model-driven engineering are applied on a running example (i.e., the login+password – Figure 13), whose simplicity has been selected for fostering understandability.

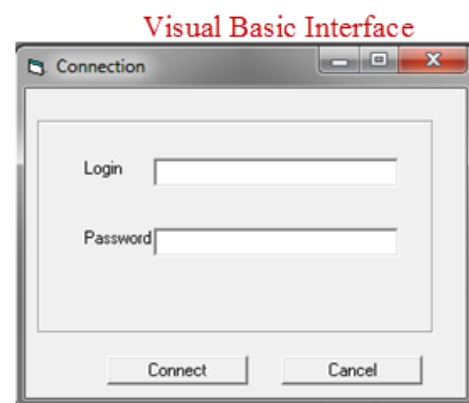
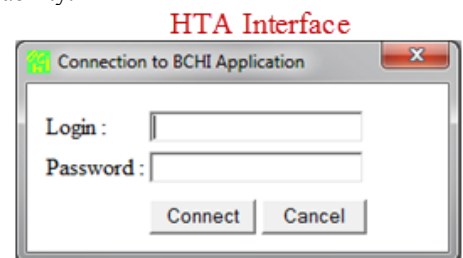


Figure 13. Final User Interfaces of Login+Password.

*Project editing.* Figure 12 explains the main first steps for creating a new UI Project in the BCHI Dialogue Editor, which basically consists of choosing the starting level of abstraction (typically, the AUI), the ending level (typically, one FUI), the toolkit, possibly with some extension, and the library of mappings to be used. Note that one can start

also at any other level such as FUI or CUI since multiple types of mappings are supported.

For the login+password example, we limit ourselves to use five properties: two properties (i.e., *left* and *top*) determine the location of each interactive object, two others properties (i.e., *height* and *width*) specify the dimensions of each interactive object and a fifth property (i.e., *label*) gives the object label text.

The values of these attributes are taken into account during future transformations. Therefore, the resulting UI Project holds the login+password with a quintuplet  $\langle \text{Label, Left, Top, Height, Width} \rangle$  for each interactive object (Table 1).

IO Name	Parent	Description	Type	Properties
FrmExist		Main Form	Container	$\langle \text{Connect, 3615,60,450,6360} \rangle$
fraIdent	frmExist	Secondary Form	Container	$\langle \text{Identification, 2295,120,240,6135} \rangle$
lbLogin	fraIdent	Login invitation	Free	$\langle \text{Login,300,480,480, 1000} \rangle$
txtLogin	fraIdent	Login contain	Free	$\langle \text{,300,1200,480,2535} \rangle$
lbPwd	fraIdent	Password invitation	Free	$\langle \text{Pass-word,300,480,1200,1000} \rangle$
txtPwd	fraIdent	Password contain	Free	$\langle \text{,300,1200,1200,2535} \rangle$
btnOk	frmExist	Validation Trigger	Command	$\langle \text{Connect, 300,3000,2880,1455} \rangle$
btnCancel	frmExist	Cancel Trigger	Command	$\langle \text{Cancel, 300,4800,2880,1455} \rangle$

**Table 1. Interactive objects of the login+password example.**

**Project transforming.** Let us assume that we want to apply Model-to-Model transformation (M2M) from AUI to CUI. For this purpose, Table 2 lists some mappings that have been implemented for this purpose, here for a vocal UI and a GUI, both appearing at the CUI level: *Container* is translated to questionnaire/Form if its name begin by *frm* or SubQuestionnaire/SubForm if its name begin by *fra*. Free object change to Request/Label if its name begin by *lb* or to Answer/Text Box if its name begin by *txt*.

Command object is expressed as verbal validation or a button depending on the interaction modality. By applying the mappings for a GUI, we obtain a CUI with a graphical modality.

**Code generating.** In order to transform this CUI into a FUI (say here that we want both the VB6 and HTA GUIs), Table 3 lists some mappings that have been implemented.

Abstract UI	Vocal UI	Graphical UI
Container	frm* $\rightarrow$ Questionnaire fra* $\rightarrow$ Sub Questionnaire	frm* $\rightarrow$ Form fra* $\rightarrow$ Sub Form
Free	Lb* $\rightarrow$ Request Txt* $\rightarrow$ Answer	Lb* $\rightarrow$ Label Txt* $\rightarrow$ Text
Command	Validation	Button

**Table 2. Mapping from Abstract to Concrete.**

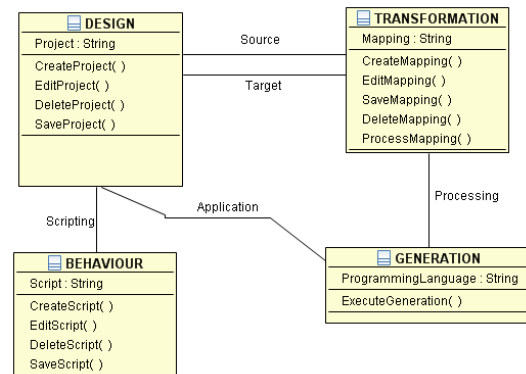
Graphic	Visual Basic	HTA
Form/Sub form	frm* $\rightarrow$ Form fra* $\rightarrow$ Frame	frm* $\rightarrow$ Form/Page fra* $\rightarrow$ FieldSet
Text/Label	Lb* $\rightarrow$ Label Txt* $\rightarrow$ Textbox	Lb* $\rightarrow$ Input (Text) Txt* $\rightarrow$ Input (Text or Password)
Command	CommandButton	Button
Ratio	1	0.05

**Table 3. Mappings from Concrete to Final User Interface.**

## DESCRIPTION OF THE BCHI DIALOGUE EDITOR

### Software architecture of the BCHI Dialogue Editor

The software architecture of BCHI Dialog Editor is composed of four components depicted in the UML Class Diagram of Figure 14 corresponding to the four phases of model-driven engineering of dialogues (Figure 11): the *Design* meta-class supplies facilities to edit any UI project (e.g., create, read, update, delete) during the project editing (Figure 11, first swim lane); the meta-class *transformation* manages mappings as defined in Figure 2 (e.g., it enables transforming a UI project of a given toolkit towards another one, possibly the same) in order to support project transforming (Figure 11, second swim lane); the meta-class *Behaviour* manages and interprets scripts at any of the three levels for the scripting (Figure 11, third swim lane); the meta-class *generation* parses any dialogue script, validates it, and transforms into code for a particular target platform in order to support the code generating (Figure 11, fourth swim lane).



**Figure 14. UML Class Diagram of the Dialogue Editor.**



## Implementation of the BCHI Dialogue Editor

The BCHI Dialogue Editor has been entirely programmed with Visual Studio 6 (Basic 6-VB6) and Visual Basic for Applications (VBA). To standardize the GUI look & feel produced by the BCHI Dialog Editor interfaces, an OCX component has been developed for every interactive object at any level (e.g., Input/output, TextBox, Combox, Check-Box). These OCX components are gathered in a library that is used in the videos demonstrating how the BCHI Dialogue Editor could be used to generate multi-platform dialogues are available on YouTube (Table 4).

Description	URL
Global View	<a href="http://www.youtube.com/watch?v=x3CtCj47iZQ">http://www.youtube.com/watch?v=x3CtCj47iZQ</a>
Architecture	<a href="http://www.youtube.com/watch?v=Nx3d-w19Oug">http://www.youtube.com/watch?v=Nx3d-w19Oug</a>
Project Editing	<a href="http://www.youtube.com/watch?v=GRKWwq5cOzU">http://www.youtube.com/watch?v=GRKWwq5cOzU</a>
Mappings	<a href="http://www.youtube.com/watch?v=gVQ8bz9wEXY">http://www.youtube.com/watch?v=gVQ8bz9wEXY</a>
Scripting	<a href="http://www.youtube.com/watch?v=EZGtL7fXtUE">http://www.youtube.com/watch?v=EZGtL7fXtUE</a>
Code Generation	<a href="http://www.youtube.com/watch?v=n7YlgpDihtY">http://www.youtube.com/watch?v=n7YlgpDihtY</a>

Table 4. Video demonstrations of the BCHI Dialogue Editor.

The underlying conceptual model of dialogues has been implemented as UML V2.0 class diagram in Moskitt (www.moskitt.org) (Figure 2) that gave rise to a XML Schema according to a systematic procedure from Moskitt. Based on this XML Schema, the conceptual model of Figure 2 is stored and maintained in the BCHI Dialogue Editor through RecordSet internal data structures from which an XML file could be exported and to which a XML file could be imported.

A RecordSet has been implemented for both native objects (Figure 15) and user objects. The UIDL that is maintained by the BCHI Dialogue Editor is therefore based on this XML Schema. Therefore, any project created in the editor is compliant with the XML Schema (Figure 16).

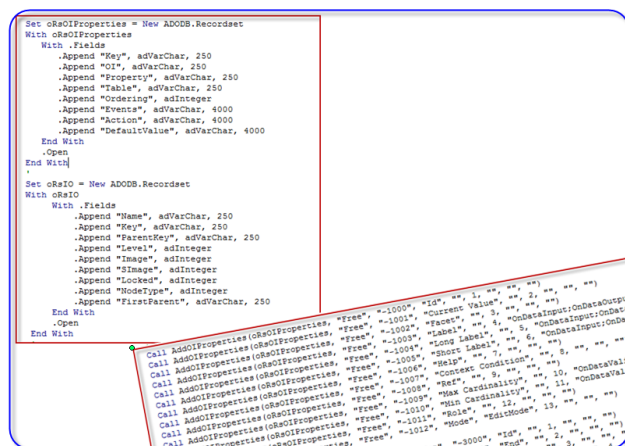


Figure 15. A RecordSet for native objects.



Figure 16. An excerpt of a XML file corresponding to a UI Project according to the UIDL of the BCHI Dialog Editor.

## CONCLUSION

This paper introduced an approach for conducting Model-Driven Engineering of dialogues for multi-platform GUIs that are compliant with the CRF [6]. For this purpose, a BCHI Dialogue Editor has been implemented that ultimately automatically generate code for four different targets (i.e., HTML V4.0, HTA, VBA V6.0, and DotNet V3.5) for two different computing platforms (Windows 7 and MacOS X) as a proof-of-concept. The main originality of this editor relies in its capability to always maintain a correspondence between native objects (belonging to the targets) and user objects (at GUI and CUI levels) and to support four types of mappings (i.e., forward, reverse, lateral, adaptation) possibly between two consecutive levels or not (cross-cutting). The BCHI Dialogue Editor however only holds mappings for GUIs only, although interactive objects have been introduced for addressing Vocal User Interfaces. Future work will be dedicated towards this goal and to integrate the conceptual model of dialogue into UsiXML V2.0 in an adequate way.

## ACKNOWLEDGMENTS

The authors would like to acknowledge the support of the ITEA2-Call3-2008026 UsiXML (User Interface extensible Markup Language – www.itea2.usixml.org) European project and its support by Région Wallonne DGO6 as well as the FP7-ICT5-258030 SERENOA (Multidimensional context-aware adaptation of Service Front-ends) project supported by the European Commission.

## REFERENCES

1. Ariav, G. and Calloway, L.-J. Designing conceptual models of dialog: A case for dialog charts, *SIGCHI Bulletin*, 20, 2 (1988) 23–27.
2. Bastide, R. and Palanque, P. A Visual and Formal Glue Between Application and Interaction. *Journal of*

- Visual Language and Computing*, 10, 5 (October 1999) 481–507.
3. Book, M., Gruhn, V., and Richter, J. Fine-grained specification and control of data flows in web-based user interfaces. In *Proc. of ICWE'2007* (Como, 16-20 July 2007). LNCS, Vol. 4607, Springer-Verlag, Berlin, 2007, 167–181.
  4. Breiner, K., Maschino, O., Görlich, D., Meixner, G. Towards automatically interfacing application services integrated in a automated model based user interface generation process. In *Proc. of MDDAUI'2009*.
  5. Cachero, C., Melia, S., Poels, G., and Calero, C. Towards improving the navigability of Web Applications: a model-driven approach. *European J. of ISs*, 16 (2007) 420–447.
  6. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J. A Unifying Reference Framework for Multi-Target User Interfaces. *Interacting with Computers* 15, 3 (2003) 289–308.
  7. Cantera, J.M., González Calleros, J.M., Meixner, G., Paternò, F., Pullmann, J., Raggett, D., Schwabe, D., Vanderdonckt, J. Model-Based UI XG Final Report. W3C Incubator Group Report, 4 May 2010. Available at: <http://www.w3.org/2005/Incubator/model-based-ui/XGR-mbui/>
  8. Carr, D. Specification of interface interaction objects. In *Proc. of CHI'94*. ACM Press, New York, 1994.
  9. Clerckx, T., Van den Bergh, J., and Coninx, K. Modeling Multi-Level Context Influence on the User Interface. In *Proc. of PERCOMW'2006*. IEEE Press, 2006, pp. 57–61.
  10. Cowan, D. and Pereira de Lucena, C. Abstract Data Views: An Interface Specification Concept to Enhance Design for Reuse. *IEEE Trans. on Soft. Eng.* 21,3 (1995) 229–243.
  11. Dittmar, A. and Forbrig, P. The Influence of Improved Task Models on Dialogues. In *Proc. of CA-DUI'2004*, pp. 1–14.
  12. Elwert, T. Continuous and Explicit Dialogue Modeling. In *Proc. of EA-CHI'96*.
  13. Green, M. A Survey of Three Dialogue Models. *ACM Transactions on Graphics*, 5, 3 (July 1986) 244–275.
  14. Harel, D. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8 (1987) 231–274.
  15. Helms, J., Schaefer, R., Luyten, K., Vermeulen, J., Abrams, M., Coyette, A., Vanderdonckt, J. *Human-Centered Engineering with the User Interface Markup Language*. In “Human-Centered Software Engineering”, Chapter 7, HCI Series, Springer, London, 2009, pp. 141–173.
  16. Jacob, R.J.K. A specification language for direct manipulation user interfaces. *ACM Transactions on Graphics*, 5, 4 (1986) 283–317.
  17. Luyten, K., Clerckx, T., Coninx, K., and Vanderdonckt, J. Derivation of a Dialog Model from a Task Model by Activity Chain Extraction. In *Proc. of DSV-IS'2003*. LNCS, Vol. 2844, Springer-Verlag, Berlin, 2003, pp. 203–217.
  18. Mbaki, E., Vanderdonckt, J., Guerrero, J., and Winckler, M. Multi-level Dialog Modeling in Highly Interactive Web Interfaces. In *Proc. of IWWOST'2008*, CEUR Workshop Proc., Vol. 445, 2008, pp. 38–43.
  19. Menkhous, G. and Fischmeister, S. Dialog Model Clustering for User Interface Adaptation. In *Proc. of ICWE'2003*. LNCS, Vol. 2722, Springer-Verlag, 2003, pp. 194–203.
  20. Meixner, G., Görlich, D., Breiner, K., Hußmann, H., Pleuß, A., Sauer, S., Van den Bergh, J. Proc. of 4<sup>th</sup> Int. workshop on model driven development of advanced user interfaces. MDDAUI'2009. In *Proc. of IUI 2009*, pp. 503–504.
  21. Meixner, G., Seissler, M., Nahler, M., Udit – A Graphical Editor for Task Models. In *Proc. of MDDAUI'2009*.
  22. Paternò, F., Santoro, C., and Spano, L.C. MARIA: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. *ACM Trans. Comput.-Hum. Interact.* 16, 4 (November 2009)
  23. Pleuß, A. Modeling the User Interface of Multimedia Applications. In *Proc. of MoDELS 2005*, pp. 676–690.
  24. Pleuß, A. MML: A Language for Modeling Interactive Multimedia Applications. In *Proc. of ISM'2005*, pp. 465–473.
  25. Reichart, D., Dittmar, A., Forbrig, P., and Wurdel, M. Tool Support for Representing Task Models, Dialog Models and User-Interface Specifications. In *Proc. of DSV-IS'2008*. LNCS, Vol. 5136, Springer, Berlin, 2008, pp. 92–95.
  26. Rückert, J. and Paech, B. The Guilet Dialog Model and Dialog Core for Graphical User Interfaces. In *Proc. of EIS'2008*. LNCS, Vol. 5247, Springer, 2008, pp. 197–204.
  27. Schaefer, R., Bleul, S., and Müller, W. Dialog Modeling for Multiple Devices and Multiple Interaction Modalities. In *Proc. of TAMODIA'2006*. Lecture Notes in Computer Science, Vol. 4385, Springer-Verlag, Berlin, 2007, pp. 39–53.
  28. Schaefer, R., Bleul, S., and Müller, W. A Novel Dialog Model for the Design of Multimodal User Interfaces. In *Proc. of EHCI-DSV-IS'2004*. LNCS, Vol. 3425. Springer-Verlag, Berlin, 2005, pp. 221–223.
  29. Traetteberg, H. Dialog modelling with interactors and UML Statecharts. In *Proc. of DSV-IS'2003*. LNCS, Vol. 2844, Springer-Verlag, Berlin, 2003, pp. 346–361.
  30. Traetteberg, H. Integrating Dialog Modeling and Domain Modeling – the Case of DIAMODL and the Eclipse Modeling Framework. *JUCS* 14, 19 (2008),

- 3265–3278.
31. Van den Bergh, J. and Coninx, K. From Task to Dialog model in the UML. In *Proc. of Tamodia'2007*, pp. 98–111.
  32. van Welie, M., van der Veer, G.M.C., and Eliëns, A. Usability Properties in Dialog Models. In *Proc. of DSV-IS'99*.
  33. Vanderdonckt, J., Limbourg, Q., Florins, M., Deriving the Navigational Structure of a User Interface. In *Proc. of 9<sup>th</sup> IFIP TC 13 Int. Conf. on Human-Computer Interaction INTERACT'2003* (Zurich, 1-5 September 2003), M. Rauterberg, M. Menozzi, J. Wesson (Eds.), IOS Press, Amsterdam, 2003, pp. 455-462.
  34. Winckler, M. and Palanque, P. StateWebCharts: A formal description technique dedicated to navigation modelling of web applications. In *Proc. of DSV-IS'2003*. LNCS, Vol. 2844, Springer-Verlag, Berlin, 2003, pp. 61–76.
  35. Winckler, M., Trindade, F., Stanciulescu, A., and Vanderdonckt, J. Cascading Dialog Modeling with UsiXML. In *Proc. of 15<sup>th</sup> Int. Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'2008* (Kingston, July 16-18, 2008). Lecture Notes in Computer Sciences, vol. 5136. Springer, Berlin, 2008, pp. 121-135.
  36. W3C State Chart XML (SCXML), State Machine Notation for Control Abstraction, Working Draft, 16 May 2008. Accessible at <http://www.w3.org/TR/SCXML>.

# Inspecting Visual Notations for UsiXML Abstract User Interface and Task Models

Ugo Sangiorgi<sup>1</sup>, Ricardo Tesoriero<sup>1,2</sup>, François Beuvs<sup>1</sup>, Jean Vanderdonckt<sup>1</sup>

<sup>1</sup>Louvain Interaction Laboratory, Louvain School of Management, Université catholique de Louvain  
Place des Doyens, 1 – B-1348 Louvain-la-Neuve (Belgium)

{ugo.sangiorgi, ricardo.tesoriero, francois.beuvs, jean.vanderdonckt}@uclouvain.be

<sup>2</sup>University of Castilla-La Mancha. Computing Science Department

Av. España S/N. (02071) Campus Universitario de Albacete. Albacete, Spain

ricardo.tesoriero@uclm.es

## ABSTRACT

In this paper, we analyze the current state of visual notations in UsiXML models for Abstract User Interface and Tasks model. A systematic analysis is presented according to Visual Variables framework – a widely considered work in graphic design field. Some directions on evolution of the visual notations are also presented.

## Author Keywords

Measurement, Design, Human Factors, Standardization, Theory.

## General Terms

Design, Human Factors, Theory

## Categories and Subject Descriptors

D2.2 [Software Engineering]: Design Tools and Techniques – *Modules and interfaces; user interfaces*. D2.m [Software Engineering]: Miscellaneous – *Rapid Prototyping; reusable software*. H.1.2 [Information Systems]: Models and Principles – *User/Machine Systems*. H5.2 [Information interfaces and presentation]: User Interfaces – *Prototyping; user-centered design; user interface management systems (UIMS)*.

## INTRODUCTION

Diagrams have the power of expressing ideas in a very effective way, because we are able to summarize concepts inside an image that would take pages to explain in textual form – textual representations are one-dimensional (linear), while visual representations are two-dimensional (spatial) [7]. Also, expressing ideas in visual and spatial medium makes comprehension and inference easier, since a quarter of the human brain is devoted to vision, which is a bigger percentage than all other senses combined [6].

It is not an easy task to design visual notations, in part due to the lack of scientific methodologies available to analyze them and to make conclusions regarding what makes a good diagram. Yet, diagrams play a critical role in all areas inside software development from requirements engineering through to maintenance.

Although the Software Engineering community has succeeded to develop methods to evaluate and design model's semantics, the same is not true to models visual syntax.

This work's aim is to do an inspection of currently available visual notations of UsiXML, mainly focusing on Abstract User Interface and Tasks models (and not yet for Concrete User Interface, Domain and Context models). The two chosen models for this analysis are more central to UsiXML than the others, and therefore are where an analysis such as the one presented on this work is expected to have more value. The driving question of the inspection is: *Are all the elements and relations of the meta-model represented on the visual notations?* We try to address it in terms of **how** to represent what is missing. Thus, this work does not intent to define what the meta-models should represent, but only whether the visual notation comprises to what is stated on the meta-model.

The next section first presents some definitions of visual notations in the domain of Software Engineering (SE), and then we present the Visual Variables framework and the conceptual basis for the analysis. The section 3 presents the analysis itself along with some recommendations of use for each variable. We conclude with some preliminary work on graphical editors, which are along with evidences that they comprise with the recommendations.

## STATE OF THE ART

In order to describe what visual notations are, some basic concepts need to be presented. The first and most basic is that diagrams are encoded as two-dimensional geometric symbolic representation of information, and this information refers to something that is outside of the diagram itself.

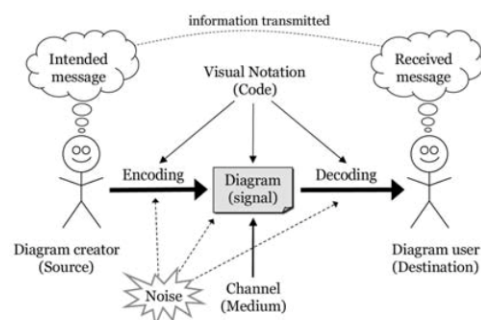


Figure 1. Theory of diagrammatic communication.  
(Source: [11]).

In general, the communication process using diagrams can be described as in Figure 1: a diagram creator (the sender) encodes information (message) in the form of a diagram (signal) and the diagram user (receiver) decodes this signal. The diagram is encoded using a visual notation (code), which defines a set of conventions that both sender and receiver understand. The medium (channel) is the physical form in which the diagram is presented (e.g., paper, white-board, and computer screen). Noise represents random variation in which the signal can interfere with communication.

The match between the **intended** and **received** messages defines the effectiveness of communication. In this sense, communication consists of two complementary processes: encoding (expression) and decoding (interpretation).

As pointed out by [11], in order to optimize communication both sides need to be considered: of the encoding: *What are the available options for encoding information in visual form?* This defines the **design space**: the set of possible graphic encodings for a given message; and of the decoding: *How are visual notations processed by the human mind?* This defines the **solution space**: principles of human information processing provide the basis for choosing among the infinite possibilities in the design space.

### Design Space

For the design space, we can account for the visual variables defined by Semiology of Graphics [1], a widely considered work in graphic design field. Like showed in Figure 2, it defines a set of atomic building blocks that can be used to construct any visual representation in the same way the periodic table can be used to construct any chemical compound.

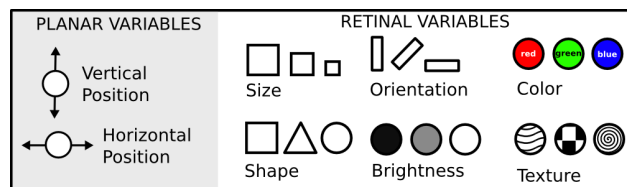


Figure 2. Visual variables.

The visual variables thus define the dimensions of the graphic design space. The visual variables also define set of primitives (a visual alphabet) for constructing visual notations: Graphical symbols can be constructed by specifying particular values for visual variables (e.g., shape = rectangle, color = green). Notation designers can create an unlimited number of graphical symbols by combining the variables together in different ways.

### Solution Space

For the solution space, we take into consideration that humans can be viewed as information processing systems [NS92], so designing cognitively effective visual notations can be seen as a problem of optimization of this processing (Figure 3).

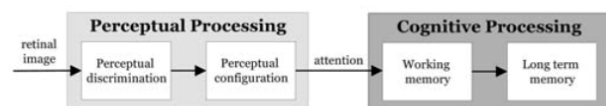


Figure 3. Information processing by the human mind. (Source: [11]).

The more evident benefit of using diagrams is perhaps the **computational offloading**, which is the shift of the processing burden from the cognitive system to the perceptual system, which is faster and frees up the scarce cognitive resources for other tasks.

The stages in human graphical processing are:

*Perceptual discrimination:* Features of the retinal image (color, shape, etc.) are detected by specialized feature detectors and based on this, the diagram is parsed into its constituent elements, separating them from the background (figure-ground segregation) [13].

*Perceptual configuration:* Structure and relationships among diagram elements are inferred based on their visual characteristics [13]. Winn had made a study to investigate

*Attention management:* All or part of the perceptually processed image is brought into working memory under conscious control of attention. Perceptual precedence determines the order in which elements are attended to [13].

*Working memory:* This is a temporary storage area used for active processing, which reflects the current focus of attention. It has very limited capacity and duration and is a known bottleneck in visual information processing [4].

*Long-term memory:* To be understood, information from the diagram must be integrated with prior knowledge stored in long-term memory. This is a permanent storage area that has unlimited capacity and duration but is relatively slow [6]. Differences in prior knowledge (expert-novice differences) greatly affect speed and accuracy of processing.

One could argue to use another frameworks such as Cognitive Dimensions [2]. However, despite of its wide use over time to evaluate all sorts of artefacts, there are several reasons for this framework does not provide a scientific basis for evaluating and designing **visual notations**. As pointed out by [11]:

- It is not specifically focused on visual notations and only applies to them as a special case (as a particular class of cognitive artefacts) [5].
- The dimensions are vaguely defined, often leading to confusion or misinterpretation in applying them [5].
- It excludes visual representation issues as it is based solely on structural properties.
- Its level of generality precludes specific predictions, meaning that it is unfalsifiable.

Physics of Notation covers the aforementioned shortcomings of Cognitive Dimensions. As it is a more theory-



grounded framework, it was the chosen one for evaluating UsiXML notations. However this paper focuses on the **design space** for its inspection, thus it begins with Visual Variables, as it is part of the Physics of Notations framework. For the **solution space** a broader analysis needs to be taken, this time based on the whole Physics of Notation framework.

### Visual Variables

The analysis uses the visual variables arranged around eight axis (one for each variable) in a way similar to Galactic Dimensions [3], but instead of comparing different languages on the same graphic, we are comparing different **versions** of the same language (current and future). Therefore, three steps are considered on each axis (Figure 4):

*Relevant / Good use*: the variable is relevant to the visual notation according to some concept on the meta-model. In other words, the concept of which the variable refers is clearly represented on the diagram.

*Ambiguous / Bad use*: the variable does not represent any concept of the meta-model but is present on the diagram, or the variable is used to represent more than one concept.

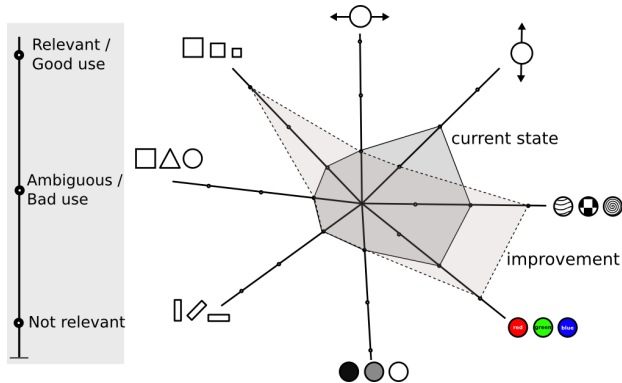


Figure 4. Visual variables and one example analysis.

*Not relevant*: If the variable is not present on the diagram thus not representing any element on the meta-model.

A visual variable can be relevant or irrelevant to the model (thus in a consistent state with the meta-model) but may never be ambiguous.

Two graphs are plotted over the axis: the darker and solid one represents the current state of the visual notation and the lighter and dotted one the improvements that can be made. For instance, if the Color variable is on the *Ambiguous* state, it is either changed to *Relevant* (the diagram can use Color to represent 'Object Type', for example) or *Not relevant* (Color should not be present, thus no suggestion is made regarding what concept to map using this variable).

### ANALYSIS

Having the frameworks presented before, this section will analyze the current state of the visual notations of UsiXML. This section, however, do not intent to define

what the meta-model should represent, but only if the visual notation comprises to what is stated on the meta-model.

Therefore, the fundamental question to be answered when evaluating the current state of UsiXML visual notations is: *Are all the elements and relations of the meta-model represented on the visual notations?* In other words, is the expressivity of the meta-model diminished or augmented onto its correspondent diagram?

This section will present the analysis of the following visual notations of the UsiXML models: Abstract, and Tasks. A visual notation can be inspected based on the visual variables that compose the design space.

### Abstract User Interface

The AUI model is an expression of the UI in terms of interaction spaces (or presentation units), independently of which interactors are available and even independently of the modality of interaction (graphical, vocal, haptic, etc) [8]. In this analysis we chose IdealXML's Abstract Model Editor [10] (Figure 5) as it is the most used visual notation for AUI, based on the high number of references on UsiXML related publications. Some other visual notations for AUI were proposed and it was based on already existing propositions (but not implemented).

An AUI defines interaction spaces by **grouping** AUIs (and implicitly tasks of the task model) according to various criteria (e.g., task model structural patterns, cognitive load analysis, semantic relationships identification).

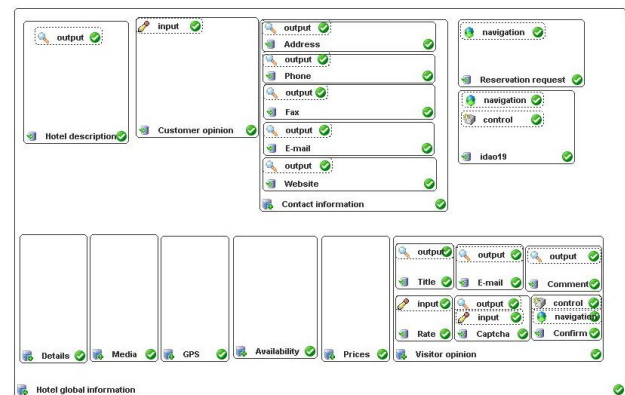


Figure 5. AUI model in IdealXML [10].

A set of abstract relationships is provided to organize AIOs in such a way that a derivation of navigation and layout is possible at the concrete level. An AUI is considered to be an abstraction of a CUI with respect to modality [8].

Based on these definitions, we can observe that the most important concepts are:

- *Grouping* – which AUIs contains other units
- *Hierarchy* – which AUIs have a higher level of importance
- *Ordering* – which AUIs have a higher level of precedence



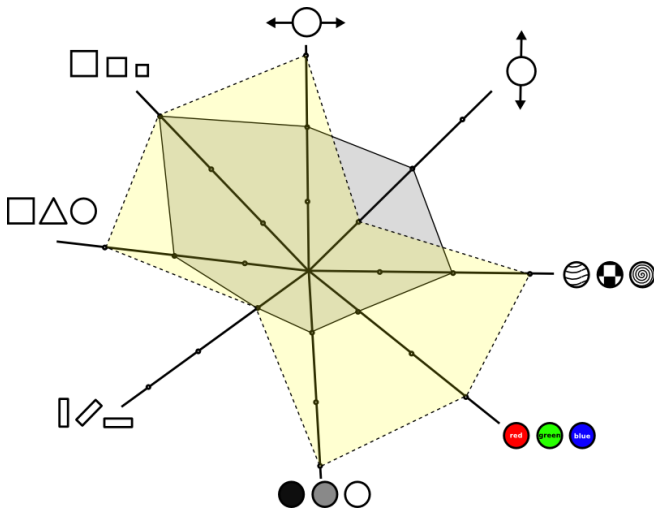
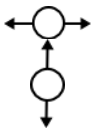

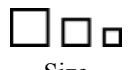




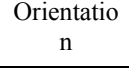


Figure 6. Abstract User Interface analysis.

Dimension	Current state analysis
	Improvement proposition
 Position	<p>As previously observed, position should not matter for an abstract user interface model, since the human brain is capable of perceiving the slightest differences on this planar variable, positioning demands attention for irrelevant details, ultimately misleading the interpretation.</p> <p>Ex: In IdealXML the small accidental differences yields unintended meaning about element's positioning. (Those differences are marked with the symbol <math>\perp</math> on the figure below, for sake of visualization. The color is removed for the same purpose.)</p> 
	<p>Fine positioning should not be considered in a visual notation that is intended to be independent of modality, for the bi-dimensional space does not exists on the abstract level. However, even though diagrams need to be represented on a bi-dimensional space, some care need to be taken in order to avoid the designer to make statements about the position of elements. One effective strategy to diminish its importance is to make the elements <b>aligned all the same</b>.</p>

 Size	Size is very relevant for representing <b>grouping</b> among the entities in the model.
	Size should indeed be relevant for representing grouping. However, in order to not give meaning to units with different heights or even to prevent the designer to worry about those details, they should have the same size.
 Shape	Shape has a big relevance in the notation used by IdealXML, since the icons's shape at the side of the labels is the <b>only</b> indication of the functionality of a given abstract unit.
	However, there is an ubiquitous check icon (✓) that is not used to represent anything in the meta-model.
 Brightness	Icons can be used as a complement, overloading the semantics yielded by other variables such as Color and Texture.
	This dimension is not used on the visual notation
 Texture	This dimension could be used to represent the <b>frequency</b> attribute, for instance.
	This dimension is not used on the visual notation
 Color	This dimension could be used instead/in complement of icons.
	This dimension is only used in the icons
 Orientation	This dimension could be used to represent the <b>importance</b> attribute, for instance.
	This dimension is not used on the visual notation
No proposition	

## Tasks

The Task model describes the interaction among tasks as viewed by the end user perspective with the system. A Task model represents a decomposition of tasks into sub-tasks linked by task relationships. In this analysis we chose CTT (ConcurrentTaskTrees) [12] (Figure 7) as it is the most used visual notation for Task modeling, based on the high number of references on UsiXML related publications.

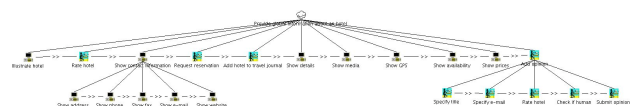


Figure 7. ConcurrentTaskTrees model.

Is important to note that in CTT (and in largely all task models) what is represented in its hierarchical diagram is not importance, but composition or **structure**. Furthermore, in CTT the sub-tasks are always related to each other through operators, which represents **behavior** as well.

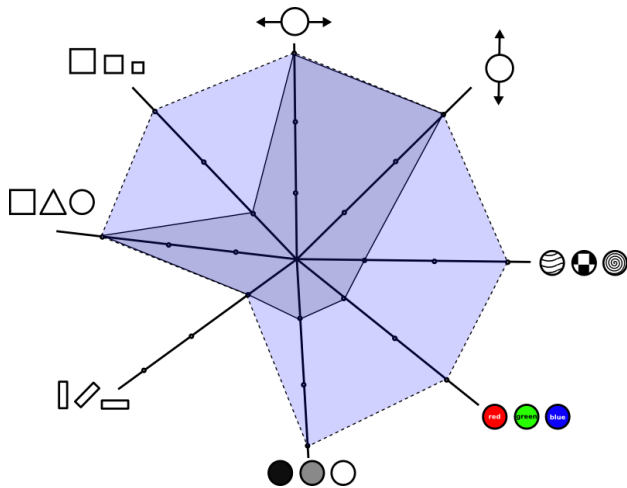
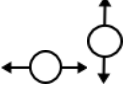
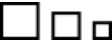












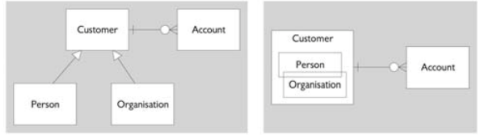




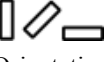


Figure 8. Tasks model analysis.

Dimension	Current state analysis				
	Improvement proposition				
 Position	<p>Vertical and horizontal positions have meaning of precedence between tasks (from left to right and from up to bottom).</p> <p>Precedence is a crucial concept on tasks models, no proposition is made to change the representation of precedence.</p>				
 Size	<p>Size has no relevance on the current model</p> <p>Winn has analyzed how people naturally interpret spatial relations in [14]. The findings were grouped in Sequence, Subclass, Intersection and Hierarchy [11]:</p> <div data-bbox="331 1527 742 1765"> <table border="1"> <tr> <td> <b>Sequence/Causality</b>   </td> <td> <b>Subclass/Subset</b>   </td> </tr> <tr> <td> <b>Intersection</b>   </td> <td> <b>Hierarchy</b>   </td> </tr> </table> </div> <p>Except for the inclusion of text describing the relationship type (  ,  &gt;,  &gt;,  &gt;,  &gt;,  &gt;,  &gt;) there is no difference between a node-leaf connection (structure relationship) and a node-node/leaf-leaf connection (behavior relation-ship).</p>	<b>Sequence/Causality</b> 	<b>Subclass/Subset</b> 	<b>Intersection</b> 	<b>Hierarchy</b> 
<b>Sequence/Causality</b> 	<b>Subclass/Subset</b> 				
<b>Intersection</b> 	<b>Hierarchy</b> 				

	<p>ship).</p> <p>Size is relevant for representing structural composition, while connections are more fitted to represent sequence/causality.</p> <p>In the image below, spatial enclosure and overlap (right side) convey the concept of overlapping subtypes in a more semantically transparent way than connecting lines [11].</p> <div data-bbox="981 526 1460 660">  </div> <p>Therefore, a significant improvement would be made if a notation could represent <b>structure</b> with subclass/subset and <b>behavior</b> with sequence/ causality.</p>
 Shape	<p>The shape variable maps to the nature property of the task, with one symbol for each: NONE, USER, SYSTEM, INTERACTIVE</p> <p>No changes are suggested; each task type should be distinguishable.</p>
 Brightness	<p>Brightness has no relevance on the current model.</p> <p>Together with an eventual tool support the model entities in the diagram could have different brightness levels according to one of the following attributes:</p> <p>optional: the optional elements would have 50% brightness, the mandatory 0%;</p> <p>iterative: the less iterative tasks would be brighter than the ones with more iterations;</p> <p>criticality: the less critical tasks would be brighter than the more critical ones;</p> <p>frequency: the less frequent tasks would be brighter than the more frequent ones;</p> <p>centrality: the less central tasks would be brighter than the more central ones;</p>
 Texture	<p>Texture has no relevance on the current model.</p> <p>Texture can be used to support the task's relationship by overloading the information present in the operators (  ,  &gt;,  &gt;,  &gt;,  &gt;,  &gt;,  &gt;)</p>
 Color	<p>Color has no relevance on the current model.</p> <p>Color can be used to complement shape, giving each task type a different color.</p>

 Orientation	Orientation has no relevance on the current model.
	No changes are suggested.

## DIAGRAMS AND TOOL SUPPORT

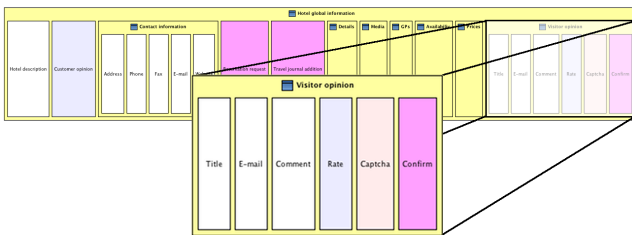
We believe tool support to be important to make UsiXML popular and foster model usage. For this purpose, there is work in progress on making the proposed visual notations usable with editors.

The propositions made by this work cannot be fully applied without proper tool support, for which we are developing a set of editors for each model in order to provide an integrated environment based on Eclipse. This platform was chosen because it is a multi-platform environment already well-known from the developers and provides a set of frameworks allowing developing modeling tools such as EMF, GMF, ... Moreover, it supports modeling standards defined by the OMG (Object Management Group).

The current state of the proposed diagrams is presented on the next section together with their tool support. As it is a work in progress, not all the improvement suggestions are implemented with the tools.

### AUI Diagram

The preliminary proposition for Abstract UI model is shown on Figure 9 (modeled for the same scenario of Figure 5)

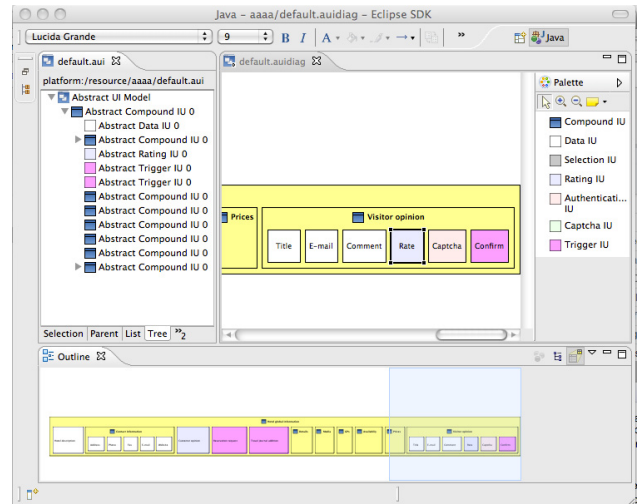


**Figure 9. Preliminary proposition for Abstract UI, with different colors for each AIU type.**

Since the bi-dimensional space does not exist on the abstract level, the diagram should not let the designer to specify position constraints.

The AUI notation in this work was designed in order to be truly independent of modality on the visual level as well (not only in the meta-model level). The tool support for AUI is showed on Figure 10, as an Eclipse integrated graphical editor. The left side shows the model being edited graphically on the right side; there is a palette with the elements from the meta-model to be dragged to the diagram.

All the elements are self-organized so the designer does not need to worry about the fine positioning, only on the **order** of the elements (horizontal positioning), as recommended by the analysis.



**Figure 10. Abstract UI editor on Eclipse.**

### Task Diagram

As said in the analysis, a significant improvement would be made if a notation could represent **structure** with subclass/subset and **behavior** with sequence/causality. In this sense, with proper tool support the designer would be able to switch between both views (structural and behavioral) on the same model.

Therefore, the Task model proposition is presented in Figure . It is similar to [9] in respect to the structure, since it represents a tree structure without connecting lines, but instead using grouping. However present on the meta-model level, the notation still doesn't visually shows the task type (system, user, abstract, interactive).

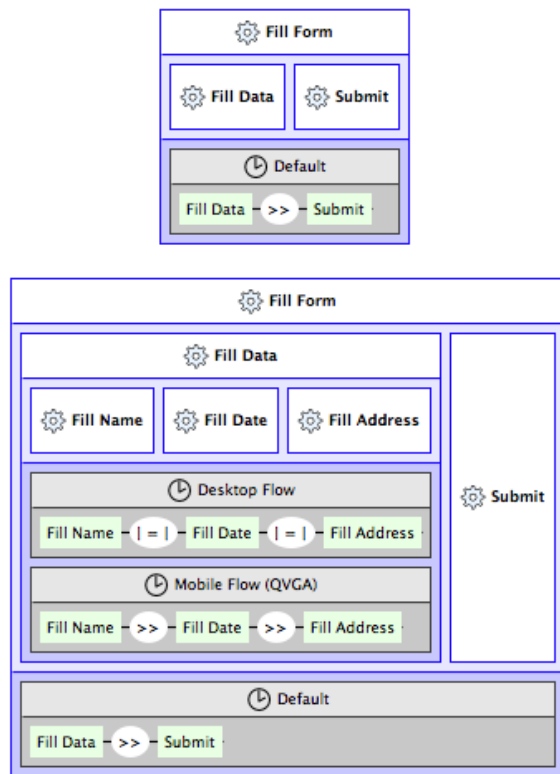
With this notation is possible to define alternative paths (or Temporalizations) for tasks depending on context situations. In the upper part of Figure 11 we show a task *Fill Form* with two subtasks *Fill Data* that enables *Submit*.

In the bottom part we show the same tasks, except that *Fill Data* has three other subtasks. Inside *Fill Data* two different flows are presented, for *Desktop* – in which the order of field filling does not matter; and *Mobile* – in which the order must be strict, since it is constrained by the device's screen size.

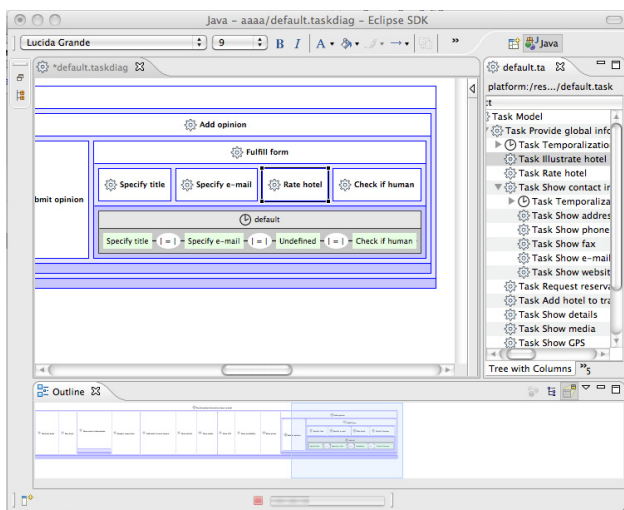
## CONCLUSION

This work aimed to present an analysis of currently available visual notations of UsiXML models in terms of a popular framework in graphic design.

By reviewing the driving question (*Are all the elements and relations of the meta-model represented on the visual notations?*) we can observe that:



**Figure 11. Preliminary Task Model with simplified and detailed visualizations.**



**Figure 12. Task Model editor in Eclipse.**

- In IdealXML not all the concepts on the meta-model are reflected in the diagrams. We have presented some possibilities of representation using visual variables;
- Vertical Position and Shape were ambiguous on IdealXML's notation. This work proposed solution for this problem;

- The Task model is more semantically transparent [11] because it uses components instead of lines to represent subsets of tasks;
- Variables such as Texture and Brightness can be used to represent importance, criticity or frequency.

As future works, as said in Section 2, this paper focused only on the design space. An inspection for the solution space needs to be taken, based on Physics of Notation [11].

Also, the remaining improvement suggestions are to be included on the notations such as *Shape*, *Brightness* and *Texture* for AUI and *Brightness*, *Color* and *Texture* for Tasks.

### ACKNOWLEDGMENTS

The authors would like to acknowledge of the ITEA2-Call3-2008026 UsiXML (User Interface extensible Markup Language) European project and its support by Région Wallonne DGO6.

### REFERENCES

1. J. Bertin, *Semiology of graphics*, University of Wisconsin Press, 1983.
2. A. Blackwell and T. Green, "Notational systems—the cognitive dimensions of notations framework," *HCI Models, Theories and Frameworks: Toward a multi-disciplinary science*, 2003, p. 103–134.
3. P. Campos and N.J. Nunes, "Galactic dimensions: A unifying workstyle model for user-centered design," *Human-Computer Interaction-INTERACT 2005*, 2005, p. 158–169.
4. G.L. Lohse, "The Role of Working Memory in Graphical Information Processing," *Behaviour and Information Technology*, vol. 16, 1997, pp. 297–308.
5. T. Green, a Blandford, L. Church, C. Roast, and S. Clarke, "Cognitive dimensions: Achievements, new directions, and open questions," *Journal of Visual Languages & Computing*, vol. 17, Aug. 2006, pp. 328–365.
6. S.M. Kosslyn, "Graphics and human information processing: A review of five books," *Journal of the American Statistical Association*, vol. 80, Dec. 1985, p. 499–512.
7. J. Larkin and H. Simon, "Why a Diagram is (Sometimes) Worth Ten Thousand Words," *Cognitive Science*, vol. 11, 1987, pp. 65 - 100.
8. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., López Jaquero, V. UsiXML: a Language Supporting Multi-Path Development of User Interfaces. In *Proc. of 9th IFIP Working Conf. on Engineering for Human-Computer Interaction. EHCI-DSVIS'2004*, Springer, 2005, pp. 200–220.
9. F. Martinez-Ruiz, J. Vanderdonckt, and J.M. Arteaga, "TRIAD: Triad-based Rich Internet Application De-

- sign,” *1st Int Workshop on User Interface Extensible Markup Language UsiXML (2010)*, vol. 2010, 2010.
10. F. Montero, M. Lozano, and P. González, *IDE-ALXML: an Experience-Based Environment for User Interface Design*, Albacete: Citeseer, 2005.
  11. D. Moody, “The ‘Physics’ of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering,” *IEEE Transactions on Software Engineering*, vol. 35, 2009, pp. 756-779.
  12. F. Paterno, “Model-based Design of Interactive Applications,” *intelligence*, vol. 11, 2000, p. 26–38.
  13. S. Palmer and I. Rock, “Rethinking perceptual organization: The role of uniform connectedness,” *Psychonomic Bulletin & Review*, vol. 1, 1994, p. 29–55.
  14. W. Winn, “Encoding and retrieval of information in maps and diagrams,” *IEEE Transactions on Professional Communication*, vol. 33, 1990, pp. 103-107.

# Adaptive Dialogue Management and UIDL-based Interactive Applications

Frank Honold, Mark Poguntke, Felix Schüssel, Michael Weber

Institute of Media Informatics, Ulm University

89081 Ulm, Germany

{frank.honold, mark.poguntke, felix.schuessel, michael.weber}@uni-ulm.de

## ABSTRACT

Different approaches exist to describe user interfaces for interactive applications and services in a model-based way using User Interface Description Languages (UIDLs). These descriptions can be device and platform independent and allow adaptivity to the context of use, although this adaptivity has to be predefined. In this paper we motivate the use of UIDLs in a broader view: As the basis for building adaptive, flexible and reliable systems using adaptive dialog management. The goal is to provide the user with the right service and the right interface at the right time. We present different requirements for adaptive models with UIDLs and discuss future work to achieve this vision.

## Author Keywords

Adaptivity, dialogue modeling, multimodality, fusion, UML, User Interface Description Language, requirements.

## General Terms

Design, Human Factors, Theory

## Categories and Subject Descriptors

H.5.2 [Information Interfaces and Presentation]: User Interfaces – *User-centered design*.

## INTRODUCTION

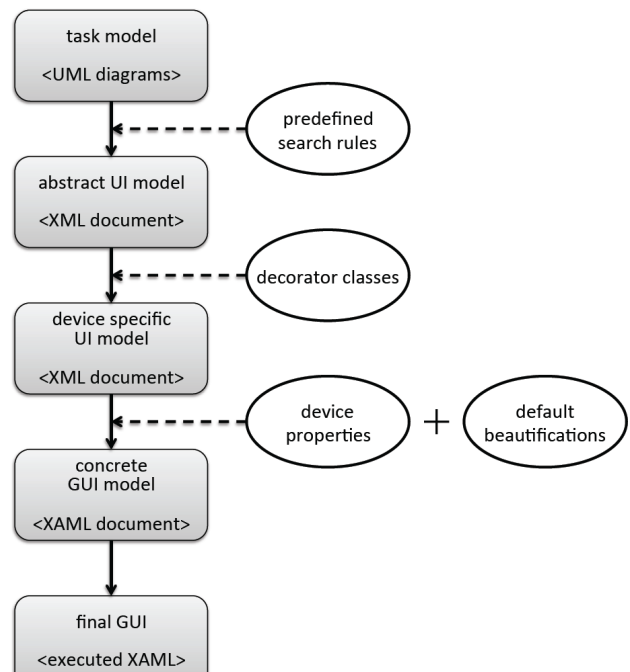
In the vision of ubiquitous computing different services do not only work separately for strictly defined use cases. The user may have a goal that requires the combination of different services. This may not be planned in advance by the individual service providers. Also, different users may prefer different modalities and may approach their goal in various ways. An adaptive interface for different devices and services to the user is desired.

With UIDLs abstract definitions of user interfaces can be achieved to allow a more flexible adaptation to different devices and modalities. However, the adaptation of a user-system dialogue is only possible if all context dependencies and the respective adaptations were defined in advance. We present requirements to achieve a flexible basis for adaptive dialogue management with flexible UIs based on existing approaches for UIDLs.

Major challenges are exemplified with a scenario for the combination of different services to achieve a user's goal.

## User Interface Modeling

Model-based methods are wide-spread for the development process within a software project. The Unified Modeling Language (UML) provides a standard notation that has proven its applicability for modeling different aspects of software in many research and industry projects. For modeling user interfaces (UI) no de facto standard exist. However, model-based user interface development holds the advantages of reusability, readability and easier adaptability of UIs amongst others [14]. Several approaches provide notations that are independent of any concrete implementation and can be used during the process of user interface development. Reported approaches allow user interface designers to specify interaction on a very high-level UIDL without any details on particular input and output devices or platform information. Examples for UIDLs include UsiXML [19], TERESA XML [15] and UIML [17]. These are special notations and require new modeling tools or plug-ins to existing tools to work with them.



**Figure 1.** Transformation steps and involved information to derive a XAML GUI from an abstract task description in UML.



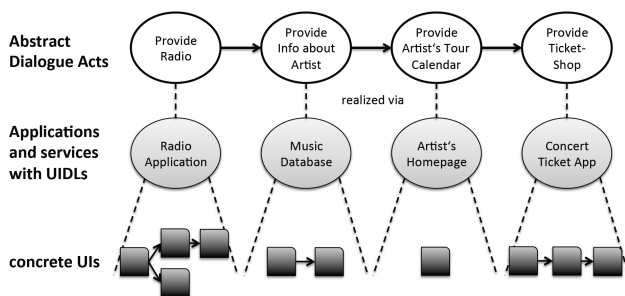
Several researchers motivate the use of UML for user interface modeling [5,6,7]. De Melo analyzed different user interface modeling approaches and defined decision criteria. UML was identified as most flexible and extendable approach with best tool support [6]. Our approach in [12] builds the basis for further considerations concerning adaptivity. We use the Cameleon Reference Framework [3] and start with a task model created with UML. Figure 1 illustrates the required transformation steps to derive the final UI. Based on pre-defined search criteria UML structures representing specific interaction schemas are then found and transformed to an abstract UI model described in XML. This is used to derive a concrete executable UI model. We illustrated this with the model of a radio application and the derivation of different XAML-based concrete user interfaces [12].

### SCENARIO

We introduce the need for adaptive dialogue management with the following scenario:

The user is listening to a song played in a *radio application* on their smartphone. The song seems familiar and the user wants to get some *info on the artist*. The user wants to get more information on *tour dates* of this artist. Browsing the tour calendar it shows *a concert in the user's home city* in a few months. Finally, the user decides to *buy a ticket for this concert*.

For all these different tasks a number of services exist the user would have to use separately to finally get the concert ticket. An intelligent system could use adaptive dialogue management to assist the user in succeeding their goal by automatically offering the different services when needed. This can be facilitated by the use of UIDLs providing abstract interfaces to the services. Figure 2 illustrates this within our given scenario.



**Figure 2. Realizing abstract dialogue acts for the example scenario using different UIDL based applications.**

Furthermore these interfaces could even be adapted to available devices and modalities in a given context. In the following, adaptive dialogue management is introduced. Then, requirements for adaptive dialog management for UIDL-based interactive applications are derived.

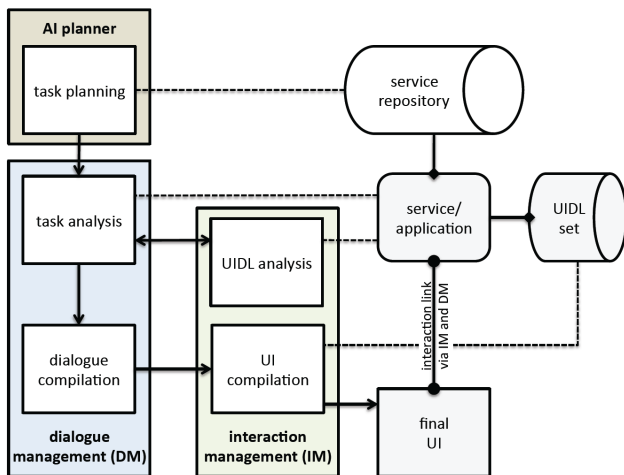
### ADAPTIVE DIALOGUE MANAGEMENT

Continually available systems like smartphones or in-car infotainment systems are ubiquitous in nowadays society and their individual success is to a great part based on user-friendly interface design and flexible integration of diverse applications. As these systems are usually already configurable in their look and feel and may be adapted individually, the next step in this area is adaptivity at runtime to the current state of the user (e.g. the emotion, the situation, his expertise) and their surroundings. Adaptivity should not only be restricted to a single application, we call this *intra-application adaptivity*, but should span all applications available which we call *inter-application adaptivity*.

Such systems would not only combine the information gathered by multi-sensor networks, but could also extend the interaction possibilities for the user to all devices capable of interaction. This allows for a more natural way of interaction and does not constrain the user to a predefined limited set of modalities. The basis for such kind of systems is an adaptive multimodal dialogue management that is able to manage multiple applications (or parts of a single application) as well as different devices for input and output. For this purpose the underlying dialogue model needs to be at a high level of abstraction, so that dialogues can be invoked independently of their concrete user interface, depending only on their purpose for the user.

A lot of work has already been done on this topic regarding multimodal dialogue system architectures [4,9,13] and complete systems like in [11, 20]. We propose a goal based dialog management in the style of spoken language dialog systems (SLDS) that can be easily modeled using a graphical tool [1].

In our scenario a computer system assists a user in succeeding his goal. The envisioned system owns a model of the state of the world as it is and a model of how it should be. The system's AI planner owns the ability to identify and structure abstract tasks from a given service repository in that way, that it generates a planned sequence of certain tasks ordered by causal links to achieve the aforementioned goal (cf. [2]). These tasks are then passed over to the dialogue management (DM) as abstract dialogue acts, which need to be realized via a suitable user interface. This could be done with the use of UIDLs. The DM identifies all services which can realize the given task. Each service provides its own UIDL description. So in co-operation with the interaction management (IM) the DM decides for one service to be integrated. The service gets integrated into the current dialogue sequence and the IM is responsible for the UI-refinement including fission and fusion. The IM looks up all available interfaces, or even compositions of such interfaces to offer the right interface at the right time (cf. Fig. 3).



**Figure 3: Realizing adaptive dialogues. Starting from a planned task, the dialogue management in co-operation with the interaction management selects and structures suitable UIDL fragments from a UIDL set to derive concrete user interfaces for a certain dialogue (sequence).**

#### MODALITY ARBITRATION

For dynamic use in ubiquitous environments novel systems will co-operate with capable devices and make use of their different input and output components. Starting from abstract UI descriptions the interaction management (IM) shall derive an appropriate UI by reasoning about unimodal or multimodal output and the corresponding concrete UIs. Throughout the process of modality arbitration the IM has to get knowledge about the later user interface's interaction concepts, the *interaction interfaces*. The IM analyzes the UI description, and explores all capable components which support the needed interaction for input and output. The IM's fission component is capable for output organization. If not yet modeled, a concrete UI will be derived from the abstract UI description within the fission process, as motivated by [3,7]. The fusion component analyses the occurring interaction input and checks, if it could be mapped to the UI's given interaction interface. If possible, the interaction gets assigned and passed back via the dialogue management to the linked service or application.

#### REQUIREMENTS

As motivated above future systems shall compose the user interface at runtime in a very flexible way. The adaptiveness of the single UI components as well as their combination is influenced by the user model, the device and components model, the surroundings model, the task model, the available widgets, and the information which shall be communicated.

To reason on a concrete user interface rises requirements for a UIDL-concept, which offers the possibility to describe the UI on different abstraction levels. The (device- and modality-independent) abstract level for the dialogue management and the concrete description to realize the UI.

To identify suitable pre-described widgets, each widget has to provide information about its purpose, its needs for and effects of a possible applicability. To seamlessly interact with a realized UI via different interaction metaphors (speech, touch, gesture, etc.), the UI description must reveal a description of its interaction interface.

The next issue addresses multi-tasking and interaction mapping support. In our scenario we described the situation where the user listens to the music and asks for the artist. The dialogue manager handles the w-words<sup>10</sup> interaction, if the current application can not do this; here: "Who is the artist playing on the radio?" So the system identifies a new goal, the planner integrates it, because the system retrieved a suitable application. Next, the dialogue manager integrates the service at runtime and the interaction management is responsible to communicate the music database's UI towards the user, and so on (see Fig. 2).

There may be temporarily at least two concurrent applications running in parallel. Due to the UIs described interaction interfaces and the fusion's interaction mapping (cf. sec **Error! Reference source not found.**) different input components can be assigned to different applications if they support the same interaction for a user input. This goes along with some of the needs for *interaction modeling* named in [8,16].

We summarize the requirements for future UIDLs for a better applicability in dialogue management and modality arbitration as follows:

- A UIDL-concept shall allow to describe user interfaces on different levels of abstraction (including concrete and abstract layout or arranging descriptions)
- There shall be a way to describe a UI in an device- and modality-independent way
- The UIDL-concept shall allow to describe the purpose of a modeled UI
- It shall be possible to describe a UI's needs for applicability as well as its resulting effects
- UIDLs shall allow to describe interaction interfaces for seamless interaction integration with different devices and components
- UIDLs for core-interaction have to provide binding mechanisms to link the UI with the functional core
- To support interaction, event mechanisms shall be describable
- The UIDL shall allow to describe reusable components for different purposes

<sup>10</sup> who, what, when, where, why, which, whom, ...

- Information – expressed or gathered via a user interface – shall be describable or referenceable by the expressiveness of a UIDL. To allow to model an abstract UI for abstract information and a concrete UI for concrete information.

It is unlikely that a single UIDL will evolve for all kinds of applications, since there are different fields of application. According to [10,18] there are differences between UIs for “command/control oriented interfaces, where the user initiates all action” and an interface which “is more modeled after communication, where the context of the interaction has a significant impact on what, when, and how information is communicated.” Thus a system can be designed using an interaction based or an information based approach.

The requirements for multimodal *interaction modeling* (cf. [8]) are different to the one of *user interface modeling*. We understand UML as a modeling language which offers possibilities to model both. In [5] we focus on interaction modeling, whereas in [12] we focus on user interface modeling, both utilizing UML.

## CONCLUSION AND FURTHER WORK

We illustrated important challenges with future adaptive dialogues in combination with adaptive user interfaces. The approach to add adaptivity to an overall user-system dialogue with different services described with UIDLs implies different requirements. The derived requirements have to be an essential part of future user interface descriptions and can be fulfilled by different types of UIDLs. We highlight the need to describe the UI on different levels of abstraction, device- and modality-independent, and the addition of different goals a user interface, respectively the underlying application, can fulfill. The work on adaptive dialogue management for abstract user interfaces is currently ongoing. UIDLs have to meet different requirements whether they shall support an interaction based or an information based approach. The requirements shall be used as basis for further discussions and push future research to achieve adaptive, flexible and reliable systems – so called Companion Systems.

## ACKNOWLEDGMENTS

This work is originated in the Transregional Collaborative Research Centre SFB/TRR 62 “Companion-Technology for Cognitive Technical Systems” funded by the German Research Foundation (DFG).

## REFERENCES

1. G. Bertrand, F. Nothdurft, F. Honold, and F. Schüssel. CALIGRAPHI – Creation of Adaptive dialogues using a GRAPHical Interface. In *COMPSAC 2011: 35th IEEE Annual Computer Software and Applications Conference*, pages 393–400, July 2011.
2. S. Biundo, P. Bercher, T. Geier, F. Müller, and B. Schattenberg. Advanced user assistance based on AI planning. *Cognitive Systems Research*, 12(3-4):219–236, April 2011. Special Issue on Complex Cognition.
3. G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon, and J. Vanderdonckt. A unifying reference framework for multi-target user interfaces. *Interacting with Computers*, 15(3):289–308, 2003.
4. J. Coutaz. PAC, an object oriented model for dialog design. In *Interact’87*, 1987.
5. M. Dausend and M. Poguntke. Spezifikation multimodaler interaktiver Anwendungen mit UML. In *Mensch & Computer*, pages 215–224. Oldenbourg Verlag, 2010.
6. G. de Melo. Modellbasierte Entwicklung von Interaktionsanwendungen. PhD thesis, Universität Ulm, 2010.
7. G. de Melo, F. Honold, M. Weber, M. Poguntke, and A. Berton. Towards a flexible ui model for automotive human-machine interaction. In *AutomotiveUI ’09: Proceedings of the 1st International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, pages 47–50, New York, NY, USA, September 2009. ACM.
8. B. Dumas, D. Lalanne, and S. Oviatt. Multimodal interfaces: A survey of principles, models and frameworks. In D. Lalanne and J. Kohlas, editors, *Human Machine Interaction – Research Results of the MMI Program*, volume 5440/2009 of *Lecture Notes in Computer Science*, chapter 1, pages 3–26. Springer-Verlag, Berlin, Heidelberg, March 2009.
9. G. Ferguson, J. Allen, B. W. Miller, E. K. Ringger, and T. S. Zollo. *Dialogue Systems: From Theory to Practice in TRAINS-96*, pages 347–376. Handbook of Natural Language Processing. Marcel Dekker, New York, 2000.
10. M. Horchani, L. Nigay, and F. Panaget. A platform for output dialogic strategies in natural multimodal dialogue systems. In *IUI ’07: Proceedings of the 12th international conference on Intelligent user interfaces*, pages 206–215, New York, NY, USA, 2007. ACM.
11. M. Johnston, S. Bangalore, G. Vasireddy, A. Stent, P. Ehlen, M. Walker, S. Whittaker, and P. Maloor. MATCH: An architecture for multimodal dialogue systems. In *ACL ’02: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 376–383, 2002.
12. V. Kluge, F. Honold, F. Schüssel, and M. Weber. Ein UML-basierter Ansatz für die modellgetriebene Generierung grafischer Benutzerschnittstellen (to appear). In *Informatik 2011: Informatik schafft Communities, Beiträge der 41. Jahrestagung der Gesellschaft für Informatik e.V. (GI)*, LNI. GI, 2011.
13. O. Lemon, A. Bracy, A. Gruenstein, and S. Peters. The witas multi-modal dialogue system i. In *EUROSPEECH*, pages 1559–1562, 2001.

14. G. Meixner. Entwicklung einer modellbasierten Architektur für multimodale Benutzungsschnittstellen. PhD thesis, TU Kaiserslautern, 2010.
15. G. Mori, F. Paterno, and C. Santoro. Design and development of multidevice user interfaces through multiple logical descriptions. *IEEE Trans. Softw. Eng.*, 30:507–520, August 2004.
16. S. Sire and S. Chatty. The markup way to multimodal toolkits. In *W3C Multimodal Interaction Workshop (2004)*. W3C, June 2004.
17. Helms, J., Schaefer, R., Luyten, K., Vermeulen, J., Abrams, M., Coyette, A., and Vanderdonckt, J. Human-Centered Engineering with the User Interface Markup Language. In *Human-Centered Software Engineering*. Chapter 7, Seffah, A., Vanderdonckt, J., Desmarais, M. (Eds.), HCI Series. Springer, London (2009), pp. 141-173..
18. M. Turk. Multimodal human-computer interaction. In *Real-Time Vision for Human-Computer Interaction*, number 3, chapter 16, pages 269–283. Springer US, 2005.
19. J. Vanderdonckt, Q. Limbourg, B. Michotte, L. Bouillon, D. Trevisan, and M. Florins. Usixml: a user interface description language for specifying multimodal user interfaces. In *Proceedings of W3C Workshop on Multimodal Interaction WMI'2004*, pages 1–7. W3C, 2004.
20. W. Wahlster. Smartkom: Symmetric multimodality in an adaptive and reusable dialogue shell. Technical report, DFKI, 2003.

# An Extension of UsiXML Enabling the Detailed Description of Users Including Elderly and Disabled

Nikolaos Kaklanis<sup>1,2</sup>, Konstantinos Moustakas<sup>1</sup>, Dimitrios Tzovaras<sup>1</sup>

<sup>1</sup>Informatics and Telematics Institute  
Centre for Research and Technology Hellas  
Thessaloniki, Greece  
nkak@iti.gr, moustak@iti.gr

<sup>2</sup>Department of Computing  
University of Surrey  
Guildford, United Kingdom

## ABSTRACT

The present paper proposes an extension of UsiXML that enables the detailed description of users, including elderly and people with disabilities. Furthermore, an application that enables the automatic extraction and editing of User Models, according to the proposed extension, is presented. Finally, a test case is presented, which shows how the proposed user models can be put into practice.

## Author Keywords

Disabled, elderly people, UsiXML, user interface description language, user model editor, virtual user.

## General Terms

Design, Human Factors, Theory

## Categories and Subject Descriptors

D.2.2 [Software Engineering]: Design Tools and Techniques – *User interfaces*. H.5.2 [Information Interfaces and Presentation]: User Interfaces – *User-centered design*.

## INTRODUCTION

The research community has shown interest in user modelling and user profiling over the last years. Many different user models have been introduced, including abstract user descriptions, like Personas [9], ontology-based user models or XML-based models. However, there is a lack of a unified user modelling technique able to describe user characteristics in detail with focus on the elderly and disabled. The present paper introduces an extension of UsiXML language that enables the detailed description of the user, including possible disabilities, the affected by the disabilities tasks as well as physical, cognitive and behavioral/psychological user characteristics. A user model editor that enables the extraction of user models, according to the proposed extension, as well as a test case showing how the proposed user modeling technique can be put into practice are also presented.

## RELATED WORK

User models are represented in the literature using different syntaxes and implementations, varying from flat file structures and relational databases to full-fledged RDF with bindings in XML. The notion of ontology-based user models was first developed by Razmerita *et al.* in 2003 [10] that presented the OntobUM, a generic ontology-based user modelling architecture. OntobUM integrated three ontolo-

gies: a user ontology characterizing the users, a domain ontology defining the relationships between the personalization applications, and a log ontology defining the semantics of user-application interaction. A similar, but way more extensive approach for ontology-based representation of the user models was presented by [4]. In [2] GUMO is proposed, which seems to be the most comprehensive publicly available user modelling ontology to date. The need for a commonly accepted ontology for user models is also justified. These works are natural extensions of earlier works on general user modelling systems [5,7,7]. Such a general user model ontology may be represented in a modern semantic web language like OWL, and thus be available for all user-adaptive systems.

XML-based languages for user modelling have also been proposed [11]. UserML [1,3] has been introduced as a user model exchange language. A central conceptual idea in UserML's approach is the division of user model dimensions into the three parts *auxiliary*, *predicate* and *range*. For example, if one wants to say *something about the user's interest in football*, one could divide this so-called *user model dimension* into the auxiliary part *has interest*, the predicate part *football* and the range part *low-medium-high*. Additionally, further important meta attributes have been identified for the user modeling domain, like the *situation* (like start, end, durability, location and position), *privacy* (like key, owner, access, purpose, retention) and *explanation* (like creator, method, evidence, confidence).

There are also many existing standards related to user modelling. The ETSI ES 202 746 standard specifies user preferences, including needs of people with disabilities; device related preferences and provides UML class diagrams describing the structure of the user profile. ETSI ES 202 746 builds on the user profile concept described in EG 202 325. ISO/IEC 24751-1:2008 is another relevant to user modelling standard, as it provides a common framework to describe and specify learner needs and preferences on the one hand and the corresponding description of the digital learning resources on the other hand so that individual learner's preferences and needs can be matched with the appropriate user interface tools and digital learning resources. ETSI EG 202 116 contains definitions of user characteristics, including sensory, physical and cognitive abilities and also describes how user abilities are changing over years.

## PROPOSED USIXML EXTENSION

In order to create user models that could be automatically used by software tools/modules/frameworks, the use of a machine-readable format is essential. The goal of the current research is to define a formal way to describe users, including elderly and people with disabilities. Thus, the detailed description of user's disabilities as well as the affected/problematic (due to the disabilities) tasks has to be supported.

UsiXML [12] has been chosen to be the basis of the proposed user modelling technique, as it can sufficiently describe user tasks, has some primal support for user description and it is easily extensible, due to its XML nature. Two new models are introduced and added to UsiXML's *uiModel* (Figure 1):

- the *disabilityModel* (Figure 2) and
- the *capabilityModel* (Figure 3).

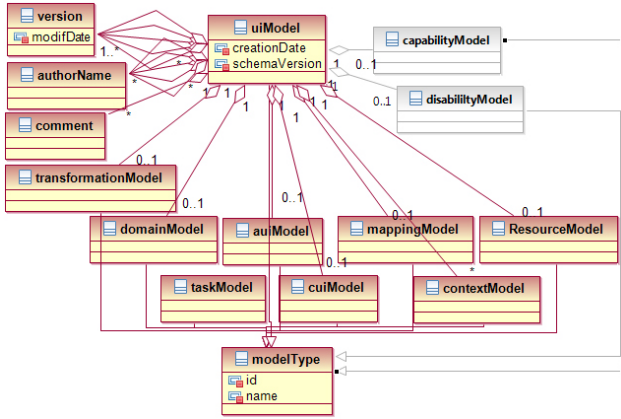


Figure 1. *uiModel* - UML class diagram.

The *disabilityModel* describes all the possible disabilities of the user as well as the affected by the disabilities tasks. Each *disability* element has a name and a type (e.g. motor, visual, etc.). Each *affectedTask* element has the following attributes:

- *id*: task's unique identity.
- *type*: the type of the task (e.g., motor, visual, etc.).
- *name*: task's name.
- *taskObject* (optional): the name of the task object (e.g. "door handle" may be the task object for task "open door").
- *details* (optional): some details/comments concerning the execution of the task.
- *failureLevel*: an indicator showing the failure level of the task due to the disabilities [accepted values: 1 to 5] – failureLevel=5 means that the user is unable to perform the specific task.

On the other hand, the *capabilityModel* describes in detail the physical, cognitive and the behavioral/psychological

user characteristics. The majority of the parameters of the proposed user model concerns the physical characteristics, as most of them are measurable and independent from the environment, in contrast with the cognitive and behavioral/psychological ones.

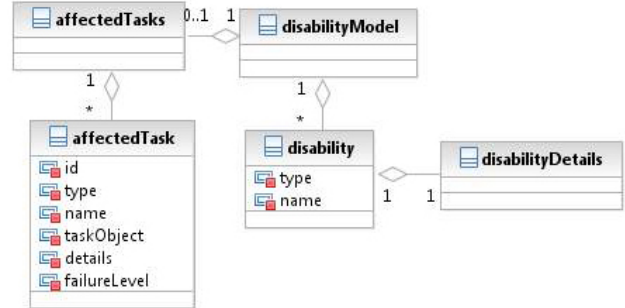


Figure 2. *disabilityModel* – UML class diagram.

More specifically, the *capabilityModel* contains the following basic elements:

- *general*: container for some general characteristics (e.g. gender, ageGroup).
- *generalPreferences*: container for user's needs/preferences (e.g., preferred input/output modality, preferred sound volume).
- *anthropometric*: container for the anthropometric data (e.g., weight, stature, head length, sitting height, bicep breadth).
- *motor*: container for the motor parameters (e.g., wrist/elbow/shoulder flexion, hip abduction).
- *vision*: container for the visual parameters (e.g., visual acuity, glare sensitivity, spectral sensitivity).
- *hearing*: container for the hearing parameters (e.g., resonance frequency, hearing thresholds).
- *speech*: container for the speech parameters (e.g., voice pitch, fundamental frequency, syllable duration).
- *Cognition*: container for the cognitive parameters (e.g., memory).
- *Behaviour*: container for the behavioral parameters (e.g., valence, emotional intelligence).

As an example, if we suppose a user with arthritis that affects hand fingers' flexion, causing reduction of hand's movement and, thus, making grasping a problematic task, the corresponding user model would include:

- An abstract description of arthritis and the definition of the affected task "grasping", in the *disabilityModel*.
- The angles of flexion of each finger for both hands, which would be reduced compared with the corresponding angles of a person having no disabilities, in the *capabilityModel*.



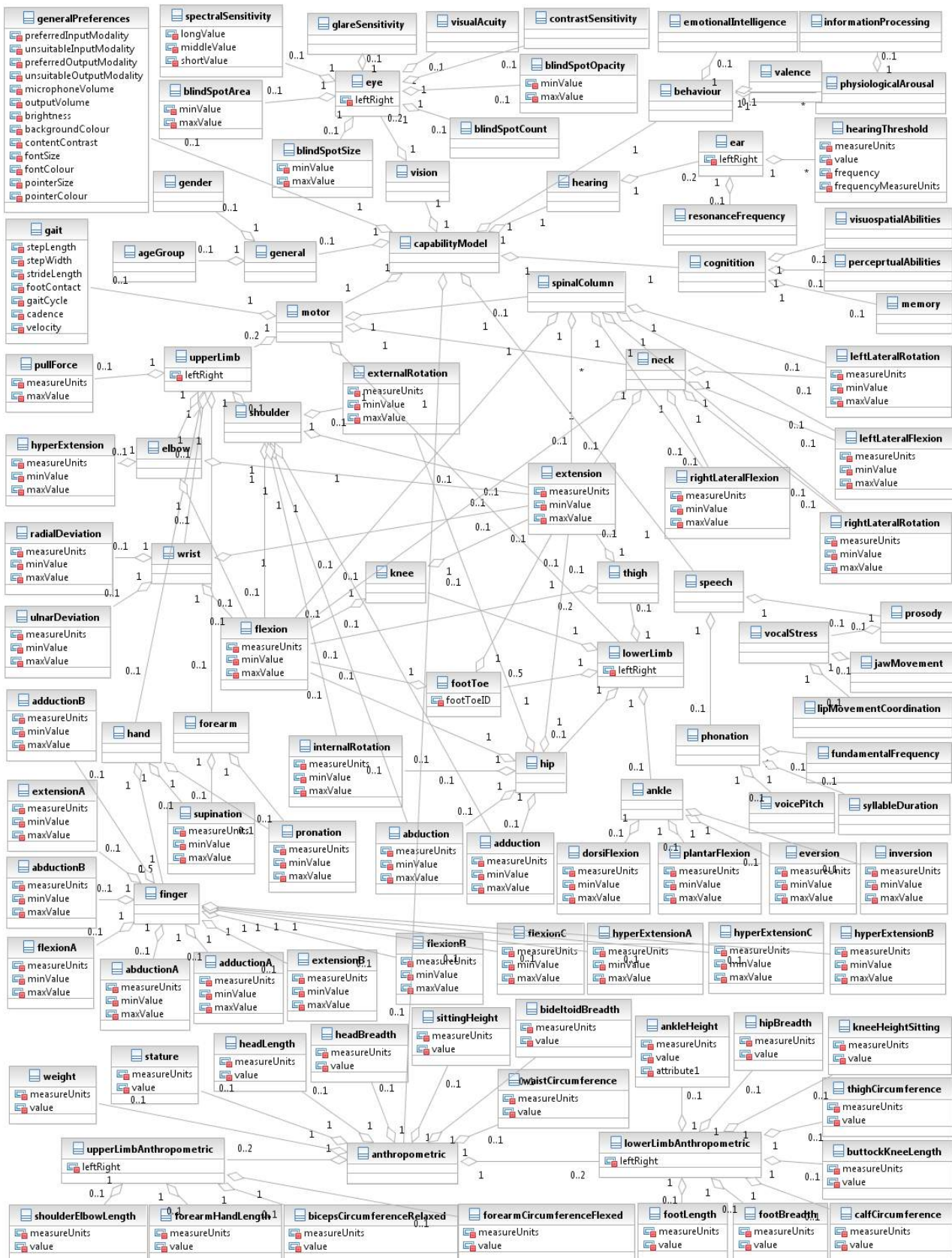


Figure 3. capabilityModel – UML class diagram.

## USER MODEL EDITOR

In this section, an editor is presented that has been developed to enable the easy creation of user models in UsiXML format, according to the proposed extension, as described in the previous section. The user is able to create a new user model or load a previously saved user model and edit it, as presented in Figure 4. Through the graphical user interface (GUI) of the application, the value of each supported parameter can be set (e.g. Figure 6 depicts a form where hip parameters can be set). Moreover, all the possible disabilities can be defined through the GUI (Figure 7).

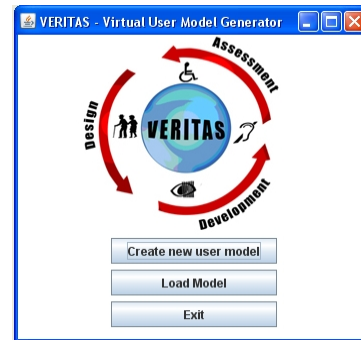


Figure 4. User Model editor – Main screen.

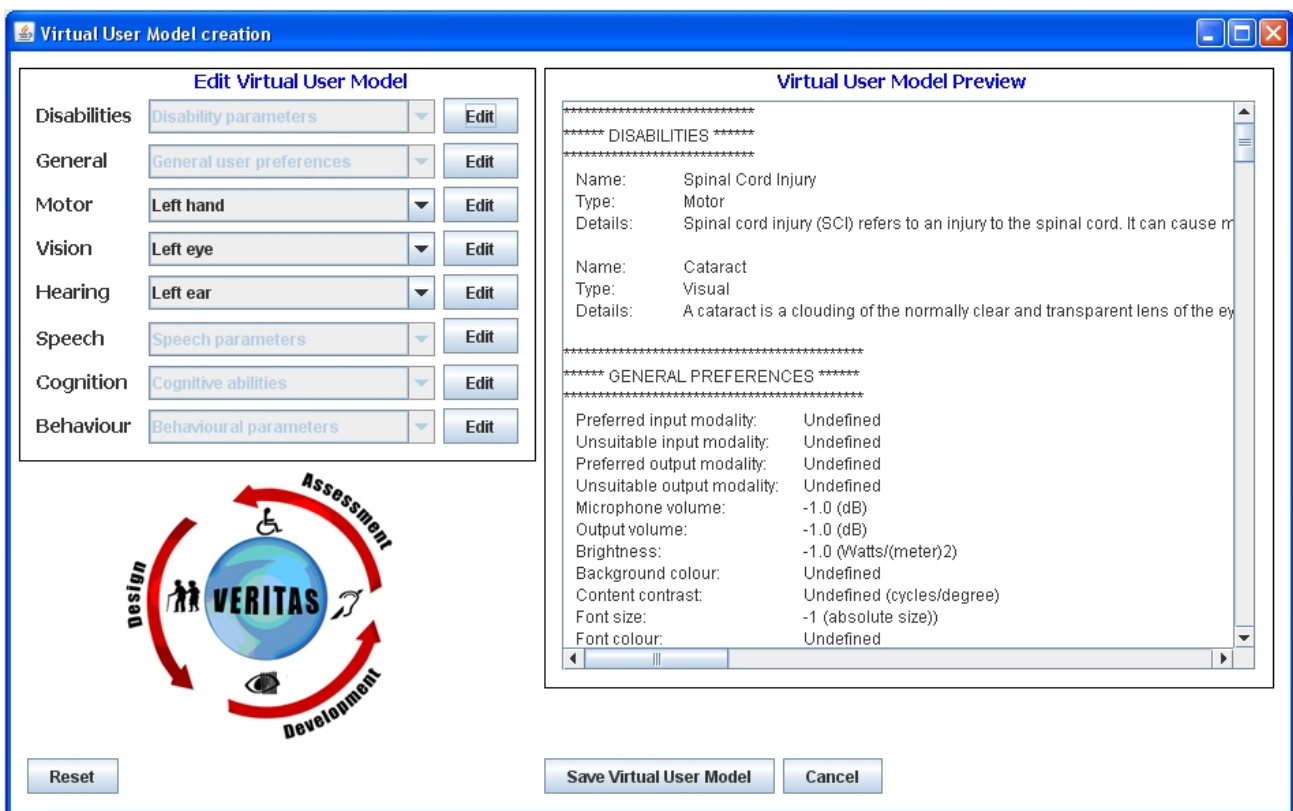


Figure 5. User Model preview.

At the beginning of the user model's creation process, the User Model Editor sets all the parameters, using a set of default values that have been found in the literature. These default values correspond to a typical user having no disabilities.

The User Model Editor provides also a preview of the User Model in a human readable format (Figure 5). The produced User Model can be automatically extracted in the proposed UsiXML format, as depicted in Table 1.

**Hip parameters**

Abduction :	min :	<input type="text" value="0.0"/>	degrees	max :	<input type="text" value="37.5"/>	degrees
Adduction :	min :	<input type="text" value="0.0"/>	degrees	max :	<input type="text" value="25.0"/>	degrees
Flexion :	min :	<input type="text" value="0.0"/>	degrees	max :	<input type="text" value="135.0"/>	degrees
Extension :	min :	<input type="text" value="0.0"/>	degrees	max :	<input type="text" value="10.0"/>	degrees
Internal rotation :	min :	<input type="text" value="0.0"/>	degrees	max :	<input type="text" value="40.0"/>	degrees
External rotation :	min :	<input type="text" value="0.0"/>	degrees	max :	<input type="text" value="40.0"/>	degrees

OK Cancel

**Figure 6. Edit hip parameters form. The user is able to define the range of hip's abduction, adduction, flexion, extension, internal and external rotation for each hip.**

**Disabilities**

	Type	Name	Details
Disability No.1 :	Motor	Spinal Cord Injury	Spinal cord injury (SCI) refers to an injury to the spinal cord. It can cause myelopathy or damage to nerve roots or myelinated fiber tracts that carry signals to and from the brain. Depending on its classification and severity, this type of traumatic injury could also damage the grey matter in the central part of the cord, causing
Disability No.2 :	Visual	Protanomaly (Moderate) - Both eyes	Having a mutated form of the long-wavelength (red) pigment, whose peak sensitivity is at a shorter wavelength than in the normal retina, protanomalous individuals are less sensitive to red light than normal. This means that they are less able to discriminate colors, and they do not see mixed lights as having the same col
Disability No.3 :	Visual	Deutanomaly (Mild) - Both eyes	Having a mutated form of the medium-wavelength (green) pigment. The medium-wavelength pigment is shifted towards the red end of the spectrum resulting in a reduction in sensitivity to the green area of the spectrum. Unlike protanomaly, the intensity of colors is unchanged. This is the most common form of color blind
Disability No.4 :	Visual	Glaucoma (Severe) - Both eyes	Glaucoma is an eye disorder in which the optic nerve suffers damage, permanently impacting vision in the affected eye(s) and progressing to complete blindness if untreated. It is often, but not always, associated with increased pressure of the fluid in the eye (aqueous humour).
Disability No.5 :	Motor	Rheumatoid arthritis	Rheumatoid arthritis (RA) is a chronic, systemic inflammatory disorder that may affect many tissues and organs, but principally attacks synovial joints. The process produced an inflammatory response of the synovium (synovitis) secondary to hyperplasia of synovial cells, excess synovial fluid, and the development of pann
Disability No.6 :	Cognitive	Dyslexia	Dyslexia is a broad term defining a learning disability that impairs a person's fluency or comprehension accuracy in being able to read, and spell, and which can manifest itself as a difficulty with phonological awareness, phonological decoding, orthographic coding, auditory short-term memory, and/or rapid naming.
Disability No.7 :	Undefined		

OK Cancel

**Figure 7. Add/edit disabilities form. The user can easily define the type, name and details of all the possible disabilities.**

```

<?xml version="1.0" encoding="UTF-8"?>
<uiModel xmlns="http://www.usixml.org"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:schemaLocation="http://www.usixml.org/spec/Usi
XML-ui_model.xsd" id="User_Model" name="Exported
Virtual User Model" creationDate="2011/07/04
13:39:51" schemaVersion="1.8.0">
  <head>
    <version modifDate="2011/07/04
13:39:51">1.0</version>
    <authorName>Automatically generated by the
VERITAS User Model Editor</authorName>
    <comment>This model has been generated using
the VERITAS User Model Editor</comment>
  </head>
  <disabilityModel>
    <disability type="Motor" name="Spinal Cord In-
jury">
      <disabilityDetails>Spinal cord injury (SCI)
refers to an injury to the spinal cord. It can
cause myelopathy or damage to nerve roots or mye-
linated fiber tracts that carry signals to and
from the brain. Depending on its classification
and severity, this type of traumatic injury could
also damage the grey matter in the central part of
the cord, causing segmental losses of interneurons
and motor neurons.</disabilityDetails>
    </disability>
    <affectedTasks>
      <affectedTask id="walking_ID" type="motor"
name="walking" taskObject="" details="inability to
effectively transfer weight between legs, abnormal
step rhythm, excessive plantar flexion during
swing phase, falling during activities" fail-
ureLevel="2" />
    </affectedTasks>
  </disabilityModel>
  <capabilityModel>
    <generalPreferences>
<unsuitableInputModali-
ty>Undefined</unsuitableInputModality>
    ...
  </generalPreferences>
  <motor>
    <upperLimb leftRight="left">
      <pullForce measureUnits="N"
maxValue="335.0"/>
    <hand>
      <finger fingerID="thumb">
        <flexionA measureUnits="degrees" min-
Value="0.0" maxValue="35.0"/>
      </finger>
      ...
    </hand>
    <wrist>
      <radialDeviation measureUnits="degrees"
minValue="0.0" maxValue="27.5"/>
      <ulnarDeviation measureUnits="degrees"
minValue="0.0" maxValue="35.0"/>
    </wrist>
    <forearm>
      <pronation measureUnits="degrees" min-
Value="0.0" maxValue="85.0"/>
      <supination measureUnits="degrees" min-
Value="0.0" maxValue="85.0"/>
    </forearm>
    <elbow>
      <flexion measureUnits="degrees" minVal-
ue="0.0" maxValue="142.5"/>
      <hyperExtension measureUnits="degrees"
minValue="0.0" maxValue="10.0"/>
    </elbow>
    <shoulder>
      <flexion measureUnits="degrees" minVal-

```

```

ue="0.0" maxValue="86.0"/>
      <extension measureUnits="degrees" min-
Value="0.0" maxValue="40.0"/>
      <abduction measureUnits="degrees" min-
Value="0.0" maxValue="21.0"/>
      <adduction measureUnits="degrees" min-
Value="0.0" maxValue="30.0"/>
    ...
  </shoulder>
</upperLimb>
<upperLimb leftRight="right">...</upperLimb>
<lowerLimb leftRight="left">
  <hip>
    <abduction measureUnits="degrees" min-
Value="0.0" maxValue="37.5"/>
    ...
  </hip>
  <thigh>...</thigh>
  <knee>...</knee>
  <ankle>...</ankle>
  <footToe footToeID="1">
    <flexion measureUnits="degrees" minVal-
ue="0.0" maxValue="35.0"/>
    <extension measureUnits="degrees" min-
Value="0.0" maxValue="35.0"/>
  </footToe>
  ...
</lowerLimb>
<lowerLimb leftRight="right">...</lowerLimb>
<neck>...</neck>
<spinalColumn>...</spinalColumn>
<gait>
  <stepLength>0.75</stepLength>
  ...
</gait>
</motor>
<vision>
  <eye leftRight="left">
    <glareSensitivity>0.845</glareSensitivity>
    ...
  </eye>
  <eye leftRight="right">...</eye>
</vision>
<hearing>...</hearing>
<speech>...</speech>
<cognition>...</cognition>
<behaviour>...</behaviour>
</capabilityModel>
</uiModel>

```

Table 1. User Model example – UsiXML source code.

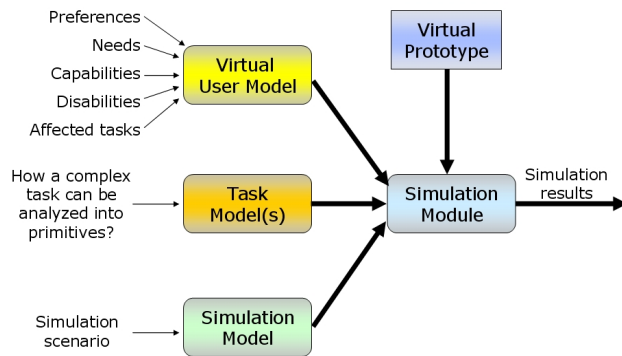
## THE PROPOSED USER MODELING TECHNIQUE IN PRACTICE

In this section, a use case is presented, where the proposed user modeling technique is used. In the context of the FP7 VERITAS EU funded project (FP7 – 247765), a framework that performs automatic simulated accessibility testing of designs in virtual environments has been developed. The Simulation Module is the core component of the VERITAS Simulation Framework. As depicted in Figure 8, the Simulation Module gets as input:

- A **Virtual User Model** expressed in UsiXML (according to the proposed extension) describing a virtual user with disabilities.
- A **Simulation Model** expressed in UsiXML (using the *taskmodel* of UsiXML) describing the functionality of the product/service to be tested.



- One or more **Task Models** expressed in UsiXML (using the *taskmodel* of UsiXML) describing in detail how the complex tasks (e.g. driving, computer use, etc.) are decomposed into primitive tasks (e.g. grasp, pull, etc.)
- A **3D Virtual Prototype** (like the one presented in Figure 9) representing the product/service to be tested.



**Figure 8. Veritas Simulation Framework architecture.**

The Simulation Module, then, simulates the interaction of the virtual user (as it is defined in the Simulation Model) within the virtual environment. The disabled virtual user is the main “actor” of the physically-based simulation that aims to assess if the virtual user is able to accomplish all the necessary actions described in the Simulation Model, taking into account the constraints posed by the disabilities (as described in the Virtual User Model).



**Figure 9. Virtual environment example representing a common car interior [6].**

In [6] the VERITAS Simulation Framework is described in detail and two test cases are presented revealing some accessibility issues of a virtual car interior for an elderly virtual user and a virtual user with spinal cord injury.

## DISCUSSION

The proposed user modeling technique seems very promising as it enables the detailed users’ description, including elderly and people with disabilities. The basic advantage of the proposed user models against the personas,

which is probably the most popular existing technique of describing a user, is the machine-readable format. Additionally, the structure of the proposed user model is easily extensible by to its XML nature. The introduced user model is strictly correlated with user tasks. Thus, the fact that the new user modeling technique is based on UsiXML offers another advantage, as UsiXML can sufficiently describe user tasks. The proposed user models could be used in various simulation platforms, enabling the simulation process for virtual users with different characteristics or in adaptive user interfaces, where the user interface of an application could dynamically change in order to fulfill user’s needs/preferences. Possible future extensions will be considered, in order to enable the description of more human body parameters that could be affected by a disability.

## CONCLUSION

In the present paper, an extension of UsiXML that enables the detailed description of a user, including users’ possible disabilities, the affected by the disabilities tasks as well as physical, cognitive and behavioural/psychological characteristics was presented. A User Model Editor that has been developed, in order to support the proposed user modeling technique, was also presented. Finally, an indicative use case showed how the proposed technique can be put into practice. The great importance of the proposed user modelling technique lies to the fact that it enables for the first time the description of the elderly and disabled users in a formal way. Additionally, the machine-readable format allows the produced user models to be used by different systems/applications.

## ACKNOWLEDGMENTS

This work is supported by the EU funded project VERITAS (FP7 – 247765).

## REFERENCES

1. Heckmann, D. Introducing situational statements as an integrating data structure for user modeling, context-awareness and resource-adaptive computing. In *Proc. of ABIS’2003* (Karlsruhe, 2003), pp. 283–286.
2. Heckmann, D. *Ubiquitous User Modelling*. Akademische Verlagsgesellschaft Aka GmbH, Berlin (2006).
3. Heckmann, D. and Krüger, A. A user modeling markup language (UserML) for ubiquitous computing. *Lecture Notes in Artificial Intelligence*, vol. 2702. Springer, Berlin (2003), pp. 393–397.
4. Heckmann, D., Schwartz, T., Brandherm, B., Schmitz, M., and von Wilamowitz-Moellendorff, M. GUMO – The General User Model Ontology. In *Proceedings of the 10th International Conference on User Modelling UM’2005* (Edinburgh, 2005). *Lecture Notes in Artificial Intelligence*, vol. 3538. Springer, Berlin (2005), pp. 428–432.
5. Jameson, A. Modelling both the context and the user. *Personal Technologies* 5, 1 (2001), pp. 29–33.

6. Kaklanis, N., Moschonas, P., Moustakas, K., and Tzovaras, D. A Framework for Automatic Simulated Accessibility Assessment in Virtual Environments. In *Proc. of Int. Conf. on Human-Computer Interaction HCI International'2011* (July 2011).
7. Kay, J. The UM toolkit for reusable, long term user models. *User Modelling and User-Adapted Interaction* 4, 3 (1995), pp. 149-196.
8. Kobsa, A. Generic user modelling systems. *User Modelling and User-Adapted Interaction* 11, 1-2 (2001), pp. 49–63.
9. Pruitt J. and Grudin, J. Personas: Practice and Theory. In *Proc. of ACM Conf. on User Experience DUX'2003*. ACM Press, New York (2003). <http://research.microsoft.com/en-us/um/people/jgrudin/>
10. Razmerita, L., Angehrn, A., and Maedche, A. Ontology-based User Modeling for Knowledge Management Systems. In *Proc. of the Ninth International Conference of User Modeling UM'2003*. 2003.
11. Souchon, N. and Vanderdonckt, J. A Review of XML-Compliant User Interface Description Languages. In *Proc. of 10th Int. Conf. on Design, Specification, and Verification of Interactive Systems DSV-IS'2003* (Madeira, 4-6 June 2003). J. Jorge, N.J. Nunes, J. Cunha (Eds.). Lecture Notes in Computer Science, vol. 2844, Springer-Verlag, Berlin (2003), pp. 377–391.
12. Vanderdonckt, J. Model-Driven Engineering of User Interfaces: Promises, Successes, and Failures. In *Proc. of 5<sup>th</sup> Annual Romanian Conf. on Human-Computer Interaction ROCHI'2008* (Iasi, September 18-19, 2008), S. Buraga, I. Juvina (Eds.). Matrix ROM, Bucarest (2008), pp. 1–10.



# Issues in Model-Driven Development of Interfaces for Deaf People

**Paolo Bottoni, Fabrizio Borgia, Daniel Buccarella, Daniele Capuano,  
Maria De Marsico, Anna Labella, Stefano Levialdi**

Computer Science Department, "Sapienza" University of Rome

Via Salaria 113, 00198, Rome (Italy)

(bottoni, capuano, demarsico, labella, levialdi)@di.uniroma1.it, danielbuccarella@yahoo.it

## ABSTRACT

Model-driven development of interfaces for deaf users encounters specific problems: interaction must be based on visual communication, either graphical or through gestures, but standard solutions for developing visual interfaces are not adequate and one must consider specific, possibly unusual, ways of structuring interaction. We discuss such issues in relation to the development of a Deaf-centered E-Learning Environment. Moreover, we consider problems related to the modeling of Sign Languages, both in written and gestural form.

## Author Keywords

Storytelling, Deaf-centered E-Learning Environment, MOF, UIDL, UsiXML, SignWriting.

## General Terms

Design, Human Factors, Theory

## Categories and Subject Descriptors

H.5 [Information Interfaces and Presentation]: User Interfaces

## INTRODUCTION

Web contents accessibility for deaf users is a hidden problem as one might think that the intact visual channel is enough for deaf people to grasp all visual information, texts included [6]. Instead, deafness as a sensorial deficit hinders deaf people's acquisition of both written and vocal language skills, which raises important issues with respect to interfaces for e-learning [1]. Moreover, including Sign Language (SL) videos in web pages as translation of textual contents is not an adequate solution, since significant portions of the deaf community do not learn their national SL and prefer to communicate through verbal language, due to social constraints and prejudice. On the other hand, both signing and non-signing deaf people show similar difficulties in verbal language comprehension [1]. Hence, the use of e-learning to cover this literacy gap must adopt creative ways of presenting and coordinating interactive visual materials, providing different ways of access to content comprehension.

The Deaf-centered E-Learning Environment (DELE) targets adult deaf people attending university. In DELE, all information is presented visually, without using text. DELE structure design is based on Conceptual Metaphors (based on the Embodied Cognition paradigm [11,12]) and

Storytelling [4,13] and explores how metaphors based on interaction between humans and their environment (e.g. container, path), can facilitate learning. Moreover, we "translate" typical concepts from Web sites and social networks (e.g. personal pages, forums) into domain-specific entities, generating familiar environments.

The central metaphor in DELE is that of the university campus. Users can browse the campus environment via personal avatars, exploiting intuitive body-based actions. An exhaustive mapping links e-learning concepts to the environment (e.g. the campus main square represents the forum, personal houses stand for users' personal pages.).

In DELE, the whole learning process is seen as a story, following a path from a starting place to a conclusion, through several steps and detours. A visual representation is given by moving an avatar along the path, and the process is told as an entirely visual story, omitting textual information unless it is part of the didactic contents.

A StoryEditor integrated into DELE allows a tutor to design arbitrary learning paths as stories. A story-path is specified by composing several types of nodes, generating object code for web pages. Process execution relies on the mapping of stories onto workflows, enacted by a suitable engine. Navigational, graphical, and behavioral structures of the story pages are specified with the UsiXML ([www.usixml.org](http://www.usixml.org)) User Interface Description Language (UIDL). UsiXML was chosen due to its several models encompassing the different aspects of DELE design: in fact, since StoryEditor allows the generation of final web pages starting from an abstract formal description of story-paths, the different levels of abstraction available with UsiXML descriptions fit the need to deal with task coordination, flexible style of presentation, and domain concepts (in the e-learning domain) found in the different steps of code generation. Model-to-model transformations have been defined to move from the definition of stories as paths to the concrete organization of the environment. Moreover, we have used intra-model transformations to specify important behaviors. We illustrate the model-driven design of stories in DELE via the StoryEditor and the adopted transformation patterns.

One missing aspect of UsiXML is the possibility of specifying gesture-based interaction. As we are concerned with Sign Languages (SL), and we are introducing writing fa-

cilities based on Sign Writing (SW – [www.signwriting.org](http://www.signwriting.org)), we discuss a meta-model of SW abstract syntax, currently at the basis of the SWift graphical editor- With SWift, a user composes SW sentences using visual symbols (glyphs) to represent SL configurations, movements and facial expressions.

**Paper organization.** After discussing related work, we present the use of UsiXML for the formal description of fully iconic interactive contents. Then, we illustrate the use of StoryEditor to produce web pages from abstract models, and the different UsiXML models involved. Finally, we discuss representation of signs and gestures and propose a meta-model for SW.

## RELATED WORK

Storytelling is the basic abstraction behind the definition of storyboards in user interface development, and its integration with model-driven approaches is discussed in [10]. However, projects related to storytelling and e-learning environments for deaf people have generally not been mainly preoccupied with interface development.

The Signed Stories project (<http://www.signedstories.com>) makes children’s stories accessible in British Sign Language (BSL), using animation, pictures, text and sound to improve the literacy of deaf children. The MOODLE Course Management System (<http://moodle.org>) has been adapted at the University of Bristol Centre for Deaf Studies (CDS - <http://www.bris.ac.uk/deaf>) to deliver in BSL e-learning contents available in written English. The Digital Storytelling Program at Ohio State University (<http://digitalstory.osu.edu>) proposes showcases, presentations, publications, and workshops where deaf and hearing participants learn to use digital tools and interactive story circles to craft narratives.

Little has been done as to Model-Driven Development of interfaces for Deaf People. Traditional approaches to model-driven assistive interfaces have focused more on support to blind people, for which the alternative audio channel can be exploited [9,20]. On the other hand, gesture-based interaction has been prominently seen as an integration of speech (see e.g., [16]). Work on the MARIA framework is currently trying to integrate support for modeling of gesture-based interaction, but its model of gestures is typically related to movement sensors as available in games, or to multi-touch interfaces [15,17].

Concerning efforts to provide specifications of SLs, for example for generation of movements in avatars [7], the Hamburg Notation System is the basis for SiGML (Signing Gesture Mark-up Language) [8], which is concentrated on movements of hands and arms and not on the whole set of features in SLs, as is for SW.

## TELLING ICONIC STORIES

Our approach tries to exploit storytelling in order to develop an e-learning environment designed in a fully-visual fashion. The visual modality is often a central part of storytelling, especially when children are involved [19].

It has been observed that stories activate visual thinking ([www.learningandteaching.info/learning/dale\\_bruner](http://www.learningandteaching.info/learning/dale_bruner)) and that they represent one of the most important tools by which human beings represent experienced life, before a symbolic level of understanding is fully acquired [4]. Although this iconic modality is partially overcome by symbolization in adults, deaf people maintain a deep connection between symbolic and iconic levels of meaning due to their visual approach to knowledge. In other words, deaf people are essentially visual [1]. In the development of storytelling environments, two parallel levels of description must be considered:

1. A diachronic level representing the action flow along a story-path from the starting to the end place.
2. An iconic level for appearance and “felt quality” [10].

Computational modeling tools able to describe both levels are needed to allow DELE tutors to design and run iconic stories from scratch. While workflows provide a computational model for the diachronic level, a formal tool able to adequately describe the iconic level is needed.

## UIDLS AND USIXML

UIDLS represent an important resource in this direction. They allow the description of UIs in an implementation-independent way, specifying the UI features at several levels of abstraction. These levels, as described in the Cameleon Reference Framework (CRF) [5], are:

- *Tasks & Concepts*, describing user tasks which have to be realized independently of interaction modalities and concrete UI elements.
- *Abstract UI* (AUI), providing a modality-independent description, showing a UI as a collection of abstract containers and components.
- *Concrete UI* (CUI), including modality-specific concerns, but without reference to specific platforms or implementations.
- *Final UI* (FUI), describing the UI as users see it in specific hardware/software platforms.

UsiXML implements the Cameleon Reference Framework with different models for different levels of abstraction (with the exception of the Final UI which needs to be described directly in the target platform and is not specified in UsiXML). Model-to-model transformations are defined in a homogeneous formalism based on transformation rules, typically in the form of graph transformations. A rule is a triple (NAC, LHS, RHS) (see Figure 1), where:

- LHS (*Left Hand Side*) specifies a graph pattern, an occurrence of which has to be found in the original graph  $G$ , for the rule to be applied.
- RHS (*Right Hand Side*) specifies a different graph pattern, which has to replace the occurrence of LHS in  $G$  to produce the *result graph*  $G'$ .
- NAC (*Negative Application Condition*) is an extension of LHS for which a match must not be found.

Transformations allow designers to move from UsiXML *Task Model*, with relationships between tasks specified via the ConcurTaskTree [14] and LOTOS [2] formalisms, to the *CUI Model*, describing graphical and vocal modalities of interaction with interface elements and the associated behaviours, through the *AUI Model*, where a UI is represented in terms of *AbstractIndividualComponents* (AICs) and *AbstractContainers* (ACs), thus specifying the UI abstract building blocks and their relationships.

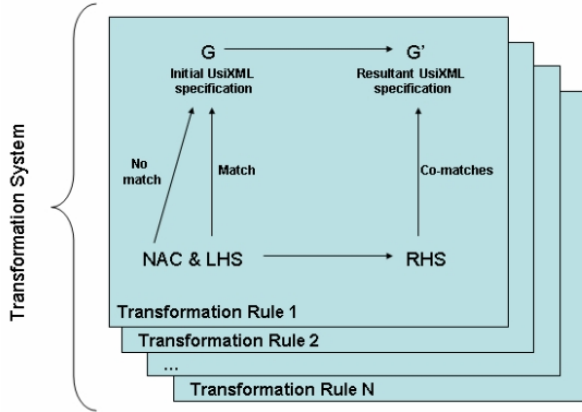


Figure 1. The UsiXML transformation rule system.

Using UsiXML-based descriptions within an editor for object code generation, the abstract models – together with abstract-to-concrete transformation rules – represent the information needed by the environment to determine the input parameters for the generation process. We now describe model-aided generation activities in StoryEditor.

### STORYEDITOR

The diachronic description level is the first perspective involved for iconic stories. From a computational point of view, we see stories as workflows: learning processes within an organization (e.g. university), in which tasks are assigned to actors (e.g. students and tutors). For DELE, we considered the YAWL (Yet Another Workflow Language - <http://www.yawlfoundation.org>) workflow management system providing a workflow engine for story-paths execution and an editor for their description (see Figure 2).

YAWL is an open-source workflow specification language extending Petri nets with dedicated constructs to capture Workflow Patterns (<http://www.workflowpatterns.com>).

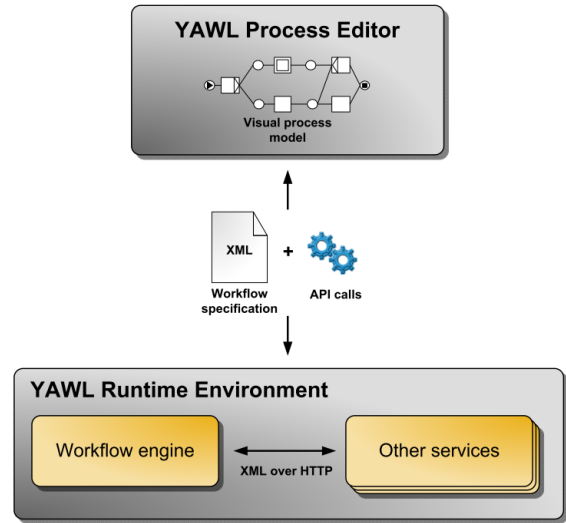


Figure 2. The YAWL Components.

DELE runtime environment is based on its engine, offering comprehensive support for the control-flow and resource patterns, but some problems arose with the YAWL Process Editor. This is a Java-based desktop application, while a major requirement for DELE was full and easy integration of the story editor as a browser-based application. This motivates the decision to develop an *ad-hoc* visual editor. *StoryEditor* (Figure 3) extends the WiringEditor component of WireIt (<http://neyric.github.com/wireit>), an open source JavaScript library to create wirable Web interfaces for dataflow applications, visual programming languages, graphical modeling or graph editors.

WireIt is based on the YUI (Yahoo! User Interface) Library (<http://developer.yahoo.com/yui/>), a set of utilities and controls, written in CSS and JavaScript, for building interactive web applications, and on inputEx, an open-source JavaScript framework to build fields and forms for web applications (<http://neyric.github.com/inputex>). The WiringEditor provided some of the common features of most visual editors for the story-paths visual language. The following subsections give a detailed explanation of the three steps in the definition of interfaces for stories.

### First Step: Visual Language Definition

By using StoryEditor, tutors are able to design both the structure and the visual, iconic content of a story.

#### Story Structure Definition

As arbitrary topological structures must be designable by didactic tutors, a formal meta-model has been defined to describe admissible learning paths, to eliminate ambiguity and to make automated syntax check possible.

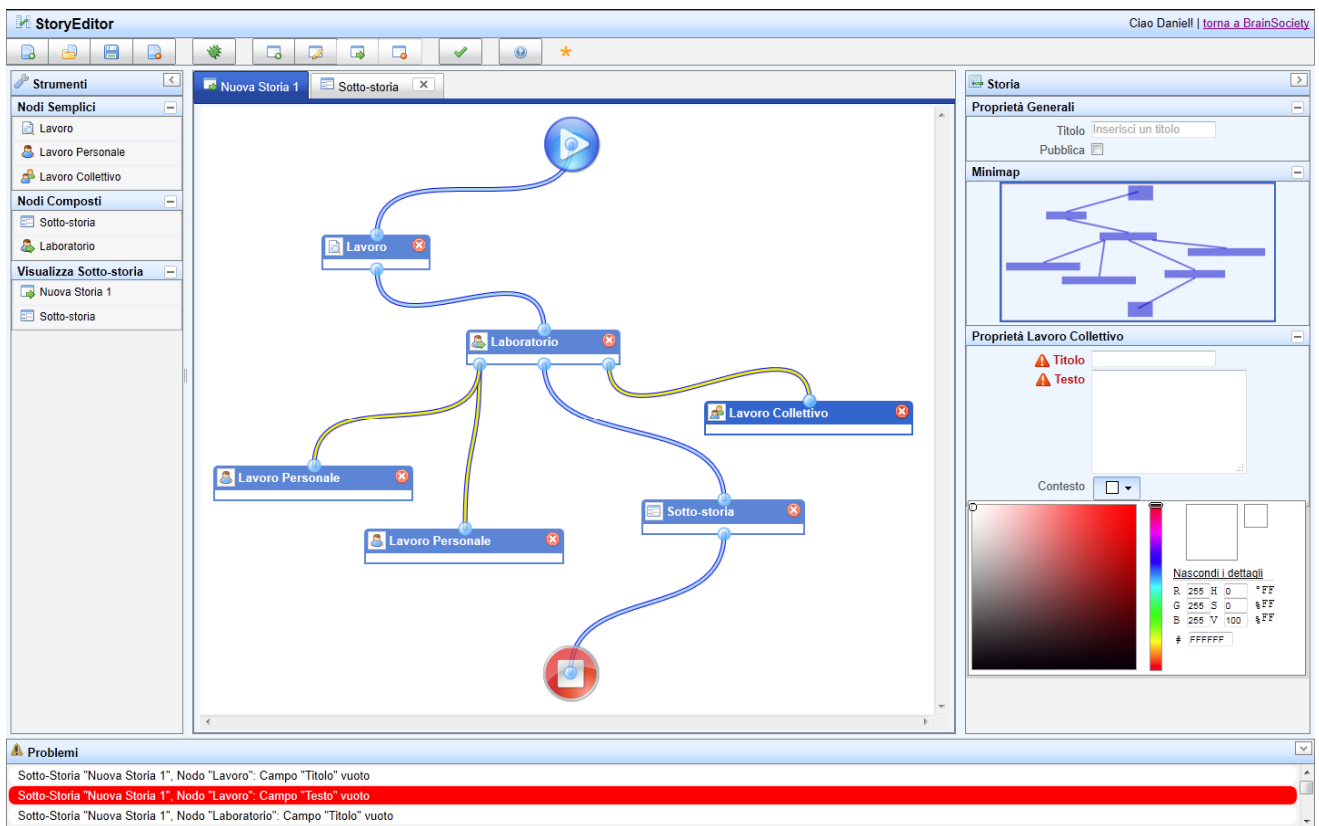


Figure 3. A screenshot of StoryEditor.

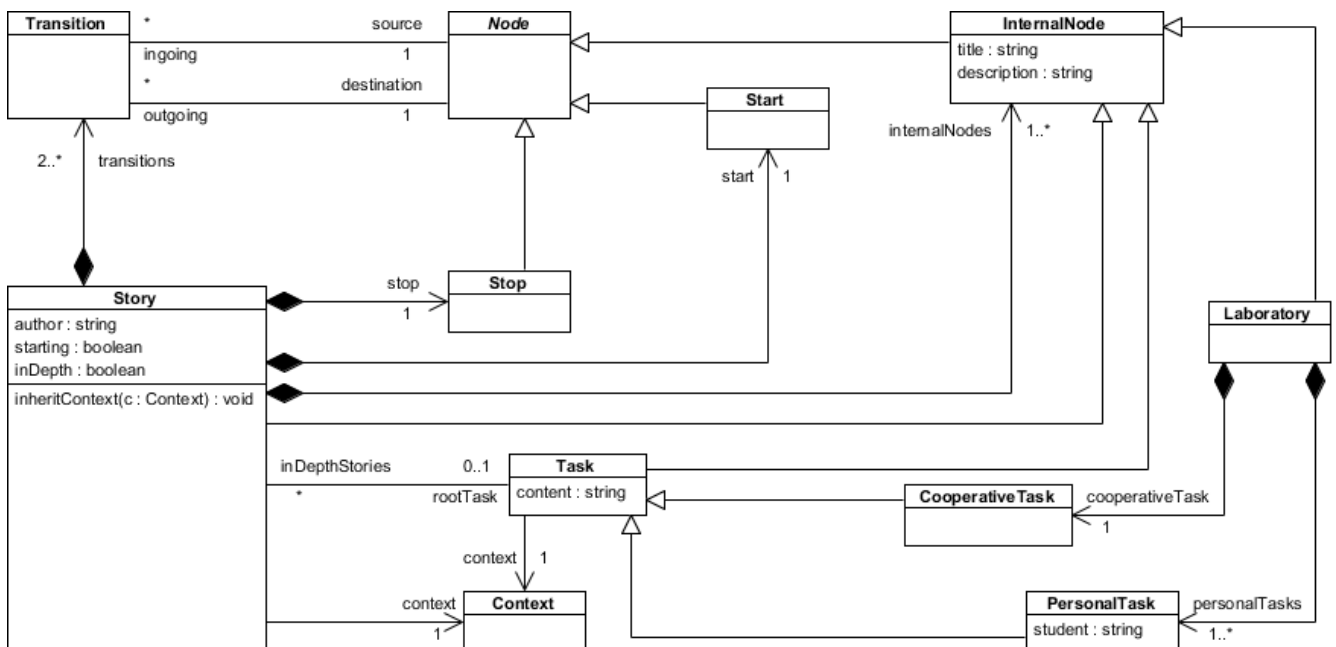


Figure 4. The formal meta-model for story-paths.

Figure 4 shows the DELE meta-model for story-paths expressed in MOF (OMG's Meta Object Facility - <http://www.omg.org/mof>) syntax, with each class an instance of the *Class* meta-class. The abstract class *Node* defines the basic elements that can be connected together to build a path, represented by the *Story* class. This contains a *Start* node, a *Stop* node and a set of *InternalNodes*, with arbitrary links among them. *Transition* represents connections between two nodes. Every node can have any number of ingoing and outgoing transitions, but these are only permitted among nodes within the same sub-story.

A node along a path can be either a *Task* (where a student has to perform a proper learning activity) or a story in turn (i.e., a sub-story). The structure described by the meta-model is high-level and there is no theoretical limit to the nesting of stories. As seen in Figure 4, from a task node a student can also access in-depth sub-stories. The latter do not belong to the normal path of the main story, but create entirely alternative paths which can be followed by the student within the navigation. The last extension of the internal nodes class is the *Laboratory* node.

The work students have to perform in a laboratory is divided into several *PersonalTasks* and a single *CooperativeTask*. An activity of the first type is assigned to each student, who has to perform it alone and asynchronously, while cooperative tasks can be performed in different ways (e.g. through shared documents).

#### Task Content Abstract Description

For the iconic level of story description, *Task* node internals are described by UsiXML models. A few main categories have been developed for DELE story pages (e.g. “story container” or “star story access”), providing a uniform structure for each of them. In particular, abstract descriptions have been developed in terms of *page patterns*, involving instances of the UsiXML's *Abstract Interaction Objects* (AIO) – either AIC or AC – and describing the structure of all pages in a category. To develop the models needed for a category, a Task Model is first given. Then the abstract page pattern is shown, and a first transformation is provided between tasks and AIOs the instances of which are defined in the pattern. Finally, graph transformation rules from AUI to CUI models are used.

These have been defined using the AGG tool (<http://user.cs.tuberlin.de/~gragra/agg/index.html>), while Task Models have been specified in the IdealXML (<http://www.usixml.org/index.php?mod=pages&id=15>) editing environment using LOTOS [2] syntax for temporal operators. The main structure for all pages within DELE is called *StoryContainer*. In a container, the main actions which can be performed by users within a story are defined, and adequate graphical structures for such actions are provided as the output of the reification process. The Task Model for *StoryContainer* is shown in Figure 5, where the two main task patterns are presented:

1. *InteractWithAnotherStory*: the user can enter a story for which the current story has to be left.
2. *InteractWithAStoryNode*: the user can enter a node from within the current story.

In Figure 5 both “abstract” (*StoryInteractionTasks*) and “interaction” tasks (e.g. *InteractWithAStoryNode*) are shown via IdealXML standard icons. The *alternate choice* operator “[ ]” is the relationship between interaction tasks.

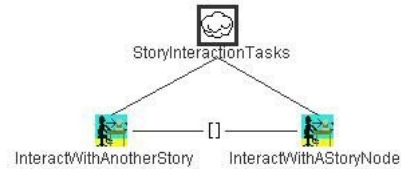


Figure 5. Task Model for *StoryContainer*.

The page pattern for *StoryContainer* is given in Figure 6 in terms of instances of AIC and AC classes from the UsiXML AUI model and of relationships between them.

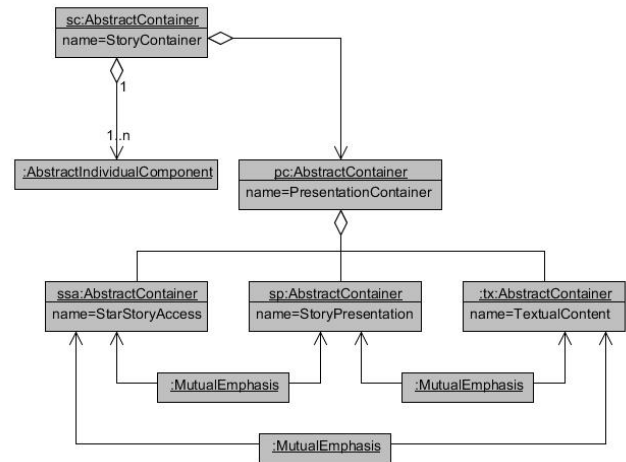


Figure 6. The page pattern for *StoryContainer*.

A main *PresentationContainer* (an AC) and several AIC instances form the pattern. *PresentationContainer* is composed of several different “page contents” related by *MutualEmphasis* (i.e. they cannot be shown together). The AUI Model derives from the Task Model, where either an AC or an AIC is associated with each task. Three transformations – one for each task pattern - generate the *StoryContainer* AUI Model. Figure 7 shows the rule for the *InteractWithAnotherStory* task. As in [18], we represent *InterModelRelationship* instances as associations between model components. A task is executed within an AIC, contained in an AC and composed of two facets: a *control* allows the AIC to activate a visualisation into another component in the concrete UI, and an *output* allows reification of the AIC as an iconic image component. In a similar way, the CUI Model is obtained from the AUI Model. Each AIC in *StoryContainer* is implemented as an *ImageComponent*. Page dynamics is drawn as shown in Figures 8 and 9.

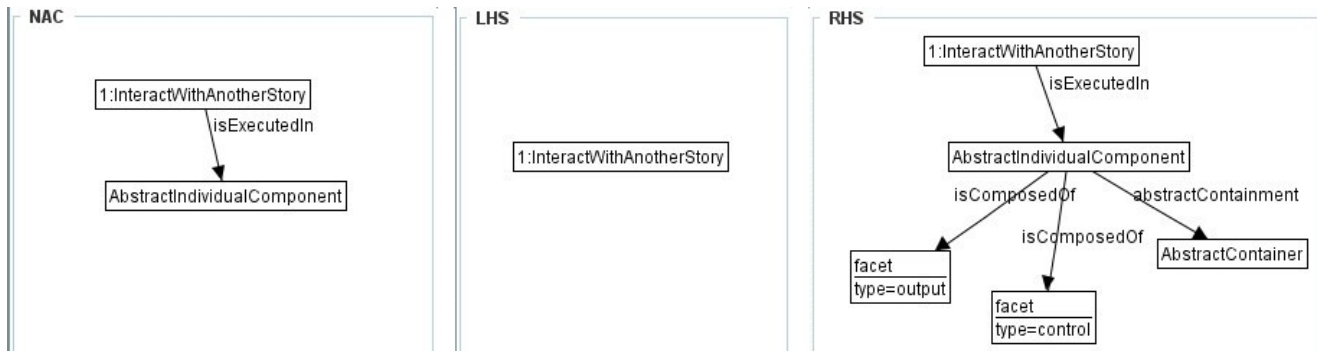


Figure 7. An example of transformation from the Task model to the AUI Model.

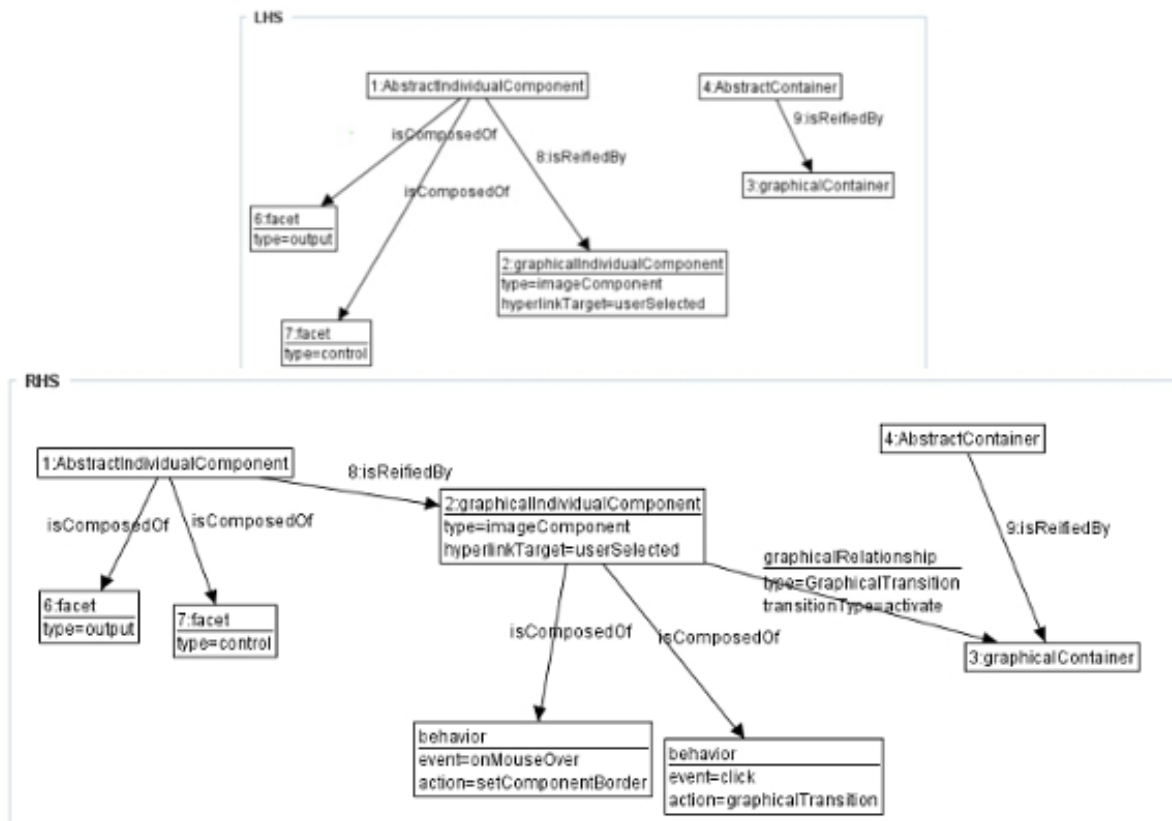


Figure 8. The main part of a transformation rule for the *StoryContainer*'s AUI-to-CUI reification.

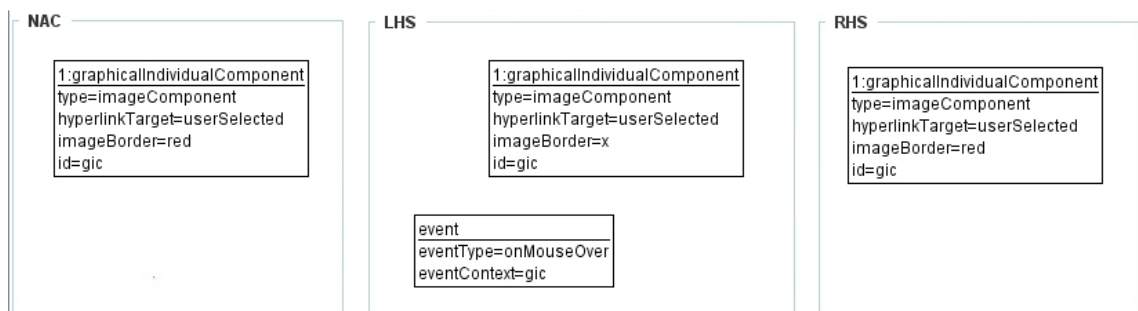


Figure 9. A transformation rule for the *onMouseOver* event.



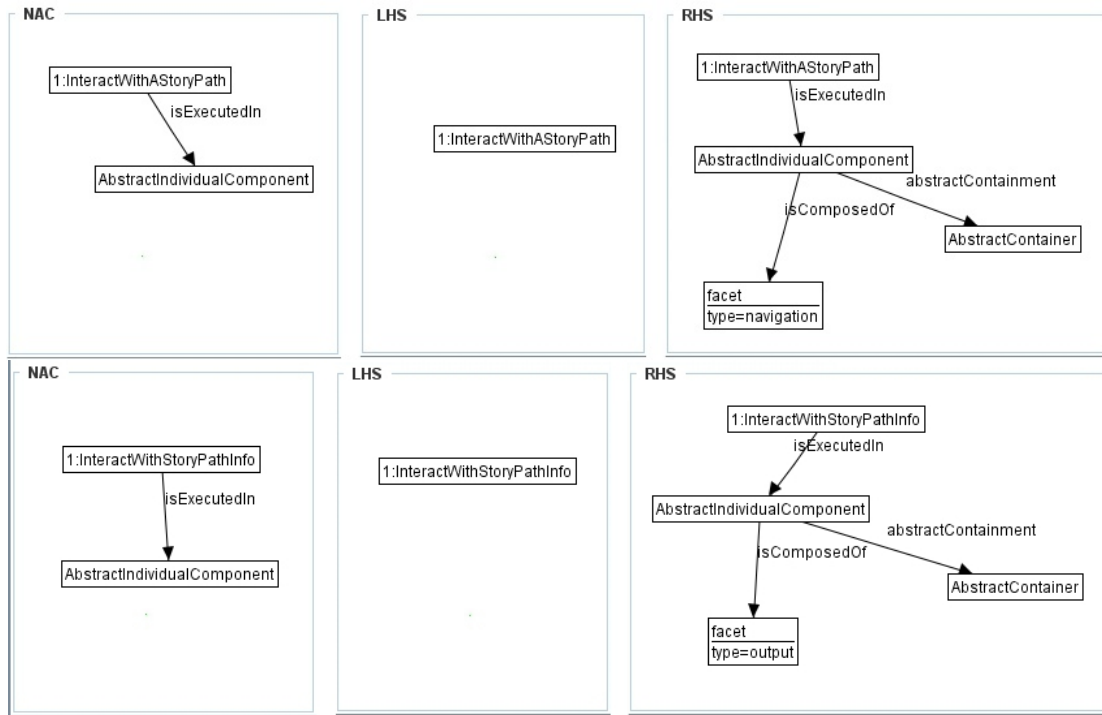


Figure 11. Rules for the *StarStoryAccess'* Task-to-AUI transformation.

To save space, the presentation of the rule for AUI-to-CUI reification in Figure 8 omits the NAC, which simply duplicates the rule RHS. Two graphical behaviors are associated with the AIC. In particular, a *click* event causes a *graphicalTransition* to start on the connected *graphicalContainer*. In the second rule (see Figure 9), the *mouseOver* event produces a graphical transformation in the AIC itself, i.e. it receives a colored border. Another example of UI description model is the *StarStoryAccess* structure. It represents the access door to “star” stories, i.e. stories where multiple paths can be entered by the user without an imposed order. The Task Model of such structures is quite simple and defines only two possible task patterns: users can either interact with a story-path or find information about a path (Figure 10).

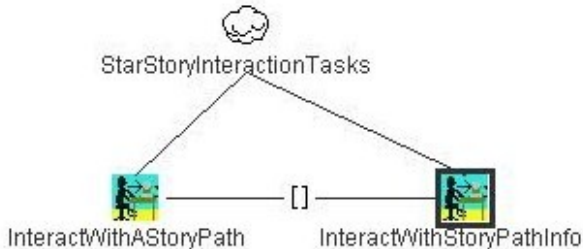


Figure 10. Task Model for *StarStoryAccess*.

The page pattern of *StarStoryAccess* extends the diagram shown in Figure 6, detailing the *StarStoryAccess* tree (Figure 12). The structure of *StarStoryAccess* simply states that a star story must have at least two paths which can be explored. In fact, if only one path can be followed,

a “linear” path would result. Both task patterns described for this UI have to be executed in a single AIC instance, thus the AUI model has to provide a multi-faceted GIC. In particular, as shown in Figure 11, each task is associated with one of the two facets of the GIC: the *InteractWithAStoryPath* task is responsible for the *navigation* facet, since the AIC itself will work as a navigation target when it is activated (as the CUI model below will show clearly). On the other hand, the *InteractWithStoryPathInfo* task requires an *output* facet for the AIC, since a textual label will be shown to the user as a *mouseOver* event is performed. The transformations needed to complete the CUI Model explain this. In particular, the two transformations for event management on the GICs are shown in Figure 13.

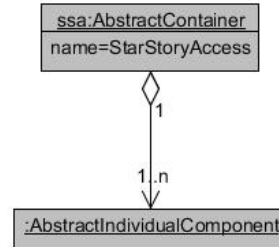
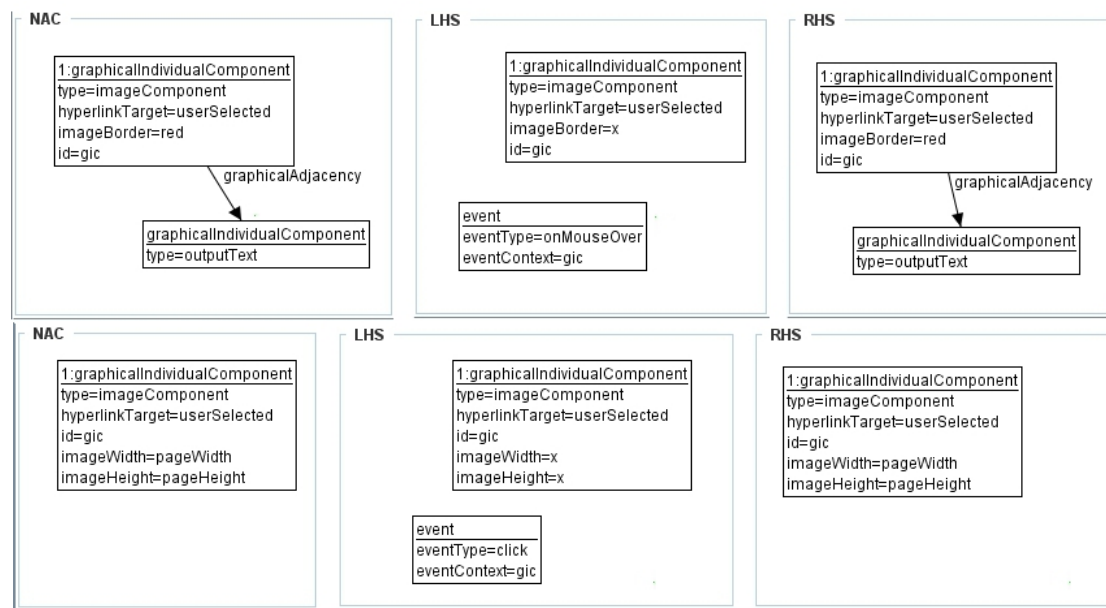


Figure 12. The page pattern for *StarStoryAccess*.

When a *click* event is triggered, the GIC has to display the page contents, expanding to full page size through an animation. In a similar fashion, the *mouseOver* event causes the GIC to react by visualizing information about the path it represents (i.e., a graphical output-text is put near the image component).



**Figure 13. Rules for the event management of the StarStoryAccess GICs.**

The set of models and transformation rules provide the needed input for *StoryEditor* to calculate the generation parameters: for example, as the page pattern shown in Figure 6 states that multiple AICs instances could be put in a *StoryContainer*, *StoryEditor* should require the exact number of these elements from the story designer; similarly, since AUI-to-CUI transformation rules specify that AICs are to be reified as *ImageComponent* elements, the designer will be asked for these images, etc. After all the needed models have been provided, *StoryEditor* is ready to run. On launching, *StoryEditor* reads the story content definition and structure from a JSON Language Definition module.

### Second Step: Story-Paths Editing

In fact, in order to achieve format homogeneity between software modules, the UsiXML descriptions are translated into JSON as well. This organization makes visual language editing simpler, because it only means editing this definition file. Hence, all the changes that can possibly be made to the structure or content meta-models are immediately reflected in the visual editor at launch. The resulting view is found in Figure 3. It gives a module list on the left side of its GUI layout. By selecting, dragging and dropping a module from the left tool panel to the main center drawing area, a node of that type is inserted into the diagram (i.e. created within the story). In order to facilitate the tutor in the process of designing a story-path, each time a new sub-story is created, the editor automatically inserts start and stop pseudo-nodes in the diagram. Visualization and navigation within sub-stories are made easier by creating a different window tab for each of them. In-depth sub-stories can be attached to a task node by linking it, for example, to some words of the node containing text. With respect to the needs they must

satisfy, in-depth sub-stories can be created by the tutor, deriving from the past navigation of the students or added by the users themselves based on their personal interests [3]. A syntax check verifies the story correctness and completeness based on the meta-model descriptions, which provide the set of constraints that a story must respect to be published. Finally, *StoryEditor* uses a JavaScript adapter to provide loading and saving features. It connects to a MySQL database through AJAX calls to a PHP backend to store the wirings and bring them back.

### Third Step: Story-Paths Translation

Once the editing phase is concluded and the syntax check has been passed, the new story can be published. The *StoryEditor* saves the story description in the form of an XML string, specifying a workflow to the interpreter of a runtime environment based on the YAWL engine. Moreover, the UsiXML CUI Model is generated by the editor using the transformation rules provided.

#### Story Structure Translation

In order to translate the structure of a story into a YAWL workflow specification, a mapping has been defined between the structure meta-model and the YAWL formal foundation. This gives the story designers the possibility to use YAWL expressiveness, having to learn only the few simple rules described by the meta-model. Many YAWL constructs, such as conditions, AND and XOR splits and joins, are in fact handled transparently to the users of the *StoryEditor*. Tasks from the story-path visual language are translated into YAWL *Atomic Tasks*, while sub-story nodes become YAWL *Composite Tasks*. A “Correction and support” atomic task is inserted after each activity. This task is assigned to tutors so they can

verify how learning activities have been performed. Using YAWL *Conditions*, a cycle is eventually generated, in which tutors can support students inserting links towards in-depth sub-stories close to their errors, when they are not sufficiently assimilated. A difference between laboratories and simple tasks is that for collaborative activities tutor verification starts only after all the personal and co-operative tasks have been completed by students.

## STORY ENGINE

As stated before, DELE runtime environment is based on the YAWL Engine, enabling students to go through paths composed of learning activities and stories within stories. When all nodes within a sub-story have been visited, the latter is marked as completed and the global context of the parent sub-story (if present) is recreated. When navigating along sub-stories, each time a sub-story is entered, and until there are nodes to be accessed, students can choose the next learning unit among “allowed” nodes. In a linear path, for example, the only allowed node is the first node at the beginning of the story. After choosing one node, a student can reach either a task node, another sub-story or a laboratory. Inside a laboratory, students can freely move between personal and cooperative task. The set of students attending the laboratory must always be known because a synchronization is required when one student tries to reach the cooperative task: in this case, in fact, a message is sent to all students which are not currently attending the cooperative task, and they are requested to enter it. If all students accept the request, the synchronization has been reached and the shared work can be done. Otherwise, students are redirected to their personal tasks. According to the order by which students visits nodes or to the alternative paths they must - or choose to - follow as in-depth sub-stories during execution of a story-path, each student “lives” a personal story. Hence, different sets of *Past Correlated Story-Paths* and *Personal In-Depth Story-Paths*, one for each student, are maintained and shown in a laboratory. The latter is considered concluded when all its tasks (both personal and cooperative) are done and the final tutor verification is passed.

*StoryEditor* generates XML strings conforming to YAWL specifications, describing each of the three perspectives of a process control flow (task sequences, splits, joins etc.), data (variables, parameters, predicates etc.) and resources (participants, roles, allocators, filters etc.). At runtime, the YAWL Engine is only responsible for the correct scheduling of tasks and the management of input/output data to/from tasks. Actual task execution is delegated by the engine to so-called YAWL Custom Services, which are able to communicate with the YAWL Engine and to perform activities in a task. Communication between DELE and the YAWL Engine is realized by implementing an appropriate custom service: the DELE *StoryEngine*. It provides the actual execution environment

for a story, taking its description as input and generating all the web pages needed for the story execution.

Other, specific custom services are provided in order to execute each DELE node. At runtime, the data within each node instance will be selected for viewing and/or updating. When a task is scheduled, the Engine will notify whichever custom service has been associated with the task that there is an activity ready to be delegated to it. Hence, the custom service performs a checkout of the task data and generates an appropriate editing form based on the CUI description provided by the *StoryEditor* for that node. Two examples of generated Final User Interfaces are shown in Figure 14 and 15. They show two initial pages of “star” story-paths. In particular the screenshot in Figure 15 shows the presentation page of a star story-path encountered as a sub-story of a general linear path.

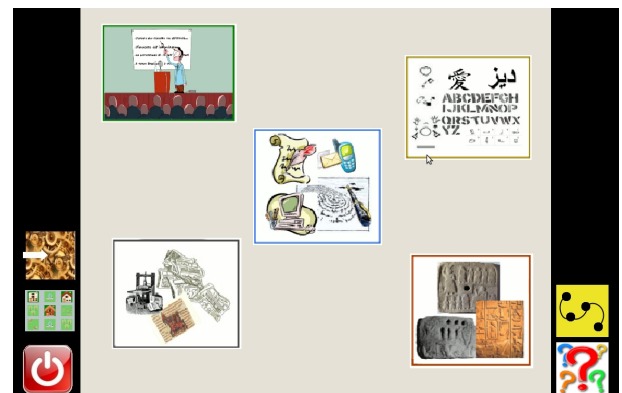


Figure 14. Initial page of a "star" story-path.

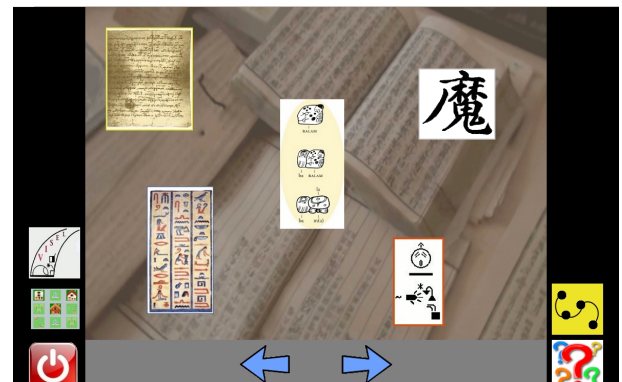


Figure 15. Initial page of a "star" sub-story.

## SIGN WRITING REPRESENTATION

SignWriting (SW - <http://www.movementwriting.org/symbolbank/>) is an alphabet, i.e. a list of symbols, used to produce a transcription of any Sign Language in the world. Compared with other notations, SW can express a signed sequence by itself, without the need for accessory descriptions, often written in a different language (typically the written form of a spoken language) as in the case of “glosses”. SW glyphs are all gathered in the International Sign-Writing Alphabet (ISWA), which therefore includes everything is needed to express any SL. ISWA is

available as an archive with tens of thousands of images (.png), each representing a SW glyph. Figure 16 shows examples of families.

In the metamodel of Figure 17, we define an *Utterance* as an instance of a *Concept*, where a same concept can be expressed using many different expressions. Utterances are formed by smaller entities; if in a spoken language we have words, in SL (and SW) there are *Sign* elements, as represented by the association between these two classes. A sign is produced by one or more *Occurrence* of some *Glyph* representing the specification of a physical manifestation of an individual traceable element. While the term *glyph* is more typical of the definition of SW, we use it also to describe atomic elements common to both SW and SL specifications.

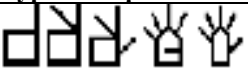
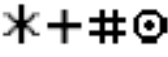


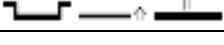
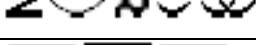

Glyph examples	Family description
	Hand configurations.
	Contacts: where, how and how many times a hand comes into contact with a body part.
	Movements (of hands, wrists, forearms, etc.).
	Head (expressions, movements, etc.).
	Shoulders, arms, bust.
	Dynamics and movement coordination.
	Punctuation.

Figure 16. Some examples of families of glyphs.

In this meta-model we also set an important difference between the specification of a glyph and its occurrences. This is motivated by two facts: first, the ISWA is composed by tens of thousands of glyphs, so there might be glyphs not occurring in any sign, but which are worth being stored and coded, because they might be useful in the future; second, a specification also encompasses the admissible variations which can be applied to each individual occurrence. In particular, a *Category* is an aggregate of glyphs subject to some constraint on each possible occurrence and the body parts and movements which can be used to generate their occurrences. Depending on the adopted concrete specification of the lexicon of signs, a category can be organized into sub-categories. Figure 17 presents the collection of categories associated with the ISWA definition e.g. *Configuration*, *ForearmMovement*, *HandMovements*.

An example of sub-categorisation in the ISWA definition (not shown in the metamodel, but which has been used in Swift, is the specialization of the *HandMovement* category as *StraightHandMovement* and *CircularHandMovement*.

In Sutton's original proposal for a concrete presentation of SW, of which our meta-model is a refinement, glyphs were organized according to the following hierarchy. We present it to give an idea of the complexity of the dimensions involved.

- *Category*: it distinguishes anatomical areas and other elements such as punctuation and contacts: configurations, movements, head and face, body, dynamics and rhythm, punctuation, advanced annotation.
- *Group*: each category is divided into a maximum of 10 groups, distinguishing different areas within the category macro-area. Groups in a category can be heterogeneous, e.g. a single category gathers all movements (of hands, forearms, wrists, and fingers) but also contacts.
- *Base symbol*: identifies a specific glyph in a group.
- *Variation*: distinguishes different manifestations of some symbols; as an example, for the symbol representing the bended forefinger at knuckle, the two possible variations code the difference in angle of the knuckle.
- *Filling*: they identify modifications of the same base symbol; as an example, filling in configurations are used to distinguish the visible side of the hand and the plane where the sign is performed: depending on whether the palm, the edge or the back of the hand are seen, and whether the hand rests on the vertical or horizontal plane, we have 6 different fillings.
- *Rotation*: as for fillings, they identify modifications of a same base symbol; as an example, in some configurations rotation allows users to distinguish hand orientations, i.e. how it is turned, and the used hand (left or right).

The main difficulty in the definition of a complete concrete for representations of Sign Languages is in their four-dimensional quality (three spatial dimensions plus the temporal one) which requires a higher degree of freedom in arranging the base structures that express signs in two dimensions. As a consequence, the SWift interface allows great freedom in the composition and characteristics of aggregated glyphs. The proposal of an abstract syntax through the meta-model drafted in Figure 17 is at the basis of SWift and can serve as a guide to build a software framework where signs might be synthesized as well as analyzed. Issues related to its incorporation into UsiXML are a matter of discussion.

A first prototype has been implemented according to the concurrent design process performed with a group of deaf users (Figure 18). The interaction pattern reflects the absence of particular limitations on the sign composition. The user can select a specific body part, and is introduced to the set of variations and rotations provided for the glyphs pertaining to that group. Once the glyph has been chosen, it can be dragged and dropped in the board space.

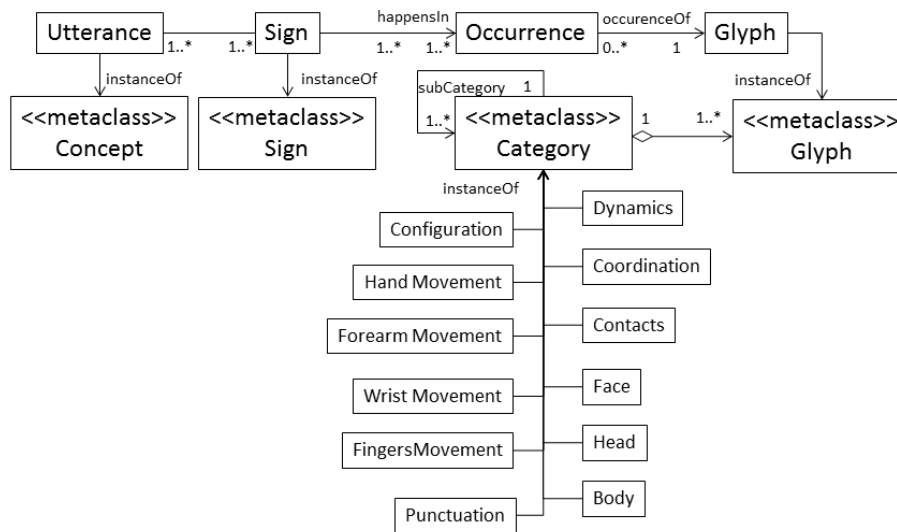


Figure 17. Metaclasses and classes in a metamodel for Sign Languages and Sign Writing.

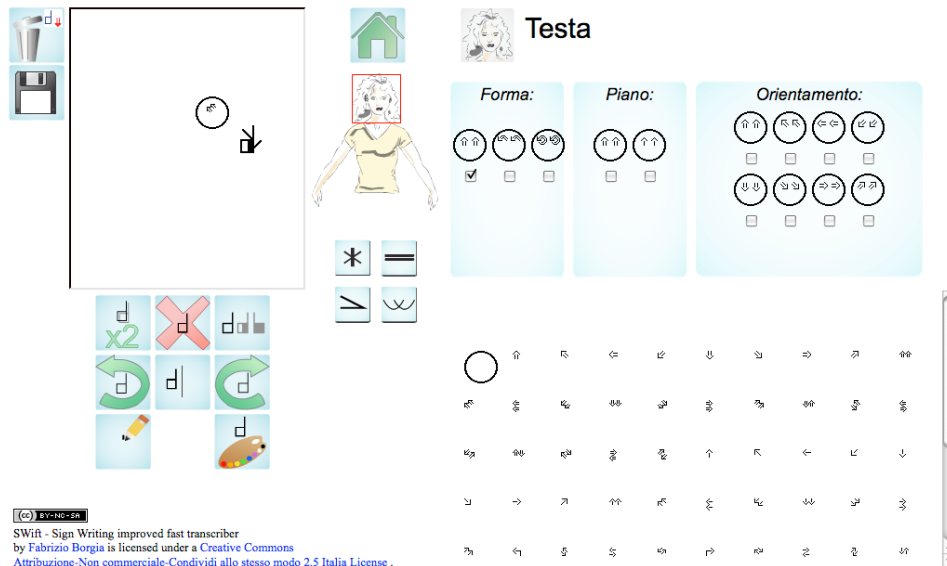


Figure 18. A screenshot of Swift.

## CONCLUSION

In this paper a model-driven approach to developing UIs for deaf people has been presented. A fully-iconic page structure is proposed to enhance deaf people's motivation while navigating in virtual environments. In fact, the iconic modality aims at leveraging the deaf-peculiar visual way of grasping information.

This iconic structure is applied to the pages of story-based learning paths, and the StoryEditor visual editor has been presented as a powerful tool for manipulating the two levels of stories description, i.e. iconic and diachronic. Finally, the written representation of Sign Languages has been taken into account, proposing a meta-model for the SignWriting code extendible to Sign Languages in general, and which can be the basis for incorporating specifications for this form of interaction into UsiXML.

## ACKNOWLEDGMENTS

The preparation of this work was partially funded by the Italian Ministry of Education and Research (MIUR-FIRB), Project *E-learning, Deafness and Written Language/ VISEL* – RBNE074T5L (2009-2012) <http://www.visel.cnr.it>.

## REFERENCES

1. Antinoro Pizzuto, E., Bianchini, C.S., Capuano, D., Gianfreda, G., and Rossini, P. Language Resources and Visual Communication in a Deaf Centered Multimodal E-Learning Environment: Issues to be Addressed. In *Proc. of the first LREC 2010 Workshop on Supporting eLearning with Language Resources and Semantic Data* (Valletta, May 22, 2010). P. Monachesi, Gliozzo, A.M., and Westerhout (Eds.). 2010, pp. 18-23. Available at <http://it.science.cmu>.

- ac.th/ejournal/modules/journal/file/11-04-26-16c22.pdf
2. Bolognesi, T., and Brinksma, E. Introduction to the ISO Specification Language LOTOS. *Computer Networks and ISDN Systems* 14, 1 (1987), pp.25-59.
  3. Bottoni, P., Capuano, D., De Marsico, M., Labella, A., and Levialdi, S. DELE: a Deaf-centered E-Learning Environment. *Chiang Mai J. Sci.* 38, 1 (2011), pp. 31-57.
  4. Bruner, J.S. The Narrative Construction of Reality. *Critical Inquiry* 18, 1 (1991), pp. 1-21.
  5. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J. A unifying reference framework for multi-target user interfaces. *Interacting with Computers* 15,3 (200), pp. 289–308
  6. Capuano, D., De Monte, M.T., Roccaforte, M., Tomasuolo, E., and Groves, K.M. A Deaf-Centered e-Learning Environment (DELE): challenges and considerations. *Journal of Assistive Technologies, special issue on children with speech disabilities* 5, December 2011
  7. Cox, S., Lincoln, M., Tryggvason, J., Nakisa, M., Wells, M., Tutt, M., and Abbott, S. TESSA, a system to aid communication with deaf people. In *Proc. of the 5<sup>th</sup> Int. ACM Conf. on Assistive technologies SIGCAPH'2002* (Edinburgh, July 08-10, 2002). ACM Press, New York (2002), pp. 205–212.
  8. Elliott, R., Glauert, J., Jennings, V., and Kennaway, R. An overview of the SiGML notation and SiGMLSigning software system. In *Proc. of 4th Int. Conf. on Language Resources and Evaluation LREC'2004*. O. Streiter and C. Vettori (Eds.), 2004, pp. 98–104.
  9. Göhner, P., Kunz, S., Jeschke, S., Vieritz, H., and Pfeiffer, O. Integrated Accessibility Models of User Interfaces for IT and Automation Systems. In *Proc. of the ISCA 21<sup>st</sup> Int. Conf. on Computer Applications in Industry and Engineering CAINE'2008* (Honolulu, November 12-14, 2008). ISCA (2008), pp. 280-285.
  10. Haesen, M., Van den bergh, J., Meskens, J., Luyten, K., Degrandart, S., Demeyer, S., and Coninx, K. Using Storyboards to Integrate Models and Informal Design Knowledge. In *Model-Driven Development of Advanced User Interfaces*. Hussmann, H., Meixner, G., Zuehlke, D. (Eds.). Studies in Computational Intelligence, vol. 340. Springer, Berlin (2011), pp. 87-106.
  11. Johnson M. *The meaning of the body*. University of Chicago Press (2007).
  12. Lakoff, G., and Johnson, M. *Metaphor We Live By*. University of Chicago Press (1980).
  13. McDrury, J., and Alterio, M. *Learning Through Storytelling in Higher Education: Using Reflection & Experience to Improve Learning*. Dunmore Press (2002).
  14. Paternò, F. *Model-Based Design and Evaluation of Interactive Applications*. Springer, Berlin (1999).
  15. Paternò, F., Santoro, C., and Spano, L.D. MARIA: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. *ACM Trans. On Computer-Human Interaction* 16, 4 (2009).
  16. Sharma, R. *et al.* Speech-Gesture Driven Multimodal Interfaces for Crisis Management. In *Proc. of the IEEE*, 91, 9 (2003), pp. 1327–1354.
  17. Spano, L.D. A model-based approach for gesture interfaces. In *Proc. EICS'2011*. ACM Press, New York (2011), pp. 327-330.
  18. Stanciulescu, A., Limbourg, Q., Vanderdonckt, J., Michotte, B., and Montero, F. A Transformational Approach for Multimodal Web User Interfaces based on UsiXML. In *Proc. of 7<sup>th</sup> Int. Conf. on Multimodal Interfaces ICMI'2005* (Trento, 4-6 October 2005). ACM Press, New York (2005), pp. 259-266.
  19. Szechter, L.E., and Liben, L.S. Parental Guidance in Preschoolers' Understanding of Spatial-Graphic Representations. *Child Development* 75, 3 (2004), pp. 869-885.
  20. Van Hees, K., and Engelen, J. Non-visual Access to GUIs: Leveraging Abstract User Interfaces. In *Proc. ICCHP'2006*, pp. 1063-1070.



# Concurrent Multi-Target Runtime Reification of User Interface Descriptions

Kris Van Hees, Jan Engelen

Katholieke Universiteit Leuven, ESAT/SCD/DocArch, Kasteelpark Arenberg 10,  
B-3001 Leuven-Heverlee (Belgium)  
kris@alchar.org, jan@docarch.be

## ABSTRACT

While providing non-visual access to graphical user interfaces has been the topic of research for over 15 years, blind users still face many obstacles when using computer systems. Existing solutions are largely based on either graphical toolkit hooks, queries to the application and environment, and scripting, or model-driven user interface development or runtime adaptation. Parallel User Interface Rendering (PUIR) provides a novel approach based on past and current research into Abstract User Interfaces (AUIs), User Interface Description Languages (UIDLs), and accessibility. The framework presented here provides a mechanism to render a user interface simultaneously in multiple forms (e.g., visual and non-visual). The research this paper is based on has a primary focus on providing accessibility for blind users.

## Author Keywords

Accessibility, human-computer interaction, multi-modal interfaces, universal Access, user interface description language.

## General Terms

Design, Human Factors, Theory

## Categories and Subject Descriptors

H.5 [Information Interfaces and Presentation]: User Interfaces

## INTRODUCTION

Over the past few years, our world has become more and more infused with devices that feature Graphical User Interfaces (GUIs), ranging from home appliances with LCD displays to mobile phones with touch screens and voice control. The emergence of GUIs poses a complication for blind users due to the implied visual interaction model. Even in the more specific field of general purpose computer systems at home and in the work place, non-visual access often still poses a problem, especially in Unix-based environment. While popularity keeps growing for this group of systems, advances in accessibility technology in support of blind users remain quite limited. Common existing solutions are built on toolkit extensions, scripting, and complex heuristics to obtain sufficient information in order to build an off-screen model (OSM) as basis for a non-visual rendering [14,27,36]. Other approaches use model-driven UI composition and/or runtime adaptation [7,26].

The flexibility of X Windows-based graphical environments introduces an additional level of complexity. Not only is the user faced with a primarily visual interaction model, but it is also very common to combine elements from a variety of graphical toolkits into a single desktop environment. Part and current research indicates that abstracting the User Interface (UI) offers a high degree of flexibility in rendering for a multitude of output modalities. Leveraging the adoption of UI development using abstract user interface definitions, we can build a solid base for providing non-visual access as an integral component of the UI environment [31,32,33,34,35]. Rather than providing a non-visual rendering as a derivative of the visual form, representations of the UI are rendered concurrently at runtime based on the same abstract UI description, presented as a UIDL document. This approach ensures coherence between the representations, and enhances collaboration between users with differing abilities.

The remainder of this paper first presents related work on GUI accessibility, UIDLs, and UI abstraction. The third section provides a discussion of the design principles behind the parallel user interface rendering approach, and is followed by the design of a proof-of-concept implementation. The fifth section discusses the use of a UIDL to specify the UI for runtime reification. Section six concludes this paper, offering conclusions and a view on future work.

## RELATED WORK

Accessibility of GUIs for blind users has been the topic of research for many years. Mynatt and Weber discussed two early approaches [16], introducing four core design issues that are common to non-visual access to GUIs. Expanding on this work, Gunzenhäuser and Weber phrased a fifth issue, along with providing a more general description of common approaches towards GUI accessibility [9]. Weber and Mager provide further details on the various existing techniques for providing a non-visual interface for X11 by means of toolkit hooks, queries to the application, and scripting [36].

Blattner *et al.* introduce the concept of MetaWidgets [3], abstractions of widgets as clusters of alternative representations along with methods for selecting among them. Furthermore, any specific manifestation of a metawidget is inherently ephemeral, meaning that as time passes, the appearance will change. MetaWidgets can handle these temporal semantics as part of the object state.

The “Fruit” system described by Kawai, Aida, and Saito [10] addresses an important aspect of user interface accessibility: dynamic UI customisation. It is based on an abstract widget toolkit, and the necessity to separate the user interface from the application logic. Application software is still written as if a graphical widget toolkit is being used, while the actual presentation of the user interface is handled by device-specific components. The “Fruit” system does not support synchronised presentation in multiple modalities, nor does it provide any form of accessibility at the level of the windowing environment.

Savidis and Stephanidis researched alternative interaction metaphors for non-visual user interfaces [21], which formed the basis for the HAWK toolkit [23]. It provides interaction objects and techniques that have been designed specifically for non-visual access. The HOMER UIMS [21,22] uses this toolkit to provide blind users with a custom interface alongside the visual interface. This dual interface concept builds on a UIDL-based source document for UI specification.

The use of abstract user interfaces is largely based on the observation that application UIs and World Wide Web forms are very similar. Barnicle [1], Pontelli *et al.* [17], and Theofanos/Redish [28] all researched the obstacles that blind users face when dealing with user interaction models, confirming this observation.

The Views system described by Bishop and Horspool [2] introduces the concept of runtime creation of the user interface representation based on an XML specification of the UI. Independently, Stefan Kost also developed a system to generate user interface representations dynamically based on an abstract UI description [13].

His thesis centred on the modality-independent aspect of the AUI description, providing a choice of presentation toolkit for a given application. His work touches briefly on the topic of multiple interface representations, offering some ideas for future work in this area, while identifying it as an area of interest that faces some significant unresolved issues.

User Interface Description Languages (UIDLs) have been researched extensively throughout the past eight to ten years. Souchon and Vanderdonckt [25] reviewed 10 different XML-compliant UIDLs, finding that no single UIDL satisfies their requirements for developing fully functional UIs. Trewin, Zimmermann, and Vanderheiden [29, 30] present technical requirements for abstract user interface descriptions based on Universal Access and “Design-for-All” principles, and they evaluated four different UIDLs based on those requirements. The authors noted that further analysis at a more detailed level is required in order to provide a realistic assessment.

User Interface eXtensible Markup Language (UsiXML) [15] is aimed at “capturing the essential properties [...] that

turn out to be vital for specifying, describing, designing, and developing [...] UIs”: [15]. Of special importance are:

- The UI design should be independent of any modality of interaction.
- It should support the integration of all models used during UI development (context of use, user, platform, and environment, ...).
- It should be possible to express explicit mappings between models and elements.

Building on the growing interest in AUI descriptions, Draheim *et al.* introduced the concept of “GUIs as documents” [5]. The authors provide a detailed comparison of four GUI development paradigms, proposing a document-oriented GUI paradigm where editing of the graphical user interface can take place at application runtime. In the discussion of the document-based GUI paradigm, they write about the separation of GUI and program logic: “This makes it possible to have different GUIs for different kinds of users, e.g. special GUIs for users with disabilities or GUIs in different languages. Consequently, this approach inherently offers solutions for accessibility and internationalization.” The idea did not get developed further, however.

## DESIGN PRINCIPLES

Extensive research into the design of the graphical user interface, universal access requirements, UIDLs, and existing approaches for assistive technology drove the formulation of four design principles that are fundamental to the design of the PUIR framework. This section discusses each design principle in some detail, highlighting the importance of each principle.

### A consistent conceptual model with familiar manipulatives as basis for all representations

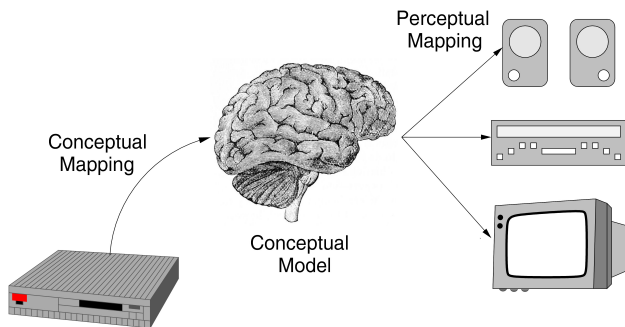
When circa 1974 a team at Xerox PARC developed the concept of the Graphical User Interface, they may not have realized that they actually laid the foundation of a much broader and powerful entity:

***Metaphorical User Interface (MUI):** An abstract user interface that uses a metaphor of the physical world to model the operational characteristics of an application using concepts familiar to the user population.*

Central to the MUI is the metaphor of the physical world, essentially establishing a conceptual model that users are comfortable with. Smith *et al.* define a user’s conceptual model as [24]:

***Conceptual Model:** The set of concepts a person gradually acquires to explain the behaviour of a system.*

Given the need to define a conceptual model for a user interface, designers have essentially two choices: design the user interface based on a existing model employing familiar metaphors, or develop a brand new model.



**Figure 1. The role of the conceptual model.**

Extensive research done at Xerox PARC led to the conclusion that the metaphor of a physical office is an appropriate model [24]. It is however important to note that this conclusion was reached in function of developing a user interface for visual presentation, and non-visual interaction was therefore not taken into consideration.

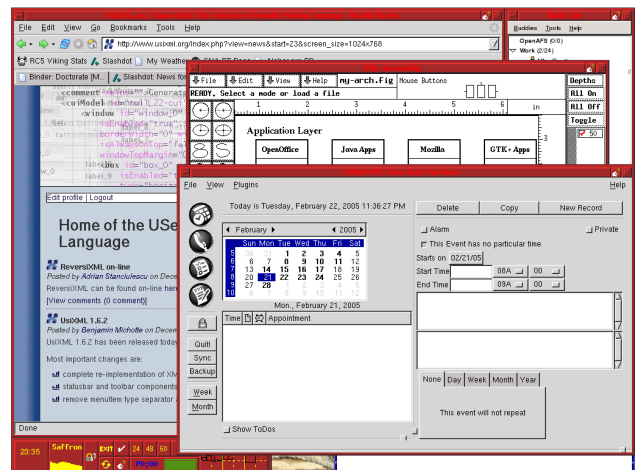
In view of the fact that the early GUI work can be generalised outside the constraints of any specific representation, one can conclude that the conceptual model is more of a “user illusion” [11] than a strict metaphor. It establishes a mental model that users can relate to (because it is based on familiar concepts), hiding the complexities of the internal working of the computer system. The presentation of the application UI based on the conceptual model is then accomplished by means of a perceptual mapping onto a multi-modal environment as described by Gaver [8] (Fig. 1).

Can a single conceptual model serve users with distinctly different abilities? The modality-independent nature of the model would indicate that there is no implicit visual aspect to it. While sighted (and many blind) individuals tend to use visual imaginary when reasoning about the metaphor of the virtual office or the desktop, research shows that this is not a necessity [6, 19]. Furthermore, the underlying concepts are generally familiar to all users, regardless of their abilities. It is obvious that a blind individual can be quite productive in a physical office, regardless of the fact that the spatial configuration of offices is predominantly based on visual concepts.

It is clear that a single conceptual model is appropriate for sighted and blind users if a clear separation between the perceptual and the conceptual is maintained. There is therefore no need to consider a separate non-visual UI design at the conceptual level.

#### **Use of multiple toolkits across the GUI environment**

Providing access to GUIs for blind users would be relatively easy if one could make the assumption that all applications are developed using a single standard graphical toolkit *and* if that toolkit provides a sufficiently feature-rich API for assistive technology. Unfortunately, this situation is not realistic. While the majority of programs under MS Windows are developed based on a standard toolkit, the provided API still lacks functionality that is necessary to ensure full accessibility of all applications.



**Figure 2. X11 session with multiple graphical toolkits.**

X Windows does not impose the use of any specific toolkit, nor does it necessarily promote one. It is quite common for users of a UNIX-based system to simultaneously use any number of applications that are each built upon a specific graphical toolkit. Some applications even include support for multiple graphical toolkits, providing the user with a configuration choice to select a specific one (see Figure 2). In order to be able to provide access to application user interfaces regardless of the graphical toolkits they are developed against, the chosen approach must ensure that the provision of non-visual access is not only medium-independent but also toolkit-independent.

#### **Collaboration between sighted and blind users**

In order to ensure that segregation of blind users due to accessibility issues can be avoided, appropriate support for collaboration between the two user groups is important. This collaboration can occur in different ways, each with its own impact on the overall requirements for the accessibility of the environment.

Savidis and Stephanidis [22] consider the need for collaboration based on proximity between users. When considering a mix of sighted and blind users, proximity is however less relevant because the overall characteristics of interaction remain the same regardless of whether the participants are local or remote, provided that an adequate communication channel is available.

Upon analysis of the substance of the collaboration, it becomes clear that there are two distinct levels of interaction (applying [8] in the broader context of interaction between users): perceptual vs conceptual. When collaboration occurs at the perceptual level, users discuss the details of manipulating specific controls in a specific modality. Conceptual collaboration involves interaction at a higher level, discussing the semantics of user interaction instead. At this level, users reason and interact based on a shared conceptual model that is independent from any modality.

### Coherent concurrently accessible representations

A common problem with existing approaches for non-visual access to GUIs is related to the use of an off-screen model: lack of coherence between the visual and the non-visual interfaces. Mynatt and Weber identified this as one of the important HCI issues concerning non-visual access [16].

Entry Form

First Middle Last

Full name No spaces

Occupation

Figure 3. Example of a visual layout that can confuse screen readers.

Real Text

This is a Java TextArea that contains more characters per line than can be displayed. It uses a viewport to display a subset of the data, both horizontally and vertically. Screenreaders often cannot determine what part of the data is visible.

Viewport Text

ers per line than can a viewport to display a, both horizontally a readers often cannot the data is visible

Figure 4. Example of the effects of a viewport on text visualisation.

The problem is most often related to the information gathering process that drives the construction of the OSM. Kochanek provides a detailed description of the construction-process for an off-screen model for a GUI [12]. Limitations in being able to obtain accurate information and/or to interpret the information tend to lead to this lack of coherence. Figure 3 shows an example where lack of semantic relation information can confuse a screen reader (i.e. what is the label, if any, for each of the text input fields). In Figure 4, a blind user would typically be presented with an equivalent of the view at the left whereas a sighted user would be presented with the (more limited) view on the right. This leads to significant difficulties if the two users wish to collaborate concerning the operation of the application and the content of the text area.

By ensuring that representations are rendered based on a single source, it is possible to provide all users direct access to the system. This is also a requirement for successful collaboration. The requirements to provide direct access for all are:

- Users can access the system concurrently.
- Users can interact with the system using metaphors designed to meet their specific needs.
- Coherence between UI representations is assured.

### PARALLEL UI RENDERING

Using the design principles presented in the previous section, a framework has been designed for providing access to graphical user interfaces for blind users. This section provides details on the various components.

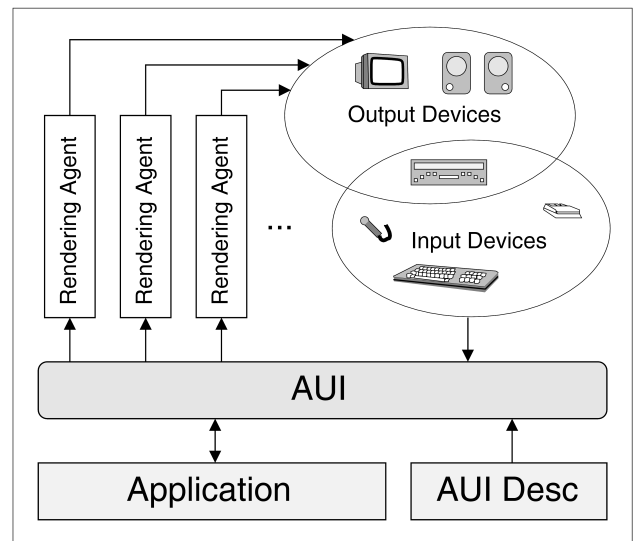


Figure 5. Schematic overview of Parallel User Interface Rendering.

Figure 5 provides the schematic overview of the Parallel User Interface Rendering approach. Rather than constructing the UI programmatically with application code that executes function calls into a specific graphical toolkit, applications provide a UI description in abstract form, expressed in a UIDL. This authoritative AUI description is processed by the AUI engine, and a runtime model of the UI is constructed. The application can interact with the AUI engine to provide data items for UI elements (e.g. text to display in a dialog), to query data items from them (e.g. user input from a text input field), or to make runtime changes in the structure of the UI. The AUI engine implements all application semantics, ensuring that the functionality does not depend on any specific modality.

The representation of the UI is delegated to modality specific rendering agents, using the UI model at their source of information. At this level, the AUI is translated into a concrete UI (CUI), and the appropriate widget toolkit (typically provided by the system) is used to present the user with the final UI (FUI) by means of specific output devices. Therefore, the UI model that is constructed by the AUI engine serves as information source for all the different rendering agents. All representations of the UI are created equally, rather than one being a derivative of another<sup>11</sup>. The application cannot interact with the rendering agents directly, enforcing a strict separation between application logic and UI representation.

<sup>11</sup> It is important to note that it is not a requirement that all representations are generated at runtime, although development time construction of any representations could imply that dynamic updates to the UI structure are not possible.

The handling of user interaction events from input devices<sup>12</sup> occurs at the AUI engine level. The PUIR framework is based on meaningful user interaction, and therefore only semantic user interaction events are given any consideration. Given that events are typically presented to toolkits by means of OS level device drivers, and the fact that these event sources are very generic<sup>13</sup>, additional processing is required in order for the PUIR framework to receive the semantic events it depends on.

#### RUNTIME REIFICATION OF UI DESCRIPTIONS

The design principles and the actual design described in the previous sections do not impose a specific requirement that the PUIR approach be based on UI descriptions in any specific UIDL, or in any source format whatsoever. It is conceivable to implement this framework based on programmatically specified UI designs. Program code can construct the UI as if the widgets at the AUI engine level actually comprise a UI user interface toolkit (Rose et al. describe such a system [18]). Why then does the PUIR design incorporate the concept of runtime reification of UI descriptions?

The specification of user interface semantics within the context of the conceptual model establishes an abstraction of the UI, regardless of how that AUI is represented. Limbourg *et al.* provide definitions for some important concepts [15]:

**Abstract User Interface (AUI):** *A canonical expression of the rendering of domain concepts and tasks (conceptual model) in a way that is independent from any modality of interaction.*

**Concrete User Interface (CUI):** *A reification of an AUI within the context of an abstracted modality, defining a specific “Look & Feel”, independent from any computing platform (devices).*

**Final User Interface (FUI):** *A final representation of a CUI within the context of a specific computing platform (devices).*

The only truly valid source of information for providing multiple coherent representations is at the AUI level because it is modality independent. From an implementation perspective there is very little difference between encoding the AUI in the application and providing it as a UIDL-based description. Both approaches result in an object hierarchy that forms the basis for rendering the UI in the re-

quired modalities. However, the use of a textual description does offer a great degree of flexibility in various areas:

- **Maintainability:** Various changes can take place in the UI description without requiring a program code rebuild.
- **Customisation:** Many modalities allow for custom attributes to be associated with widgets, ranging from background and foreground colours, fonts to iconic images. Since the AUI layer cannot know the various supported attributes, such information can only be encoded in the application as arbitrary data items, not unlike textual specifications.
- **Expandability:** In view of Universal Access and the multitude of needs that users might have, an almost endless range of custom modalities may present themselves. Providing support for as yet unknown rendering agents is quite complex, and textual UI descriptions offer a greater level of flexibility in handling this situation.
- **Appeal:** While less scientific in nature, the overall appeal of the approach can be significant. Developers are less likely to adopt a change in how they operate unless they find a compelling reason to do so. Writing program code to build a UI based on an abstract widget toolkit, just to have that construct be transformed into a CUI, and then being presented to the user as the FUI, is less than ideal. But when a change to UIDL-based UI descriptions allows for the previous three benefits, the new approach is likely to be more appealing.
- **Familiarity:** When comparing a form on a web page with an application UI where data entry is expected to occur, striking similarities can be observed. Both feature almost identical UI elements: buttons, drop-down lists, text entry fields, and labels. In addition, the obstacles that blind users face when using web forms [17,28] are known to be very similar to the obstacles they face when interacting with GUIs [1]. Furthermore, HTML documents are essentially abstract descriptions although specific modality dependent information can be embedded in the document as augmentation to the abstract description.

One possible solution could be to make the AUI description available alongside the application, to be used as an information source in support of AT solutions (i.e. the Glade project [4]). Because the implementation of the UI is still generally hardcoded in the application, this approach does open up the possibility that inconsistencies between the application and the UI description occur (This is a common problem in any circumstance where essentially the same information is presented in two different locations). In support of the coherence design principle, the PUIR framework is designed on the concept that all representations are to be reified from the same AUI. This is a significant paradigm shift from the majority of AT solutions that are still implemented as a derivative of the GUI.

<sup>12</sup> The physical devices that the user employs to perform operations of user interaction with the application.

<sup>13</sup> Device drivers at the OS level are meant to serve all possible consumers. The events they generate are most commonly very low-level events.

This paradigm shift from representing the UI by means of program code in the application to utilising a system that interprets and renders the UI based on an AUI description document has slowly been taking place for the past ten to twelve years. Yet, the shift has not progressed much past the point of using the AUI description as part of the development process. The preceding discussion shows that it is possible (and necessary for this work) to complete the shift to what Draheim, et al. refer to as “the document-based GUI paradigm” [5]. Expanding the notion of the representation of the UI description to the realm of concurrent alternative representations, this can be extended as “the document-based UI paradigm”. The advantages of this approach are significant, although there are also important trade-offs:

- *Separation of concerns between UI and application logic.* This has been identified as (part of) an important technical requirement for AUI description languages, but the very use of AUI descriptions also enforces this concept through the need for a well-defined mechanism to incorporate linking UI elements to program logic. This also implies a trade-off in flexibility because the application logic is limited in its ability to directly interact with the UI.
- *Maintainability of the application.* When a UI is described programmatically as part of the application, it typically will have a stronger dependency on system features such as the presentation toolkit that it is developed for. AUI descriptions do not exhibit this complication because they are toolkit independent. In addition, the document-based nature of AUI makes it much easier to modify the UI for small bug fixes, whereas a code-based UI requires changes in the application program code.
- *Adaptability.* The adaptability of code-based UIs is generally limited to toolkit-level preference-controlled customisation, whereas the ability to make changes to the AUI description at runtime (e.g., by means of transformation rulesets) provides for a high level of adaptability.

The document-based UI paradigm is powerful, but it imposes some limitations on the designer and developer. Toolkits for code-based UI development generally offer a higher level of flexibility to the developer because they make lower level (lexical and/or syntactic) elements available. A rendering agent that creates a UI representation based on an AUI description provides higher level constructs, offering a higher level of consistency and stability, but at the cost of some flexibility.

The AUI → CUI → FUI reification transformation process that essentially takes place when a UI description is used to create the actual UI representation in the context of a mo-

dalidity commonly requires<sup>14</sup> additional information that cannot be determined automatically.

Given a sufficiently expressive UIDL, the UI description can incorporate annotations or attributes that are rendering specific. While the AUI engine has no use for this information, it can be made available to one or more specific rendering agents. Since the information can be associated as a widget level, a high degree of customisation in terms of presentation is possible. This approach *is* different from e.g. systems where the UI description encapsulates multiple object hierarchies, one for each level of abstraction (i.e., AUI, CUI, and FUI).

## CONCLUSION

Parallel User Interface Rendering is proving to be a very powerful technique in support of the Design-for-All and Universal Access principles. It builds on a solid base of research and development of abstract user interfaces and UIDLs, and it provides a framework where non-visual rendering of the UI operates at the same level as the visual rendering rather than as a derivative. The design principles build a foundation for a very powerful approach. Especially user collaboration benefits greatly from this system because of the coherence between all renderings, allowing communication about user interaction to be based on a substantially similar mental model of the user interface.

The current proof-of-concept implementation is based on a custom UIDL, but for further development<sup>15</sup> collaboration with a UICL is expected. The decision to continue with a broader interpretation of the original GUI design principles drives the choice of underlying UIDL. The ability of e.g. UsiXML to capture the UI with models at different levels of abstraction can be a real asset to this novel approach.

The PUIR framework can contribute to the field of accessibility well beyond the immediate goal of providing non-visual access to GUIs. The generic approach behind the PUIR design lends itself well to developing alternative rendering agents in support of other disability groups. Because rendering agents need not necessarily execute local to applications, accessible remote access is possible as well. The PUIR framework may also benefit automated application testing, by providing a means to interact with the application programmatically without any dependency on a specific UI rendering.

## ACKNOWLEDGEMENTS

The research presented in this paper is part of the author’s doctoral work at Katholieke Universiteit Leuven, Belgium, under supervision by Prof. dr. ir. Jan Engelen (ESAT - SCD - Research Group on Document Architectures).

<sup>14</sup> Or at a minimum, “benefits from” because presentation details are important to many users.

<sup>15</sup> And in order to work towards a possible future adoption as an AT support solution.



## REFERENCES

1. Barnicle, K. Usability testing with screen reading technology in a windows environment. In *Proceedings of the ACM Conf. on Universal Usability CUU'2000*. ACM Press, New York (2000), pp. 102–109.
2. Bishop, J. and Horspool, N. Developing principles of GUI programming using views. In *Proc. of the 35<sup>th</sup> SIGCSE Technical Symposium on Computer science education SIGCSE'2004*. ACM Press, New York (2004), pp. 373–377.
3. Blattner, M., Glinert, E., Jorge, J., and Ormsby, G. Metawidgets: towards a theory of multimodal interface design. In *Proceedings of the 16th Annual International Computer Software and Applications Conference COMPSAC'1992*. IEEE Computer Society Press, Los Alamitos (September 1992), pp. 115–120.
4. Chapman, M. Create user interfaces with Glade. *Linux J.* 87 (July 2001), pp. 90–94.
5. Draheim, D., Lutteroth, C., and Weber, G. Graphical user interfaces as documents. In *Proc. of the 7<sup>th</sup> ACM SIGCHI New Zealand chapter's International Conference on Computer-human interaction: design centered HCI CHINZ'2006*. ACM Press, New York (2006), pp. 67–74.
6. Edwards, A.D.N. The difference between a blind computer user and a sighted one is that the blind one cannot see. Interactionally Rich Systems Network. Working Paper No. ISS/WP2 (1994).
7. Gajos, K. and Weld, D. S. SUPPLE: Automatically generating user interfaces. In *Proceedings of the 9<sup>th</sup> ACM Int. Conf. on Intelligent User Interfaces IUI'2004*. ACM Press, New York (2004), pp. 93–100.
8. Gaver, W.W. The sonicfinder: an interface that uses auditory icons. *Human-Computer Interaction* 4, 1 (March 1989), pp. 67–94.
9. Gunzenhäuser, R. and Weber, G. Graphical user interfaces for blind people. In *Proc. of 13<sup>th</sup> World Computer Congress WCC'94, Volume 2*. K. Brunnstein and E. Raubold (Eds.). Elsevier Science B.V., Amsterdam (1994), pp. 450–457.
10. Kawai, S., Aida, H., and Saito, T. Designing interface toolkit with dynamic selectable modality. In *Proc. of the 2<sup>nd</sup> Annual ACM Conf. on Assistive Technologies ASSETS'1996*. ACM Press, New York (1996), pp. 72–79.
11. Kay, A.C. User interface: A personal view. In *The Art of Human-Computer Interface Design*. B. Laurel (Ed.), Addison-Wesley Publishing Co., New York (1990), pp. 191–207.
12. Kochanek, D. Designing an offscreen model for a GUI. In *Computers for Handicapped Persons*. W. Zagler, G. Busby, and R. Wagner (Eds.). Lecture Notes in Computer Science, vol. 860. Springer, Berlin (1994), pp. 89–95.
13. Kost, S. *Dynamically generated multi-modal application interfaces*. PhD thesis, Technische Universität Dresden, Dresden (2006).
14. Kraus, M., Völkel, T., and Weber, G. An off-screen model for tactile graphical user interfaces. In *Computers Helping People with Special Needs*, K. Miesenberger, J. Klaus, W. Zagler, and A. Karshmer (Eds.). Lecture Notes in Computer Science, vol. 5105. Springer, Berlin (2008), pp. 865–872.
15. Limbourg, Q. and Vanderdonckt, J. UsiXML: A User Interface Description Language Supporting Multiple Levels of Independence. In *Engineering Advanced Web Applications*, M. Matera, S. Comai, S. (Eds.). Rinton Press, Paramus (2004), pp. 325–338.
16. Mynatt, E. D. and Weber, G. Nonvisual presentation of graphical user interfaces: contrasting two approaches. In *Proc. of the ACM Conf. on Human factors in Computing Systems CHI'1994*. ACM Press, New York (1994), pp. 166–172.
17. Pontelli, E., Gillan, D., Xiong, W., Saad, E., Gupta, G., and Karshmer, A. I. Navigation of HTML tables, frames, and XML fragments. In *Proc. of the 5<sup>th</sup> ACM Int. Conf. on Assistive Technologies ASSETS'2002*. ACM Press, New York (2002), pp. 25–32.
18. Rose, D., Stegmaier, S., Reina, G., Weiskopf, D., and Ertl, T. Non-invasive adaptation of black-box user interfaces. In *Proc. of the 4<sup>th</sup> Australasian User Interface Conference on User interfaces AUIC'2003*. Vol. 18. Australian Computer Society, Inc. (2003), pp. 19–24.
19. Sacks, O. The mind's eye: What the blind see. *The New Yorker* (28 July 2003), pp. 48–59.
20. Savidis, A. and Stephanidis, C. Building non-visual interaction through the development of the rooms metaphor. In *Conference companion on Human factors in computing systems CHI '95*. ACM Press, New York (1995), pp. 244–245.
21. Savidis, A. and Stephanidis, C. Developing dual user interfaces for integrating blind and sighted users: the HOMER UIMS. In *Proc. of the ACM Conf. on Human factors in computing systems CHI'95*. ACM Press/Addison-Wesley Publishing Co., New York (1995), pp. 106–113.
22. Savidis, A. and Stephanidis, C. The HOMER UIMS for dual user interface development: Fusing visual and non-visual interactions. *Interacting with Computers* 11, 2 (1998), pp. 173–209.
23. Savidis, A., Stergiou, A., and Stephanidis, C. Generic containers for metaphor fusion in non-visual interaction: the HAWK interface toolkit. In *Proc. of the Interfaces '97 Conference* (1997), pp. 194–196.
24. Smith, D. C., Harslem, E. F., Irby, C. H., Kimball, E. B., and Verplank, W. L. Designing the Star User Interface. *BYTE* (April 1982), pp. 242–282.
25. Souchon, N., and Venderdonckt, J. A review of XML-compliant user interface description languages. In

- Interactive Systems. Design, Specification, and Verification*, J.A. Jorge, N. Jardim Nunes, and J. Falcão e Cunha (Eds.). Lecture Notes in Computer Science, vol. 2844. Springer, Berlin (2003), pp. 391–401.
26. Stephanidis, C. and Savidis, A. Universal access in the information society: Methods, tools, and interaction technologies. *Universal Access in the Information Society* 1, 1 (2001), pp. 40–55.
  27. Sun Microsystems. GNOME 2.0 desktop: Developing with the accessibility framework. Tech. report. Sun Microsystems (2003).
  28. Theofanos, M. F. and Redish, J. G. Bridging the gap: between accessibility and usability. *Interactions* 10, 6 (November 2003), pp. 36–51.
  29. Trewin, S., Zimmermann, G., and Vanderheiden, G. Abstract user interface representations: how well do they support universal access? In *Proc. of the ACM Conf. on Universal Usability CUU'2003*. ACM Press, New York (2003), pp. 77–84.
  30. Trewin, S., Zimmermann, G., and Vanderheiden, G. Abstract representations as a basis for usable user interfaces. *Interacting with Computers* 16, 3 (2004), pp. 477–506.
  31. Van Hees, K. and Engelen, J. Abstract UIs as a long-term solution for non-visual access to GUIs. In *Proc. of the 3<sup>rd</sup> Int. Conf. on Universal Access in Human-Computer Interaction UAHCI'2005*. Springer, Berlin (2005).
  32. Van Hees, K. and Engelen, J. Abstracting the graphical user interface for non-visual access. In *Proc. of 8<sup>th</sup> European Conference for the Advancement of Assistive Technology in Europe*. A. Pruski and H. Knops (Eds.). IOS Press, Amsterdam (2005), pp. 239–245.
  33. Van Hees, K. and Engelen, J. Non-visual access to GUIs: Leveraging abstract user interfaces. In *Proc. of Int. Conf. on Computers Helping People with Special Needs ICHPP'2006*. K. Miesenberger, J. Klaus, W. Zagler, and A. Karshmer (Eds.). Lecture Notes in Computer Science, vol. 4061. Springer, Berlin (2006), pp. 1063–1070.
  34. Van Hees, K. and Engelen, J. Parallel User Interface Rendering: Accessibility for custom widgets. In *Proc. of the 1<sup>st</sup> Int. AEGIS Conf.* (2010), pp. 17–24.
  35. Van Hees, K. and Engelen, J. PUIR: Parallel User Interface Rendering. In *Proc. of Int. Conf. on Computers Helping People with Special Needs ICHPP'2010*. K. Miesenberger, J. Klaus, W. Zagler, and A. Karshmer (Eds.). Lecture Notes in Computer Science, vol. 6179. Springer, Berlin (2010), pp. 200–207.
  36. Weber, G. and Mager, R. Non-visual user interfaces for X windows. In *Proc. of the 5<sup>th</sup> Int. Conf. on Computers helping people with special needs. Part II*, R. Oldenbourg Verlag GmbH (1996), pp. 459–468.

# User Interface Description Language Support for Ubiquitous Computing

Raúl Miñón, Julio Abascal

Laboratory of HCI for Special Needs, University of the Basque Country,  
Euskal Herriko Unibertsitatea, Manuel Lardizabal 1, 20018 Donostia (Spain)  
Tel.: +34 943 015113 - {raul.minon, Julio.abascal}@ehu.es

## ABSTRACT

This paper introduces a proposal tool, called SPA4-USXML, aimed to graphically create instances of task models, abstract user interfaces and multimedia resources models. This tool is feed with the description of services provided by ubiquitous environments and web services. The main goal is to help service designers to create abstract specifications of the services for the EGOKI adaptive system. EGOKI automatically generates user interfaces adapted to the different needs and abilities of people with special needs, in order to provide access to services offered in ubiquitous environments. Therefore, SPA4USXML is aimed to complement the EGOKI system, in order to grant better user experience to people with special needs and to enhance their autonomy and security in their daily routines.

## Author Keywords

Ubiquitous computing, Adaptive Systems, Accessibility, User Interface Description Language, Supportive Tool.

## General Terms

Design, Human Factors, Theory

## Categories and Subject Descriptors

H.5 [Information Interfaces and Presentation]: User Interfaces

## INTRODUCTION

Diverse types of digital services are locally provided through local machines in ubiquitous environments or by means of remote web services (as described in the specification of Web Services [31] or Google services [7]). In general, when a ubiquitous environment provides local services, each machine has its own communication protocols, so it requires a middleware layer that is able to offer, among other features, a common communication protocol and standard access to the different services. In addition, the middleware layer usually handles the discovery, communication and control of the different services [2]. Some types of middleware also provide abstract interfaces describing the functionality given by each of the services.

For our ubiquitous environment we use the Universal Control Hub (UCH) [27], which is an implementation of the standard Universal Remote Console (URC). UCH generates a simple user interface to control each service. Since each person may have different needs the interface provided by UCH is not always accessible to everyone.

In a previous project our Laboratory created the EGOKI system to avoid accessibility barriers [1]. EGOKI is an adaptive system devoted to provide accessible user interfaces to elderly people and people with special needs in ubiquitous environments. This system automatically generates final accessible user interfaces adapted to the needs of people requesting a specific service.

In order to perform the adaptation, EGOKI requires an abstract description of the user interface, provided by means of a User Interface Description Language (UIDL) [9, 25]. To this end, we adopted UIML (User Interface Markup Language), a declarative XML-compliant meta-language used to specify user interfaces. In the current version of EGOKI, UIML documents are created manually by the designer of the ubiquitous service from a description of the functionality of the service provided by the middleware.

The UIML document cannot be generated automatically because the information provided by the middleware is not enough. Therefore, the designer of the service must enrich the description of the functionality, associate the interactive multimedia resources with the interaction elements of the abstract interface, and provide a structure for generating the final user interface.

This process has a number of disadvantages:

- The manual creation by the service designer of the UIML text file for each service takes too long and is very time inefficient.
- A textual representation of the abstract interface is less representative than a graphical interface making it easy to commit mistakes in designing the user interface and in turn more difficult to detect them.
- Maintaining and updating the user interface may be faster using a graphical (or mixed graphical-textual) user interface for the designer.
- The designer of the service must have a good command of XML [30] and UIML syntax.

This paper proposes the development of a graphical tool called SPA4USXML (Service Provider Annotations for Ubiquitous Services through UsiXML) in order to avoid these drawbacks. This tool is intended to assist the service designer in the process of creating abstract user interfaces. Using it, the design process will take place in a much faster and intuitive way, allowing the detection of errors at design

time and facilitating the modification of the resulting interfaces. In addition, the tool will allow enriching the services with additional resources. In this way the adaptation of interfaces, the service quality and the user experience will be improved.

Although currently EGOKI uses the UIML syntax, the tool SPA4USXML will generate model instances compliant with the UsiXML syntax [29]. The reason is that UsiXML covers the requirements of the enhanced next version of EGOKI better than UIML [16]. For instance, UIML does not provide the task models that are required for the new version of EGOKI. However, UsiXML provides a task model based on the CTT model of Paternò [23].

The rest of the paper is organized as follows: section two presents the related work; section three describes an application scenario. The architecture which integrates the tool and its objectives and functionality are described in section four; finally, in section five we discuss some open issues.

## RELATED WORK

As it has been previously mentioned, the tool SPA4USXML is a complement for the adaptive system EGOKI. Therefore, we start the state of the art analyzing several systems that have similar features to EGOKI. Subsequently we will revise some tools related to SPA4USXML.

In the last years the web has attracted the attention of numerous researches, who have developed several adaptive web systems [3]. However, there are few examples of systems devoted to adapt the content, presentation or navigation for people with special needs. Among them, we can mention AVANTI [26], which provides hypermedia information to adapt the content, navigation and presentation to people with disabilities.

On the other hand, SADIE [13] adapts news websites for blind users and SUPPLE [6] generates graphical user interfaces mainly for people with motor impairments.

Following these ideas, EGOKI adapts the user interface to the capabilities of people with special needs, selecting the most appropriate interaction resources to allow the user interaction.

Among the systems that provide access to ubiquitous services, the framework ViMos [10] supplies an architecture to dynamically generate user interfaces allowing the visualization of service information available in a particular context.

When the user devices used to access the ubiquitous service are diverse, each user interface requires a different configuration suited to each service and adapted to the user, such as the Ubiquitous Interactor [21]. In this case it may be required the use of a User Interface Description Language [9, 25].

For instance, Vermeulen *et al.* (2007) [32] propose a framework to design services that automatically present a suitable user interface on a wide variety of computing plat-

forms. Their main aim is to provide mobile users with flexibility to interact with services in a city environment. UIDLs allow the development of abstract user interfaces independent of the modality and the platform. This is also the approach followed for EGOKI, allowing the generation of different configurations of interfaces for a wide range of platforms and modalities.

Several tools have been developed to graphically generate diverse types of UIDLs. For example, Liquid Apps [12] allows graphically editing abstract user interfaces and afterward generating code compliant with the UIML syntax. On the other hand, GUMMY [14] is a generic multi-platform GUI builder designed to create user interfaces incrementally for a wide range of computing platforms. UIML is also the underlying UIDL used in Gummy.

There exist numerous tools that conform to the UsiXML syntax [11]. This syntax allows taking advantage of its architecture [4] and applying the language to different areas, such as the ubiquitous environments. Some of them have features that are coincident with our approach:

- *IdealXML* [19] is a tool that provides a simple editor used by application designers also to share knowledge. This tool allows the edition of task models (based on the tool CTTE [17]), abstract user interfaces models, domain models, and mapping models.
- *SketchiXML* [5] is a multi-agent application able to handle several kinds of hand-drawn sources as input and to provide the corresponding specification in UsiXML.
- *GUILayout++* [18] is a tool for designing user-centric interfaces through an iterative process based on prototyping and evaluation. It is able to automatically generate abstract user interfaces compliant with the UsiXML syntax starting from the prototype created.
- *GrafiXML* [15] allows designing and generating several UIs simultaneously for different contexts of use.

.	Ideal XML	Sketchi XML	GUI Layout ++	Grafi XML
Service Processing & Standardization	No	No	No	No
Set Tasks Relationships	Yes	No	No	No
Modify an AUI	Yes	No	Yes	Yes
Associate Interaction Resources to AIO	No	No	No	No
Generate task models in UsiXML	Yes	No	No	No
Generate AUI models in UsiXML	Yes	No	Yes	Yes
Generate Resource models in UsiXML	No	Yes	No	No

Table 1. Comparison of tools.

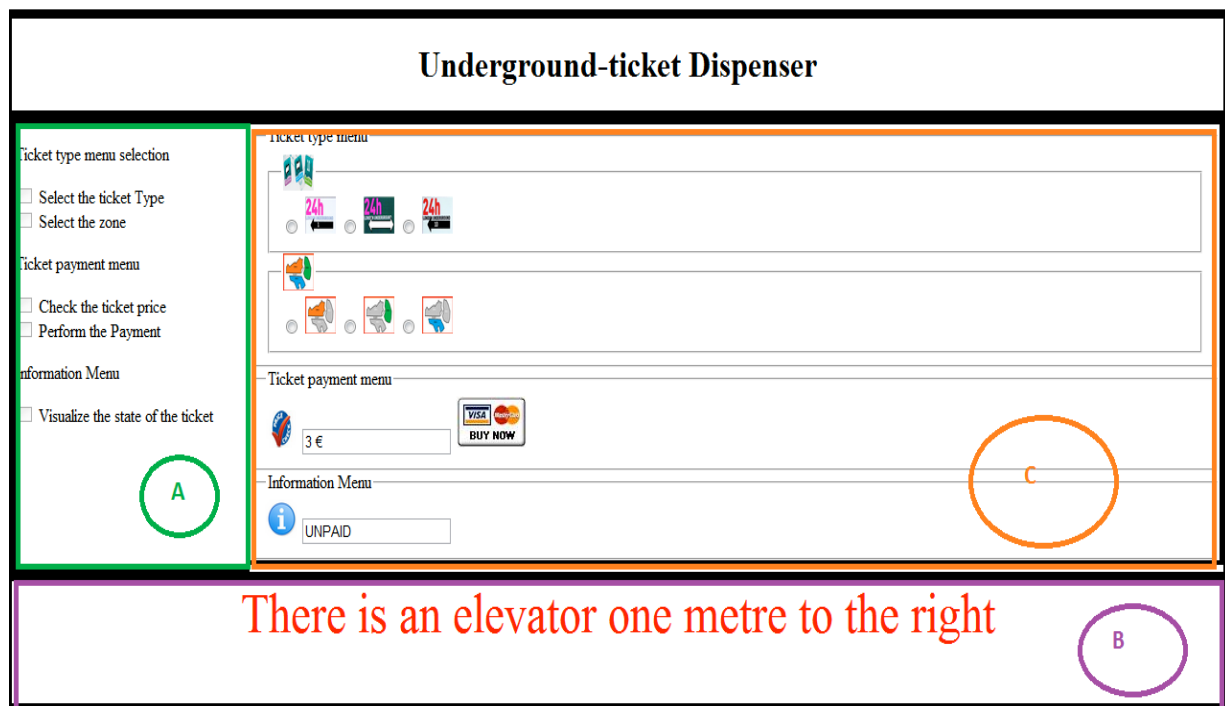


Figure 1. User Interface generated by the adaptive system EGOKI.

These tools have been compared in the table 1, taking into account the tasks required. Some of the required functionalities, such as *Service Processing & Standardization* and *Associate Interaction Resources to AIO*, are not covered by any of these tools.

For this reason, a new tool is required. Nevertheless, when possible, parts of these tools will be reused. For instance, the graphical notation provided by IdealXML for the UsiXML elements task model and the AUI model will be probably used for our tool.

To conclude, although these tools share some features with our approach, our tool will serve a different purpose. SPA4USXML is devoted to provide interaction resources adapted to people with special needs being valid for different types of ubiquitous services.

#### APPLICATION SCENARIO

Imagine a subway station that provides diverse ubiquitous services. When users enter the station with their mobile devices the ubiquitous environment announces the available services. If a user selects one of the services, the middleware downloads a specific user interface to the mobile user device. The user interface must be accessible for each specific user. Consequently, a unique user interface will not be valid for every people. Therefore the user interfaces provided by the ubiquitous environment are adapted to the abilities of each user and to his or her personal device.

The services offered are:

1. Underground ticket dispenser
2. Trains arrival information

3. Elevator
4. Automatic Teller Machine
5. Information kiosk
6. News Service.

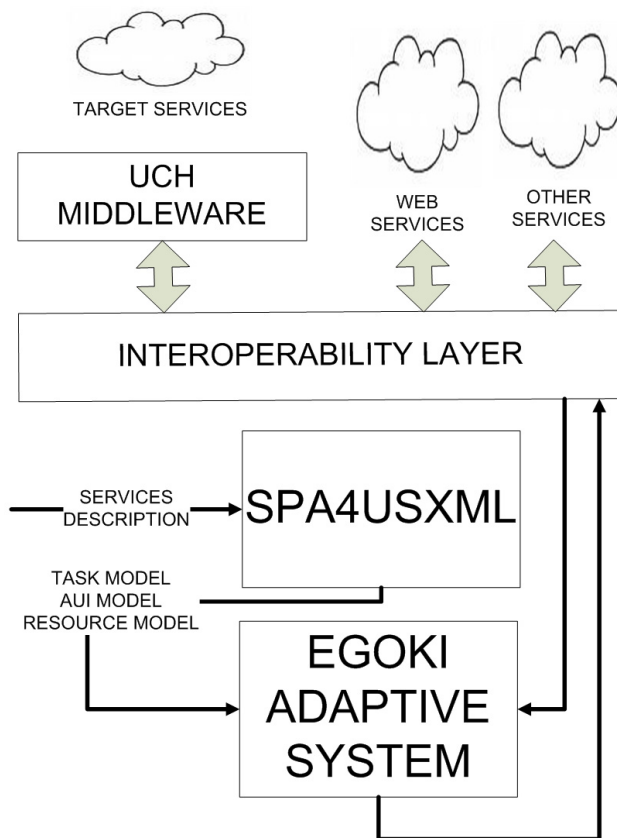
The middleware layer designed to manage the ubiquitous environment provides abstract descriptions of the services to access them. In our case, UCH takes this function.

The EGOKI adaptive system creates the accessible adapted final user interfaces from the abstract user interfaces provided by the designer of the service (who knows its semantics and functionality). Abstract specifications of services are not enough for this purpose because additional information is required, for instance, to assign some semantics to the interaction widgets.

Therefore one of the duties of service designers is the generation of an abstract user interface for each service. For this task they use the tool SPA4USXML that allows the generation of the abstract user interfaces graphically, from the description of the functionality of a ubiquitous service without the need of mastering the inner syntax.

The tool must be very intuitive and has to guide the designer through all steps of the process, making possible to create the AUI in a very short period of time.

The Abstract User Interface (AUI) generated is an input to the EGOKI adaptive system. EGOKI matches the AUI with the user profile and with the available interaction resources in order to build the adapted and accessible final user interface.



**Figure 2. Architecture of the ubiquitous environment.**

Let us consider an elderly user with mild cognitive restrictions. The user wants to buy an underground ticket to travel to a specific station. Figure 1 illustrates the interface that will be built for this user.

Part A guides the user, offering an automatically updated checklist of the steps required to successfully complete each task. This list of steps is user tailored, selecting the most suitable resources for the user.

Part B displays notifications and warnings to guide and warn users. This information is obtained from rules related to each service [16]. In order to create the messages, the user's features, the task he or she is performing and the context characteristics are taken into account. Since these parameters can frequently change, the system periodically updates them and sends these updates to the user interface if necessary.

The Final User Interface for the service selected by the user is rendered in part C. This interface offers all the service functionality allowing the user to interact with the service without barriers. The UCH middleware layer updates the state of the variables when necessary.

Therefore, the generated interface is suitable for the specific user, since the most appropriate interaction resources to his or her capabilities have been selected. In addition, EGOKI performs further adaptations [1].

## THE TOOL SPA4USXML

SPA4USXML is an ongoing research aimed at developing a high-level tool to allow designers of ubiquitous services generating from the description of the services, abstract user interfaces instances, task instances and multimedia resource instances compliant with UsiXML. The tool will have a easy to use graphical user interface, to avoid the need of writing XML code. The instances generated allow the adaptive system EGOKI to provide these services in an accessible way to people with special needs.

This tool will allow designers to offer their ubiquitous services with higher quality, providing greater amount of information to cover a larger number of diverse user interfaces, for a given service and offer interfaces compliant with a wider range of special needs.

## Architecture Overview

Figure 2 illustrates the architecture where SPA4USXML will be integrated. The context is a ubiquitous environment where various services are offered locally. These services are controlled through a middleware layer devoted to handle access, communication and control to the diverse machines available on the local environment. As it has been previously mentioned, the middleware used is the UCH implementation [27] of the URC specification [28]. The UCH interoperability layer identifies the type of service requested by the user, the structures needed to invoke it, and procures user-transparent access to the service. However, the explanation of this layer goes beyond the objectives of this article.

Besides the services offered by different machines available in the environment, this architecture allows access to other services accessible through Internet, based on the Web services specification [31] or not (such as those offered by Google [7]). Remote services have been included in the architecture because they can provide some added value to the user. In addition, the combination and integration of various services will allow improved functionality, as described in the section "Future Capabilities".

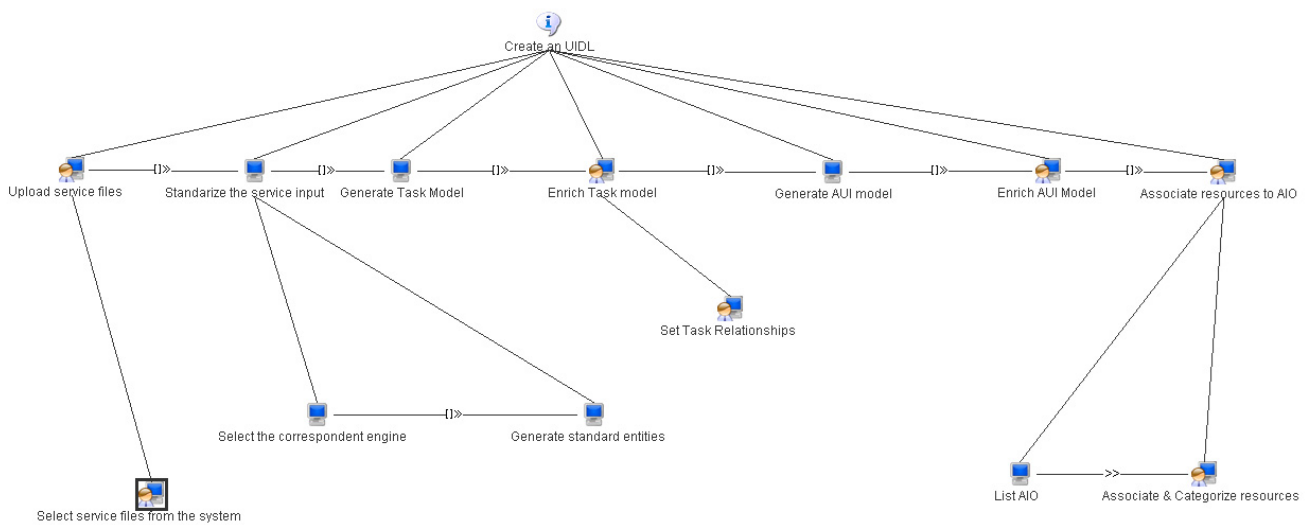
To summarize, SPA4USXML must be used to create the abstract user interface for each service to be integrated in the ubiquitous environment, because the output of the tool is the required input for the EGOKI adaptive system to be able to automatically generate a user interface accessible and adapted to the needs of the user.

This tool takes as input the abstract description of the services functionality provided by the ubiquitous environment and generates enriched instances of task models, abstract user interfaces and multimedia resources models.

## Non Functional Requirements

In addition to the functional requirements, the tool must meet some other requirements:





**Figure 3. Task model of the tool SPA4USXML.**

- *Graphical user interface*: the objective is to provide a graphical tool with textual labels to ease the work of the designer of the service. In addition, the tool must support the edition of the UsiXML code in textual mode.
- *Easy to use*: in order to implement the tool, usability principles must be considered [20]. The tool must be intuitive and the user should be able to use it correctly and immediately.
- *Accessible*: everybody must be able to use the tool to provide enriched service descriptions. The tool must conform to the usual accessibility criteria, such as the Authoring Tool Accessibility Guidelines (ATAG) [2].

#### Basic Functionalities

The SPA4USXML will provide the basic functionality to get abstract user interface descriptions. In addition, abstract containers and abstract individual components are associated with different interaction resources in order to allow EGOKI the construction of multiple CUI following the approach proposed by Limbourg *et al.* (2004) [11].

This functionality is achieved through a process divided into several steps. In Figure 3 you can see the distribution of tasks designed to define the different phases in this process and the dependencies between tasks. This tool will be both graphically and textual, to facilitate their use and to do the learning curve for the designer as minimum as possible.

The first task for the system designer is to upload the files with the description of the services' functionality. There will be a file for each type of ubiquitous service handled by the system. Once uploaded to the system, the description of the functionality of the services must be standardized to be independent of the type of service. To do it, the type of service must be identified and the correspondent transformation entities must be selected in order to transform the functionality descriptions to a common format.

The next step of the process is to get the functionality provided by the service and to represent it as task, from the files with the functionality uploaded to the platform. The representation of tasks will be carried out using the syntax proposed by the UsiXML task model. Once the task representation is built, the designer will have the possibility of editing the temporal relationships among them. The tool IdealXML [19] can be considered as an example for editing task dependencies. In this way the task model is provided with further information facilitating the generation of the different abstract models.

After that, the structure of an abstract user interface model is showed to the designer, allowing him or her to edit and enrich the structure. On this model the user can modify the relationships between different "abstract individual components" (AIC) and "Abstract Containers" (AC). This allows the service designer to decide, based on his or her experience as service provider and creator, what is the best way to interact with the different services.

Finally, the system displays all the available interaction elements in the abstract user interface through a list where the designer will be able to associate different types of interaction resources to each interaction element. The interaction resources considered, among others, are texts, icons, icons in high contrast, audio, transcripts of audio, video, video with captions, 3D elements, etc.

After the association of resources to different abstract interaction objects, EGOKI will be able to concrete different types of interfaces according to the users' needs and the system platform.

#### Future Functionalities

Besides the basic functionality that is planned to be designed and developed in the short term, further iterations with new functionalities are considered:

*Services Combination.* It is intended to offer the designer the possibility of combining services. The aim is to improve the quality of services. For instance, to combine a service offering information about points of interest in outdoor sports installations with Google Maps [8]. In this way it is possible to identify where are located these points of interest.

*Layout.* Currently the final user interfaces generated by the system are based on the styles offered by EGOKI. The future functionality of this tool will allow users to associate different styles to both, abstract containers and abstract individual components. These styles are used for the system EGOKI to use when creating concrete interfaces, allowing them to use different styles.

## DISCUSSION

We believe that the development of the tool proposed is feasible and, thanks to language UsiXML will be integrated with EGOKI properly. However, we would like to discuss some points:

- When automating the analysis of different types of services and the subsequent transformation to generate basic instances of tasks and abstract user interfaces, is there any ambiguous element that requires additional information to be automated? Is it necessary to provide the designer with syntax knowledge to extend the information of the services?
- Would it be feasible to create a general repository of multimedia resources, accessible through the Internet and tagged semantically, to provide resources to the designer of the service?
- Could the system EGOKI use the abstract user interfaces developed by other tools such as IDEALXML or SketchiXML? In what contexts or scenarios would be applicable?

## ACKNOWLEDGMENTS

This research work has been partly funded by the Department of Education, Universities and Research of the Basque Government. In addition, Raúl Miñón enjoys a PhD scholarship from the Research Staff Training Program of the Department of Education, Universities and Research of the Basque Government. We thank the INREDIS project, which has been the foundation for this work.

## REFERENCES

1. Abascal, J., Aizpurua, A., Cearreta, I., Gamecho, B., Garay-Vitoria, N. and Miñón, R. Automatically Generating Tailored Accessible User Interfaces for Ubiquitous Services. In *Proc. of the 13th Int. ACM SIGACCESS Conf. on Computers and Accessibility ASSETS 2011*. ACM Press, New York (2011).
2. Authoring Tool Accessibility Guidelines (ATAG). Available at, <http://www.w3.org/WAI/intro/atag.php>
3. Brusilovsky P., Kobsa A., and Nejdl W. (Eds.). 2007. *The Adaptive Web: Methods and Strategies of Web Personalization*. Springer-Verlag, Berlin, Heidelberg.
4. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L. and Vanderdonckt, J. A unifying reference framework for multi-target user interfaces. *Interacting with Computers* 15,3 (2000), pp. 289–308.
5. Coyette, A., Faulkner, S., Kolp, M., Limbourg, Q. and Vanderdonckt, J. SketchiXML: Towards a Multi-Agent Design Tool for Sketching User Interfaces Based on UsiXML, In *Proc. of 3<sup>rd</sup> Int. Workshop on Task Models and Diagrams for user interface design TAMODIA'2004 (Prague, November 15-16, 2004)*. Ph. Palanque, P. Slavik, M. Winckler (Eds.). Springer-Verlag, Berlin (2004), pp. 75-82.
6. Gajos, K.Z., Weld, D.S. and Wobbrock, J.O. Automatically generating personalized user interfaces with Supple. *Journal of Artificial Intelligence* 174, 12-13 (2010), pp. 910–950.
7. Google Code. Available at <http://code.google.com/intl/es-ES/more/>
8. Google Maps API Family. Available at <http://code.google.com/intl/es/apis/maps/>
9. Guerrero-García, J., González-Calleros, J.M., Vanderdonckt, J., and Muñoz-Arteaga, J. A Theoretical Survey of User Interface Description Languages: Preliminary Results. In *Proc. of Joint 4<sup>th</sup> Latin American Conference on Human-Computer Interaction-7th Latin American Web Congress LA-Web/CLIHIC'2009* (Merida, November 9-11, 2009). E. Chavez, E. Furtado, A. Moran (Eds.). IEEE Computer Society Press, Los Alamitos (2009), pp. 36-43.
10. Hervás, R. and Bravo, J. Towards the ubiquitous visualization: Adaptive user-interfaces based on the Semantic Web. *Interacting with Computers* 23, 1 (2011), pp. 40–56.
11. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., and Víctor López Jaquero. UsiXML: a Language Supporting Multi-Path Development of User Interfaces. In *Proc. of 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems EHCI-DSVIS'2004* (Hamburg, July 11-13, 2004). Lecture Notes in Computer Science, vol. 3425. Springer, Berlin (2004), pp. 200-220.
12. LiquidApps application. Available at <http://liquidapps.harmonia.com/features/>
13. Lunn, D., Bechhofer S., and Harper, S. The SADIE transcoding platform. In *Proc. of the 2008 Int. Cross-disciplinary Conf. on Web Accessibility W4A'08* (Beijing, China), 128-129.

14. Meskens, J., Vermeulen, J., Luyten, K. and Coninx, K. Gummy for multi-platform user interface designs: shape me, multiply me, fix me, use me. In *Proc. of the Working Conf. on Advanced Visual Interfaces, AVI 2008* (Napoli, Italy, May 28-30, 2008), 233-240.
15. Michotte, B., and Vanderdonckt, J. GrafiXML, A Multi-Target User Interface Builder based on UsiXML. In *Proc. of 4<sup>th</sup> Int. Conf. on Autonomic and Autonomous Systems ICAS'2008* (Gosier, 16-21 March 2008). IEEE Computer Society Press, Los Alamitos (2008), pp. 15-22.
16. Miñon, R., and Abascal, J. Supportive adaptive user interfaces inside and outside the home. In *Procs. of the 2nd Int. Workshop on User Modeling and Adaptation for Daily Routines (UMADR): Providing Assistance to People with Special and Specific Needs* (Girona, Spain, July 11, 2011). Available at <http://hada.ii.uam.es/umadr2011/>.
17. Mori, G., Paternò, F., Santoro, C. 2002. CTTE: Support for developing and analyzing task models for interactive system design. *IEEE Trans. on Soft. Eng.* 28, 797-813.
18. Montero, F. and López-Jaquero, V. Guilayout++: Supporting Prototype Creation and Quality Evaluation for Abstract User Interface Generation. In *Proc. of the 1<sup>st</sup> Workshop on User Interface eXtensible Markup Language (UsiXML'2010)* (Berlin, June 20, 2010). Thalès, Paris (2010), pp 39-44.
19. Montero, F., Lozano, M.D. and González, P. IDEAL-XML: an Experience-Based Environment for User Interface Design and pattern manipulation, Technical report DIAB-05-01-4, University of Castilla-La Mancha, Albacete, 24 January 2005.
20. Nielsen J. (Ed). *Designing Web Usability: The Practice of Simplicity*. New Riders Publishing, Indianapolis (1999).
21. Nylander, S. *The Ubiquitous Interactor - Mobile Services with Multiple User Interfaces*. Licentiate thesis, Uppsala University. 2004.
22. OASIS User Interface Markup Language (UIML). Available at [http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=uiml](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=uiml)
23. Paternò, F. *Model-Based Design and Evaluation of Interactive Applications*. Springer-Verlag, London (1999).
24. Philip, A. Bernstein. Middleware: a model for distributed system services. *Communications of the ACM* 39, 2 (1996)
25. Shaer, O., Green, M., Jacob, R.J.K, and Luyten, K. User Interface Description Languages for Next Generation User Interfaces. In *Extended Abstracts of CHI'08*. ACM Press, New York (2008), pp. 3949-3952.
26. Stephanidis, C., Paramythis, A., Sfyraakis, M., and Savidis, A. A study in unified user interface development: the AVANTI web browser. In *User Interfaces for All: Concepts, Methods and Tools*. Stephanidis, C. (Ed.). Lawrence Erlbaum Associates, Mahwah (2001), pp. 525-568.
27. Universal Control Hub 1.0 (Draft). Available at <http://myurc.org/TR/uch>
28. Universal Remote Console Standard (ISO 2008). Available at <http://myurc.org/whitepaper.php>
29. UsiXML, User Interface eXtensible Markup Language reference manual. Available at <http://www.usixml.org/index.php?mod=pages&id=5>.
30. XML, Extensible Markup Language Working Groups. Available at <http://www.w3.org/XML/>.
31. Web Services Activity. Available at <http://www.w3.org/2002/ws/>
32. Vermeulen, J., Vandriessche, Y., Clerckx, T., Luyten, K. and Coninx, K. Service-Interaction Descriptions: Augmenting Services with User Interface Models. In *Proc. of Joint Working Conferences on Engineering Interactive Systems EIS'2007* (Salamanca, March 22-24, 2007). Lecture Notes in Computer Science, vol. 4940. Springer, Berlin (2008), pp. 447-464.

# A Theoretical Survey of User Interface Description Languages: Complementary Results

**Josefina Guerrero-García**

**Juan Manuel González-Calleros**

Facultad de ciencias de la Computación  
Benemérita Universidad Autónoma de Puebla  
Ciudad Universitaria, C.P. 72400  
Puebla, México  
{jguerrero, juan. gonzalez}@cs.buap.mx  
Research and Development Unit, Estrategia360  
Puebla, México

**Jean Vanderdonckt**

Université catholique de Louvain  
Place des Doyens, 1 – B-1348  
Louvain-la-Neuve, Belgium  
jean.vanderdonckt@uclouvain.be

**Jaime Muñoz-Arteaga**

Sistemas de Información  
Universidad Autónoma de Aguascalientes  
Av. Universidad No. 940, Col. Bosques, 20100  
Aguascalientes, Aguascalientes (México)  
jmunozar@correo.uaa.mx

## ABSTRACT

This paper presents new work on our survey of user interface description language (UIDL). There is limited change or improvements in previously UIDLs reviewed but there some new approaches that are good to use. Continuing our in-depth analysis, we consider the salient features that make these languages different from each other and identifying when and where they are appropriate for a specific purpose. The review is conducted based on a systematic analysis grid and some user interfaces implemented with these languages.

## Author Keywords

User interfaces, User Interface Description Language, Extensible Markup Language, User Interface extensible Markup Language.

## General Terms

Design, Human Factors, Theory

## Categories and Subject Descriptors

I.6.3 [Computing Methodologies]: Simulation and Modeling—*Applications*; D.2.2 [Software]: Software Engineering—*Design Tools and Techniques*.

## INTRODUCTION

The ITE2 UsiXML project aims at producing a UIDL that covers the “μ7” concept defined as multi-device, multi-platform, multi-user, multi-linguality / culturality, multi-organisation, multi-context, multi-modality. In order to select and collect the most valuable aspects of existing UIDLs a deep analysis has been conducted during the last year. In the past [10], we gathered and analyzed as much literature as possible on each UIDL. Then, depending on available tools, we systematically developed a multi-platform or multi-context UI for a simple dictionary so as to identify the capabilities of the UIDL and the ability of this UIDL to be supported by editing, critiquing, analysis tools, and, of course, tools for producing executable UIs, both by compilation/execution and by interpretation.

In this paper we extend this review by including recent developments in this field in our previous survey so as to keep it up to date. The remainder of this paper is structured as follows: Section 2 describes recent progress on UIDL. Section 3 defines the comparison criteria used in the comparison analysis and provides the final analysis grid. Section 4 presents the conclusion.

## RECENT ADVANCES ON USER INTERFACE DESCRIPTION LANGUAGES

In this section we looked at our previous review [4,10] to identify if there is any advance on the UIDLs. Those languages that did not experience any progress that we are aware of are: the eXtensible Interaction Scenario Language (XISL) [13]; The eXtensible mark-up language for Multi-Modal interaction with Virtual Reality worlds (XMMVR) [19]; The Device Independent Authoring Language (DIAL) [31]; The delivery context [32]; The Extensible MultiModal Annotation Markup Language (EMMA) [34]; InkML [30]; VoiceXML [28]; XForms [33]; Dialog and Interface Specification Language (DISL) [24]; the Generalized Interface Markup Language (GIML) [15]; Interface Specification Meta-Language (ISML) [4]; Renderer-Independent Markup Language (RIML) [6]; Software Engineering for Embedded Systems using a Component-Oriented Approach (SeesoaXML) [17]; the Simple Unified Natural Markup Language (SunML) [22]; User Interface Markup Language (UIML) [1]; The eXtensible Interface Markup Language (XIML) [7,8]; Web Service eXperience Language (WSXL) [2,12]; The eXtensible user-Interface Markup Language (XICL) [9]. Yet they are still of the interest for our review as the knowledge expressed there it is useful for the future of UIDLs specification.

*TeresaXML* [20] is a UIDL for producing multiple final UIs for multiple computing platforms at design time. The language evolved in Time and turned into *MariaXML* 20. This language support dynamic behaviors, events, rich internet applications, multi-target user interfaces, in particular those based on web services. In this way, it is possible to have a

UI specified in MariaXML attached to a web service. MariaXML relies on the multi-layer levels of the Cameleon Reference Framework [4]. Thus it describes concepts like: Data objects manipulated by interactors, Events for abstract and concrete UI, Dialogue Model with conditions and CTT operators for event handlers, scripts like Ajax; and dynamic set of user interface elements. All these models are supported by the software Mariae (Maria Environment) using task modeling (using ConcurTaskTrees notation) and UI modeling (MARIA language).

UserML [12] is a UIDL to model users. This approach models are semantically using an OWL ontology, called GUMO, and it is syntactically described using the UserML language. The main concept is the *situational statement* representing: user model entries, context information or low-level sensor data [12]. Figure 1 shows the metamodel including the five elements that compose a *situational statement*.

The *Mainpart* is an extension of the Resource Definition Framework (RDF) to represent information about the user using the interactive system, including aspects such as *range* of values and *auxiliary* to point to other ontologies. The *Privacy* models keeps information about the permissions to share the statement with tusers. The *Explanation* model helps to clarify conflicts or problems that the user might confront; it defines the source of the problem, the creator, collect the evidence, the degree of confidence, and the method used to identify the problem. The *Administration* model describes the role relation of the statement with the organization. The *Situation* model represents temporal (start, end, durability) and spatial (location and position) aspects of the user action. Each user model is used as information to model context-aware systems in ubiquitous environments targeting mobile, speech, virtual and graphical UIs.

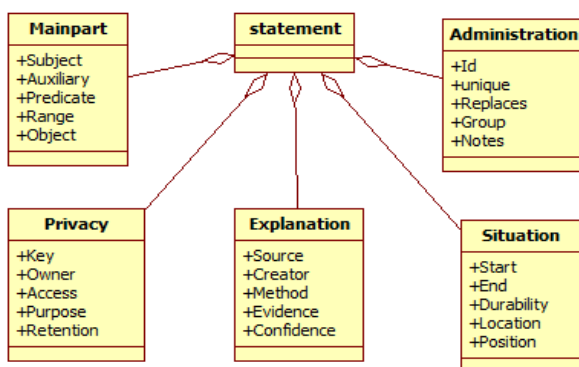


Figure 1. User Meta-Model in UserML.

XooML [28] is language that describes aspects for supporting the development of software tools to collaborate and share documents. There is a XML Schema that is used by

three different tools: Plantz (providing information to handle documents and other forms of information exchange), QuickCapture (capturing elements that can be related to information exchange), FreeMindX (mind-mapping). The XooML structure is minimal as it is composed of fragments that served as indivisible grouping of information, establishing a context for the comprehension of an selection amount its constituent associations. Although XooML is not a genuine UIDL it comprises aspects related to UIs that are of relevance when considering the definition of software tools supporting UIDL.

*User Interface eXtensible Markup Language* (UsiXML) [27] is structured according to different levels of abstraction defined by the Cameleon reference framework 3. The framework represents a reference for classifying UIs supporting a target platform and a context of use, and enables to structure the development life cycle into four levels of abstraction: task and concepts, abstract UI (AUI), concrete UI (CUI) and final UI (FUI). Thus, the Task and Concepts level is computational-independent, the AUI level is modality-independent (In the cockpit it can be several physical, Vocal, GUI, Tactile) and the CUI level is toolkit-independent. UsiXML relies on a transformational approach that progressively moves among levels to the FUI. The transformational methodology of UsiXML allows the modification of the development sub-steps, thus ensuring various alternatives for the existing sub-steps to be explored and/or expanded with new sub-steps. UsiXML has a unique underlying abstract formalism represented under the form of a graph-based syntax.

#### USER INTERFACE DESCRIPTION LANGUAGES COMPARISON

We kept the same protocol for the comparison as we did before, in accordance to a previous study conducted in [25], this work slightly updates and extends our previous survey [10] considering the latest results along the following dimensions:

- *Specificity* indicates if the UIDL could be used in one or multi platforms or devices.
- *Publicly available*: depending on the availability of the language deep analysis can be done. This category was used to discard many languages that lack on documentation or that is confidential. The possible values are: 0 = no information available; 1 = not available; 2 = poorly available or very limited information was available so no way to get the details to generate their meta-models; 3 = moderately available, an understanding of the language was possible so as to get the meta-models; 4 = completely available in the format of the language and 5 = completely available including meta-models.
- *Type criterion* informs whether the UIDL is a research or industry work.
- *Weight of the organization* behind denotes the organization to which the UIDL belongs. Efforts from Universi-

ties are significant, particularly, those where more than one university has adopted the use of the UIDL. Those UIDL coming from the industry have more impact and this is reflected in its level of usage.

- *Level of usage*: depending on the usage of the language, which was computed based on the organization weight and the research community supporting the use of the tool, we create the following categories: 0 = unknown, no information was available, 1 = one person, research of an individual, 2 = small research group, 3 = one organization or research community, 4 = two or more organizations or research communities and 5 = wide usage, globally adopted and used.

Due to its number of concepts, UsiXML has been intentionally removed from Table 2 and it is used to illustrate the comparison protocol (Figure 2). On the left a series of developments steps compliant with the Cameleon reference framework 3, to the right the supported concepts and the transformations applied to UsiXML. Details on this comparison can be found in the model based incubator group [35] where this work has been reported. Table 2 compares the properties of the different UIDLs according the eight criteria:

- *Component models*: this criterion gives the aspects of the UI that can be specified in the description of the UIs. The *task model* is a description of the task to be accomplished by the user; the *domain model* is a description of the objects the user manipulates, accesses, or visualizes through the UIs; the *presentation model* contains the static representation of the UI, and the *dialog model* holds the conversational aspect of the UI.
- *Methodology*: different approaches to specify and model UIs exist: 1) Specification of a UI description for each of the different contexts of use. As a starting point, point, a UI specification for the context of use considered as representative of most case, the one valid for the the context of use considered as the least constrained or finally the one valid for the context of use considered as as the most comprehensive is specified. From this starting UI specification, corrective or factoring out decorations [25], (e.g., to add, remove, or modify any UI description) are applied so that UI specifications can be derived for the different contexts of use. 2) Specification of a generic (or abstract) UI description valid for all all the different contexts of use. This generic UI description is then refined to meet the requirements of the different contexts of use.
- *Tools*: some of the languages are supported by a tool that helps designer and renders the specification to a specific language and/or platform.
- *Supported languages*: specify the programming languages to which the XML-based language can be translated.
- *Supported platforms*: specify the computing platform on which the language can be rendered by execution, interpretation or both.

- *Coverage of concepts*: depending on the level of abstraction, each UIDL may introduce some specific vs. generic concepts (e.g., a given presentation model vs. any model, each custom-defined), their properties (e.g., to what extent can a concrete presentation be specified), and their relations.

## CONCLUSION

Eight years from now, a first review of UIDLs was conducted [25]. That work was reviewed and updated two years ago 10, accordingly to the progress of those UIDLs. While some works have continued their research in this field, there were works with not reported update since then, at east nothing that we were aware of, based on their websites and research papers. To extend our previous work, new UIDLs and concepts were included in this update. The goal of this work remains to be a guide for authors to decide what UIDL to use for their projects. There is currently such a large number of UIDLs available that choosing among them can be time consuming and difficult to do, this comparison can assist UI designers in choosing a language suited to their purposes. The future work comprises the validation of the criteria with groups of researchers to determine the usefulness of the criteria to check if the purpose of the comparison is achieved.

## ACKNOWLEDGMENTS

We gratefully acknowledge the support of the Repatriacion CONACYT program ([www.conacyt.mx](http://www.conacyt.mx)) supported by the Mexican government, the ITEA2 Call 3 UsiXML project under reference 20080026 and its support by Région Wallonne DGO6, and the PROMEP net Project under Contract UAA-CA-48.

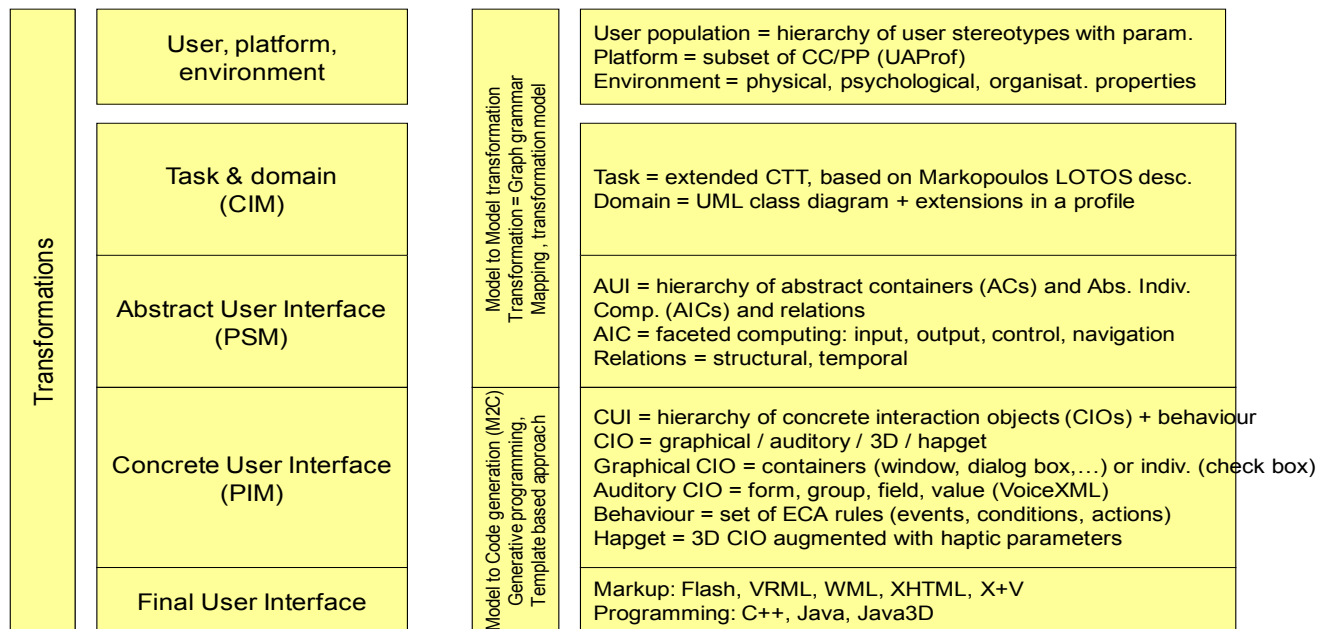
## REFERENCES

1. Abrams, M., Phanouriou, C., Batongbacal, A.L., Williams, S. & Shuster, J. (1999), UIML: An Appliance-Independent XML User Interface Language. In *Proc. of 8<sup>th</sup> Int. World-Wide Web Conference WWW'8* (Toronto, May 11-14, 1999). Elsevier Science Publishers, Amsterdam, 1999
2. Arsanjani, A., Chamberlain, D. and et al. (2002), (WSXL) web service experience language version, 2002. Retrieved from: <http://www-106.ibm.com/developerworks/library/ws-wsxl2/>.
3. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J. A Unifying Reference Framework for Multi-Target User Interfaces. *Interacting with Computers* 15, 3 (June 2003), pp. 289–308.
4. Cantera Fonseca, J.M. (Ed.). Model-based User Interface XG Final Report, W3C Incubator Group Report, W3C, 4 May 2010. Accessible at: <http://www.w3.org/2005/Incubator/model-based-ui/XGR-mbui-20100504/>



5. Crowle, S., and Hole, L. ISML: An Interface Specification Meta-Language. In *Proc. of DSV-IS 2003* (Funchal, June 11-13, 2003). Lecture Notes in Computer Science, vol. 2844. Springer, Berlin (2003).
6. Demler, G., Wasmund, M., Grassel, G., Spriestersbach, A., and Ziegert, T. Flexible pagination and layouting for device independent authoring. In *Proc. of WWW2003 Emerging Applications for Wireless and Mobile access Workshop*.
7. Eisenstein, J., Vanderdonckt, J., and Puerta A. Adapting to Mobile Contexts with User-Interface Modeling. In *Proc. of 3<sup>rd</sup> IEEE Workshop on Mobile Computing Systems and Applications WMCSA'2000* (Monterey, 7-8 December 2000). IEEE Press, Los Alamitos (2000), pp. 83-92.
8. Eisenstein J., Vanderdonckt J., and Puerta A. Model-Based User-Interface Development Techniques for Mobile Computing. In *Proc. of 5<sup>th</sup> ACM Int. Conf. on Intelligent User Interfaces IUI'2001* (Santa Fe, 14-17 January 2001), Lester, J. (Ed.). ACM Press, New York (2001), pp. 69-76.
9. Gomes de Sousa, L., and Leite, J.C. XICL: a language for the user's interfaces development and its components. In *Proc. of the Latin American conference on Human-computer interaction* (Rio de Janeiro, Brazil, August 17 - 20, 2003). ACM Press, New York (2003), pp. 191-200.
10. Guerrero-García, J., González-Calleros, J.M., Vanderdonckt, J., and Muñoz-Arteaga, J. A Theoretical Survey of User Interface Description Languages: Preliminary Results. In *Proc. of Joint 4<sup>th</sup> Latin American Conference on Human-Computer Interaction-7th Latin American Web Congress LA-Web/CLIHIC'2009* (Merida, November 9-11, 2009). E. Chavez, E. Furtado, A. Moran (Eds.). IEEE Computer Society Press, Los Alamitos (2009), pp. 36-43.
11. Helms, J., Schaefer, R., Luyten, K., Vermeulen, J., Abrams, M., Coyette, A., and Vanderdonckt, J. Human-Centered Engineering with the User Interface Markup Language. In *Human-Centered Software Engineering*, A. Seffah, J. Vanderdonckt, M. Desmarais (Eds.), Chapter 7, HCI Series, Springer, London, 2009, pp. 141-173.
12. Heckmann, D. Ubiquitous User Modelling. Akademische Verlagsgesellschaft Aka GmbH, Berlin, ISBN 3-89838-297-4 and ISBN 1-58603-608-4, 2006.
13. IBM (2002), WSXL specification, April 2002, retrieved on January 2<sup>nd</sup> 2009.
14. Katsurada, K., Nakamura, Y., Yamada, H., and Nitta, T., XISL: A Language for Describing Multimodal Interaction Scenarios, *Proceedings of the 5th International Conference on Multimodal Interfaces ICMI'03* (Vancouver, Canada).
15. Kost, S. Dynamically generated multi-modal application interfaces. Ph.D. Thesis, *Technical University of Dresden and Leipzig University of Applied Sciences*, Dresden (2004)
16. Lucent (2000), Sisl: Several Interfaces, Single Logic, Lucent Technologies, Available online: <http://www.bell-labs.com/user/lalita/sisl-external.html>
17. Luyten, K., Abrams, M., Vanderdonckt, J. & Limbourg, Q. Developing User Interfaces with XML: Advances on User Interface Description Languages, *Satellite workshop of Advanced Visual Interfaces 2004*, Gallipoli (2004).
18. Michotte, B., and Vanderdonckt, J. GrafiXML, A Multi-Target User Interface Builder based on UsiXML. In *Proc. of 4<sup>th</sup> International Conference on Autonomic and Autonomous Systems ICAS'2008* (Gosier, 16-21 March 2008). IEEE Computer Society Press, Los Alamitos, 2008.
19. Olmedo, H., Escudero, D., and Cardenoso, V.: A Framework for the Development of Applications Allowing Multimodal Interaction with Virtual Reality Worlds. In *Communications Proceedings 16th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision'2008 WSCG'2008* (Plzen - Bory, Czech Republic, February 4-7), University of West Bohemia Press (2008), pp. 79-86.
20. Paternò F., Santoro C., and Spano L.D. MARIA: A Universal Language for Service-Oriented Applications in Ubiquitous Environments. *ACM Transactions on Computer-Human Interaction* 16, 4 (November 2009), pp.19:1-19:30.
21. Paternò, F., and Santoro, C. A Unified Method for Designing Interactive Systems Adaptable to Mobile and Stationary Platforms. *Interacting with Computers* 15, (2003), pp. 349-366.
22. Picard, E., Fierstone, J., Pinna-Dery, A-M., and M. Riveill. Atelier de composition d'IHM et évaluation du modèle de composants. *Livrable I3, RNTL ASPECT*, Laboratoire I3S, mai 2009.
23. Puerta A.R. The Mecano Project: Comprehensive and Integrated Support for Model-Based Interface Development. In *Proc. of 2<sup>nd</sup> Int. Workshop on Computer-Aided Design of User Interfaces CADUI'96* (Namur, 5-7 June 1996). J. Vanderdonckt (Ed.). Presses Universitaires de Namur, Namur (1996), pp. 19-35.
24. Schaefer, R., Steffen, B., and Wolfgang, M. Task Models and Diagrams for User Interface Design. In *Proc. of 5<sup>th</sup> Int. Workshop TAMODIA'2006* (Hasselt, Belgium, October 2006). Lecture Notes in Computer Science, vol. 4385, Springer-Verlag, Berlin (2006), pp. 39-53.

25. Souchon, N., and Vanderdonckt, J. A Review of XML-Compliant User Interface Description Languages. In *Proc. of 10<sup>th</sup> Int. Conf. on Design, Specification, and Verification of Interactive Systems DSV-IS'2003* (Madeira, 4-6 June 2003). Jorge, J., Nunes, N.J., Falcao e Cunha, J. (Eds.), Lecture Notes in Computer Science, Vol. 2844, Springer-Verlag, Berlin, 2003, pp. 377-391.
26. Thevenin, D. Adaptation En Interaction Homme-Machine : Le Cas de la Plasticité. PhD thesis, Université Joseph Fourier, Grenoble, 21 December 2001.
27. Vanderdonckt, J. A MDA-Compliant Environment for Developing User Interfaces of Information Systems. In *Proc. of 17th Conf. on Advanced Information Systems Engineering CAiSE'05* (Porto, 13-17 June, 2005). O. Pastor & J. Falcão e Cunha (Eds.). Lecture Notes in Computer Science, vol. 3520. Springer-Verlag, Berlin (2005), pp. 16-31.
28. Jones, W. XooML: XML in support of many tools working on a single organization of personal information. In *Proc. of the iConference'2011* (Seattle, February 8-11, 2011). ACM Press, New York (2011), pp. 478-488.
29. W3C, Voice Extensible Markup Language (VoiceXML) Version 2.0, W3C recommendation, 16 March 2004, *W3C Consortium*. Available online: <http://www.w3.org/TR/voicexml20>.
30. W3C, W3C InkML: Digital Ink Markup Language, W3C recommendation, 24 October 2006, *W3C consortium*. Available online: <http://www.w3.org/2002/mmi/ink>
31. W3C, Dial: Device Independent Authoring Language, W3C Working Draft, 2007. *W3C consortium*. Available online: <http://www.w3.org/TR/dial/>
32. W3C, Content Selection Primer 1.0, W3C Working Draft, 2007, *W3C consortium*. Available online: <http://www.w3.org/TR/cselelection-primer/>
33. W3C, XForms 1.0 (Third Edition), W3C recommendation, 29 October 2007, *W3C consortium*. Available online: <http://www.w3.org/TR/2007/REC-xforms-20071029/>
34. W3C, EMMA: Extensible MultiModal Annotation markup language, W3C Proposed Recommendation, *W3C consortium*. Available online: <http://www.w3.org/TR/emma>.
35. W3C Model-based User Interfaces Incubator Group [http://www.w3.org/2005/Incubator/model-based-ui/wiki/Task\\_Meta\\_ModelsXML](http://www.w3.org/2005/Incubator/model-based-ui/wiki/Task_Meta_ModelsXML) User Interface Language (XUL) 1.0.



**Figure 2. Comparison protocol exemplified with UsiXML.**

UIL	Specificity	Publicly available	Type	Weight of the organization behind	Level of usage
DISL	Multimodal UIs for mobile devices	2	Research	Paderborn University	3
GIML	Multimodal	3	Research	Technical University of Dresden and Leipzig University of Applied Sciences	2
ISML	GUI, multiplatform, multidevice	2	Research	Bournemouth University	1
RIML	Mobile devices	0	Industry	Industry: SAP Research, IBM Germany, and Nokia Research Center along with CURE, UbiCall, and Fujitsu Invia	3
SeescoaXML	Multiplatform, multidevice, dynamic generation UI	2	Research	Expertise Centre for Digital Media Limburgs Universitair Centrum	3
SunML	Multiplatform	4	Research	Rainbow team, Nice University	3
UIML	Multiplatform	4	Industry	Harmonia, Virginia Tech Corporate Research (OASIS)	3
WSXL	multiplatform, multidevice	4	Industry	IBM	3
XICL	Multiplatform	3	Research	Federal University of Rio Grande do Norte	3
XIML	multiplatform, multidevice	4	Research	Redwhale Software	3
XWT	multiplatform, multidevice	5	Industry	Google Inc.	5
UserML	multiplatform, multidevice, multimodal	4	Research	University of Saarlandes	3
TeresaXML	Multiplatform, multidevice,	4	Research	HCI Group of ISTI-C.N.R.	3
XooMI	None	4	Research	University of Washington	2
UsiXML	Multiplatform	5	Research	UsiXML Consortium	4

**Table 1. General features of UIDLs.**

UIL	Models	Methodology	Tools	Supported languages	Supported platforms	Concepts
DISL	Presentation, dialog and control	Specification of a generic, platform-independent multimodal UI	Rendering engine	VoiceXML, Java MIDP, Java Swing, Visual C++	Mobile and Limited Devices	Head element, interface classes (structure, style, behavior), state, generic widgets
GIML	Presentation, dialog, and domain	Specification of a generic interface description.	GITK (Generalized Interface Toolkit)	C++, Java, Perl	Not specified	Interface, dialog, widget, objects
ISML	Presentation, task, dialog, domain	Specification of a generic UI description	Under construction	Java, Microsoft foundation class, Java swing classes	Desktop PC, 3D Screen	Mappings and constrains, action events, meta-objects, display parts, controller parts, interaction definition
RIML	There is no information	Specification of a generic UI description	There is no information	XHTML, XFORMS, XEvents, WML	Smart Phone, PDA, Mobile, Desktop PC	Dialog, Adaptation, layout, element
See-scoaXML	Task, Presentation, dialog	Specification of a generic UI description	CCOM (BetaVersion 1.0 2002) PacoSuite MSC Editor	Java AWT, Swing, HTML, java.microedition, applet, VoxML, WML Juggler	Mobile, Desktop PC, Palm III	Component, port, connector, contract, participant, blueprint, instance, scenario, platform, user, device
SunML	Presentation, dialog, domain	Specification of a generic UI description	SunML Compiler	Java Swing, voiceXML, HTML, UIML,	Desktop PC, PDA	Element, list, link, dialog, interface, generic events, synchronization
UIML	Presentation, dialog, domain	Specification of a generic UI description	UIML.net, VoiceXML renderer, WML renderer, VB2UMIL	HTML, Java, C++, VoiceXML, QT, CORBA, and WML	Desktop PC, Handheld Device, TV, Mobile	interconnection of the user interface to business logic, services
WSXL	Presentation, dialog, domain	Specification of a generic UI description	WSXL SDK	HTML	PC, Mobile	CUI=XForms, WSDL, Mapping=XLang Workflow=WSFL, Logic=XML event
XICL	Presentation, dialog,	Specification of a generic UI description	XICL STUDIO	HTML, ECMAScript, CSS e DOM.	Desktop PC	Component, structure, script, events, properties, interface
XIML	Presentation, task, dialog, domain	Specification of a generic UI description	XIML Schema	HTML, java swing, WLM	Mobile, Desktop PC, PDA	Mappings, models, sub models, elements, attributes and relations between the elements
XWT	Presentation, context	Specification of a generic UI description	Google Web Toolkit	Java, JQuery	PC, Mobile	<b>Context= language; location</b> <b>CUI = presentation (Java), layout, Mapping = Widget morphing; data binding</b> <b>Dialog = Java + JQuery</b>
UserML	Context	Specification of a generic UI description	UbisWorld, UbisOntology Editor	Java + XForms	PC, Mobile, PDA	<b>Context = User (MainPart, Privacy, Explanation, Situation, Administration), Location;</b> <b>CUI = Mobile, PC, PDA, XForms</b> <b>Mapping (Match, filter, control)</b>
TeresaXML	Presentation, task, dialog,	Specification of a generic UI description	CTTE Tool for task Models Teresa, Mariae	Markup: Digital TV, VoiceXML, XHTML/SVG, X+V	DigitalTV, Mobile, Touch-based Smartphone,	<b>AUI = Interface, Interactor, Grouping, Connection, dialog expression, composition.</b> <b>AUI Interactors = selection, edit, con-</b>

	domain, context			Programming: C#	Vocal, Mul- timodal X+V	trol, output Mappings, models, platform,
XooMI	Domain	Specification of generic mind- mapping doc- ument ex- change	Plantz, QuickCapture, FreeMindX	None	None	Domain = Documents, fragments Mapping= association Attributes

**Table 2. Properties Comparison of UIDLs.**

# Automated User Interface Evaluation based on a Cognitive Architecture and UsiXML

Jan-Patrick Osterloh, Rene Feil, Andreas Lüdtkke  
OFFIS Institute for Information Technology  
Escherweg 2  
26121 Oldenburg Germany  
+49 441 9722 524  
osterloh, luedtke@offis.de

Juan Manuel Gonzalez-Calleros  
Faculty of Computer Sciences,  
Benemérita Universidad Autónoma de Puebla,  
Ciudad Universitaria, 72592 Puebla, Mexico  
Research and Development Unit, Estrategia 360  
Puebla, Mexico  
juan.gonzalez@cs.buap.mx

## ABSTRACT

In this paper, we will present a method for automated UI evaluation. Based on a formal UI description in UsiXML, the cognitive architecture CASCaS (Cognitive Architecture for Safety Critical Task Simulation) will be used to predict human performance on the UI, in terms of task execution time, workload and possible human errors. In addition, the UsabilityAdviser tool can be used to check the UI description against a set of usability rules. This approach fits well into the human performance and error analysis proposed in the European project HUMAN, where virtual testers (CASCaS) are used to evaluate assistant systems and their HMI. A first step for realizing this approach has been made by implementing a 3D rendering engine for UsiXML.

## Author Keywords

User Interface evaluation, UsiXML, cognitive architecture, CASCaS, UsabilityAdviser

## General Terms

Design, Reliability, Human Factors, Theory

## Categories and Subject Descriptors

I.6.3 [Computing Methodologies]: Simulation and Modeling—*Applications*; D.2.2 [Software]: Software Engineering—*Design Tools and Techniques*.

## INTRODUCTION

Today, human factor analysis of aircraft cockpit systems like autopilot or flight management systems is based on expert judgment and simulator-based tests with human subjects (e.g., test pilots) when first prototypes exist. This is in general a very expensive and time-consuming approach, because a number of subjects have to be hired for the simulation and necessary changes can only be realized with huge effort in the usually late stage of system development. In order to further reduce the cost for Human-Machine Interface (HMI) design of complex assistant systems in transportation, while reducing human error and increasing usability at the same time, the HMI development process has to be improved, by integrating the evaluation of User Interfaces (UIs) into the design process of manufacturers. The European project HUMAN (7<sup>th</sup> Framework Programme) aimed at developing virtual testers, in order to

improve the human error analysis of new assistance systems, including User Interfaces. The virtual testers are based on a fully equipped cognitive architecture CASCaS (Cognitive Architecture for Safety Critical Task Simulation). In HUMAN, this approach has been extensively tested and improved by applying the method to a new assistant system, with focus on the system and the interaction with the system (see [11]), but not so much on the UI itself. In this paper, we will describe how the method could be further extended in future, so that cognitive models can also be used to improve the development of complex UIs. The objective is to provide a tool for automated UI evaluation, in terms of predicting cognitive workload, execution times, human error as well as compliance to HMI guidelines. A similar approach has already been tackled in CogTool [8], a UI prototyping tool, which uses a predictive human performance model to automatically evaluating GUI design.

In the next section, we will discuss CogTool and its application in the industrial process. Then, we will propose another approach for UI evaluation, which should improve some of CogTools shortcomings, and could be integrated in the industrial design process.

## STATE-OF-THE-ART

Currently there are different approaches to evaluation of UI designs. Beside the classical approach of evaluation with test users, automatic evaluation with tools is used. The common major shortcoming of any evaluation tool is that the evaluation logic is hard coded in the evaluation engine [14], for example, two leaders of the web evaluation market, Bobby and A-Prompt only provide the choice between the guidelines of W3C or Section 508, which makes them very inflexible for any modification of the evaluation logic or any introduction of new guidelines. In addition, many of them do not offer much possibilities of controlling the evaluation process like choosing which guideline to evaluate, or the level of evaluation at evaluation time. Not only existing tools cannot accommodate different and multiple bases of guidelines or usability knowledge but also they force the evaluator to evaluate the GUI in a predefined way: it is not possible to focus the evaluation on only some parts of the GUI, for instance by considering only those guidelines that are concerned with the contents. The goal



here is to develop an evaluation tool that addresses the above shortcomings, such as the support of multiple bases of guidelines (accessibility, usability, or both) on-demand (partial or total evaluation), with different levels of details (a presentation for a developers and a presentation for the person who is responsible for attributing the accessibility certification). For this purpose, an evaluation engine should be developed that perform guidelines evaluation or other independently of guidelines and usability knowledge.

Another, newer approach is the evaluation based on cognitive models. CogTool is a general purpose UI prototyping tool, which uses a predictive human performance model to automatically evaluating GUI design [3,8]. In order to perform an analysis, the analyst defines first a prototype of the interface (based on standard set of UI widgets, like buttons, sliders, menus), including possible transitions between different interfaces. Then, a number of tasks are demonstrated on the design, which are recorded and build the basis for the interaction tasks. Then the cognitive architecture ACT-R [1] is used to predict e.g. cognitive workload, and task execution times.

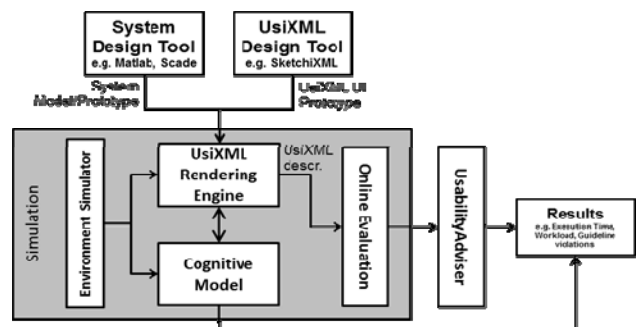
While CogTool allows fast prototyping and evaluation, the UI prototype itself can only be imported and exported as HTML code, and cannot be reused for the final interface. In the transportation domain, model driven development has become standard for development of assistance systems. Using CogTool in an industrial process would require that a given design has to be re-implemented in CogTool, and after the improvements are made within CogTool, these have to be implemented in the final version of the system, as CogTool is currently neither integrated in a UI development tool, nor in a modelling tool used in the industry (like Scade - [www.esterel-technologies.com](http://www.esterel-technologies.com), Matlab - [www.mathworks.com/products/matlab/](http://www.mathworks.com/products/matlab/), or Rhapsody - [www.ibm.com/software/awdtools/rhapsody/](http://www.ibm.com/software/awdtools/rhapsody/)).

In addition, the need to demonstrate the tasks performed in each scenario from start to the end seems for a larger set of scenarios to time consuming. Re-usage of the UI prototype that allows model driven development, as well as re-usage of the tasks that are performed, are main requirements for the proposed method.

## METHOD

In the HUMAN project, a method for system evaluation has been proposed, that integrates virtual testers into the design process of aircraft manufacturers- Main idea is to use the system models (e.g. defined in Matlab) in a simulation together with virtual testers, in order to test the system in an early design phase. The virtual testers are built based on the fully-equipped cognitive architecture CASCaS. For the UI development, we propose to use UsiXML, which stands for USer Interface eXtensible Markup Language. UsiXML is a XML-compliant markup language that describes the UI for multiple contexts of use, i.e. interactive applications with different types of interaction techniques, modalities of use, and computing platforms can be de-

scribed in a way that preserves the design independently from the physical computing platform. Figure 1 shows a possible architecture for automated UI evaluation, with UsiXML, and the cognitive architecture CASCaS.



**Figure 1. Architecture for automated UI evaluation.**

The first step in the proposed method is to model the system functionality in a design tool like Matlab or Scade, and to model the UI using UsiXML. There are multiple tools, based on UsiXML, which support the design process of UIs to create and evaluate rapid prototypes in UsiXML, e.g., SketchiXML [6], with no need for writing XML directly. In the next step, a simulation is used for the evaluation: A newly developed rendering engine for UsiXML is used to display the UsiXML to the virtual tester, or a human user respectively.

On the same time, it controls the interaction between the system model and the UI, i.e. if the virtual tester presses a button this is propagated to the system model and the UI. Each interaction may result in changes on the UI, which are then retranslated into a UsiXML description and send to an online evaluation tool. This evaluation tool calculates then online the workload that is needed for this status of the UI. The cognitive model, which is described in more detail in the next section, also calculates workload (e.g. for motor actions, goal switches, etc.) for the overall simulation, as well as task execution times, gaze distribution and predicts possible human errors. A simulator provides additional information, e.g. route, traffic and weather information. In an offline evaluation, it is also possible to use the UsabilityAdviser 3 for analysing the UI on compliance to certain usability rules, like certain undesired colour combinations (e.g. yellow on white background).

## Cognitive Model

The cognitive architecture CASCaS has initially been developed in the 6<sup>th</sup> European Commission Framework Programme project ISAAC [10], and has been widely extended and used in other projects since then. CASCaS has been used to successfully model perception [9], attention allocation [16], decision making (of drivers) [15] and human errors [10,11] of aircraft pilots and car drivers.

CASCaS is based on the concepts of ACT-R [1] and has been extended with Rasmussen's [13] three behaviour lev-

els in which cognitive processing takes place: skill-based, rule-based and knowledge-based behaviour. The levels of processing differ with regard to their demands on attention control dependent on prior experience: skill-based behaviour is acting without thinking in daily operations, rule-based behaviour is selecting stored plans in familiar situations, and knowledge-based behaviour is coming up with new plans in unfamiliar situations. Anderson [2] distinguishes very similar levels, but uses the terminology of autonomous, associative, and cognitive level, which will be used throughout the paper. Figure 2 gives an overview on the components of CASCaS. These components form the following control loop: The “Perception” component retrieves the current situation from the “Simulation Environment”, and stores the information in the “Memory” component. The “Processing” component contains components for the behaviour layers.

These layers can retrieve information from the memory and process this information according to their cognitive cycle (rule-based or knowledge-based). The layers may store new information in the memory, or start motor actions in the “Motor” component. Each component is based on psychologically and physiologically sound theories, e.g. from cognitive psychology, e.g. the memory component implements theories for forgetting as well as learning.

Each component implements detailed models of timing, e.g. for eye movements, such that CASCaS allows prediction of task execution times. In addition, the attention allocation can be predicted, based on top-down (rules) and bottom-up (peripheral view/selective attention) processes [9]. For the calculation of eye- and hand movements, CASCaS needs information on the positioning of the instruments. We call this information the “topology”, which is currently defined in a customized XML format, which should be exchanged by a UsiXML format in future implementations.

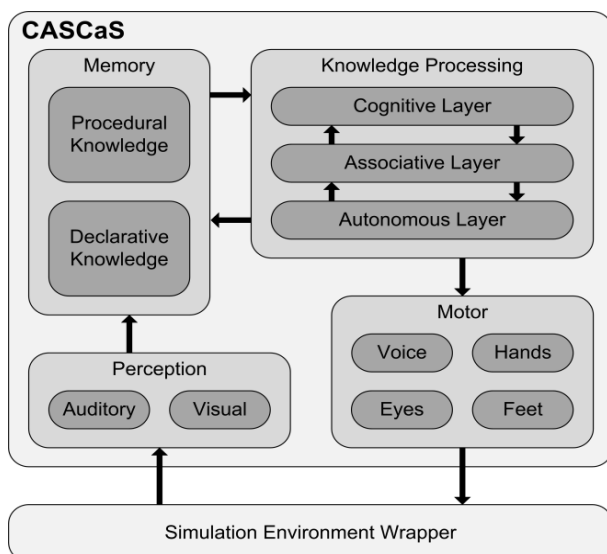


Figure 2. Architecture and components of CASCaS.

During the simulation, CASCaS will predict the task performance, performed actions (motor actions), memory consumption, as well as performed percepts (gaze distribution).

### UsiXML

UsiXML is a XML-compliant markup language which consists of a declarative User Interface Description Language (UIDL). It describes user interfaces for multiple contexts of use such as Graphical User Interfaces (GUIs), Auditory- and Multimodal User Interfaces and their constituting elements such as widgets, controls and containers [7]. Using UsiXML, a UI developer is able to model a description of interactive applications with different types of interaction techniques and modalities in a device and computing platform independent notation.

UsiXML provides an MDE approach for the specification of user interfaces and is based upon the architecture of the CAMELEON Reference Framework [5]. This framework defines UI development steps for multi-context interactive applications. Figure 3 shows a simplified version of this development process. The rendering engine is placed between the layers three and four in Figure 3. A UsiXML Concrete User Interface description serves as input data. This description is converted by a UsiXML parser and forwarded to the rendering engine. The result after this step is a Final User Interface according to the CUI.

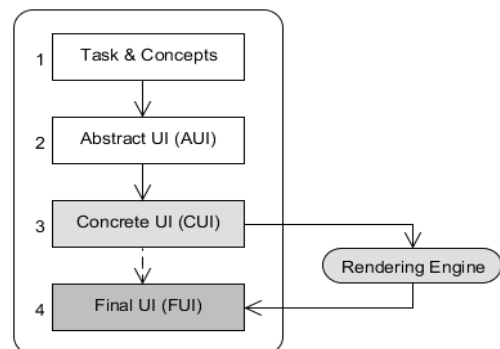


Figure 3. The Cameleon Reference Framework [5].

### Rendering Engine

Figure 4 shows the architecture of the rendering engine which consists of four main parts. First, a UsiXML parser, which conforms to a language processing system, converts a CUI description into an internal and renderable format. After this step, the converted data is passed to the rendering engine for further handling. The second component is based upon the MVC architectural pattern and handles the user actions, provides the user interface, stores the converted CUI data, supplies the application's main loop and delivers strategies for the program flow. Configuration files and log files are handled by this part of the application, too. The fourth component is a mathematical library including a useful set of algebraic and calculus functions. Finally, the rendering engine itself consists of 6 ancillary parts, as shown in Figure 5.

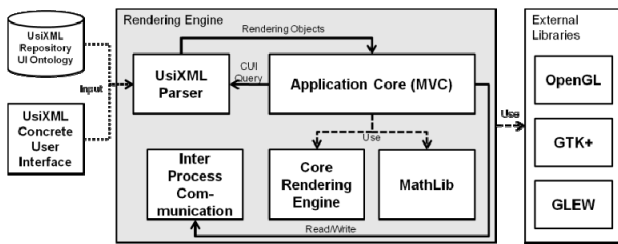


Figure 4. Architecture for Rendering Engine.

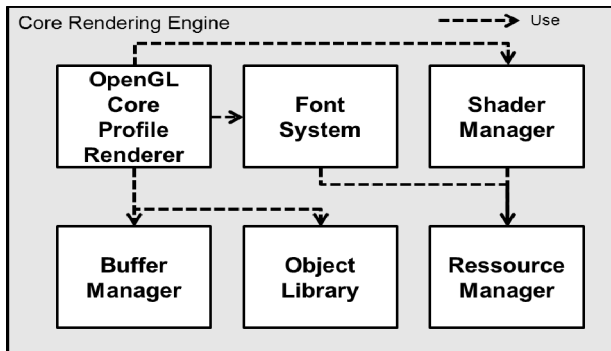


Figure 5. Components of the Core Rendering Engine.

A further component shown in Fig 4 is a module for inter process communication (IPC). This part is planned for future implementation steps, e.g. to connect the system model, or CASCAS.

The main component of the Core Rendering Engine is an OpenGL core profile renderer, which includes the functionality for drawing primitives and complex geometric objects, and allows geometry and scene management respectively manipulation. Summarized, it serves as a programming framework for creating and preparing the input for OpenGL. Further, this part includes a font system for font rendering, a buffer manager, a shader<sup>16</sup> manager for loading and preparing shader programs including a common set of shader pairs, and a resource manager for loading external resources like textures, fonts and additional shaders. The sixth component is an object library which contains a pre-rendered set of GUI objects like buttons and labels. OpenGL itself is a low-level rendering API. It does not include functions for drawing geometric objects like cylinders or spheres or GUI elements like buttons. It's up to the application developer to implement algorithms for drawing these objects. For that reason there is a need for the development of such an object library. The included object library is in an early stage and accordingly limited.

<sup>16</sup> Programmable shading is the current state of the art in real-time computer graphics. Today's graphics cards are highly programmable and the term of shader refers to according programs, written in high level languages like GLSL, HLSL or Cg, which are executed by programmable chips on modern graphics card.

## UsabilityAdviser

The global process for automatic evaluation with the UsabilityAdviser is depicted in Figure 6. The "Knowledge Base" contains a formalisation of rules for good usability and accessibility. These rules are a collection from ergonomic guidelines, for instance, structures (Smith and Mosier) or various recommendations that are encoded in a formal format, using the UsiXML language. For example, a rule that selects appropriate color combinations can be written for widgets (a slider in this example) as follows:

$$i \in \text{Slider} : \neg (\text{SliderColor}(i, \text{white}) \wedge \text{LabelColor}(i, \text{yellow}))$$

This formula expresses that yellow text on a white background is undesired. The knowledge base is used by the "Formal rules compiler" to load and parse the rules. Once this internal structure is created the tool performs a data analysis of the UI, encoded in UsiXML, which may be developed in a UsiXML editor. The UsabilityAdviser search for violations of rules formalized through the automatic evaluation of UI data. Finally, a report on the found violations of ergonomics and accessibility is presented. One major challenge is to create and update the knowledge base on ergonomic rules, which requestes a complete review and compilation of existing rules from different sources. These rules are often expressed in a natural language that is normally more complex and open compared to a programming language. Anyway, the UsabilityAdviser provides an extensible way of evaluation from multiple sources (e.g. ISO 9126, ARINC 661) of guidelines for (parts of) a User Interface.

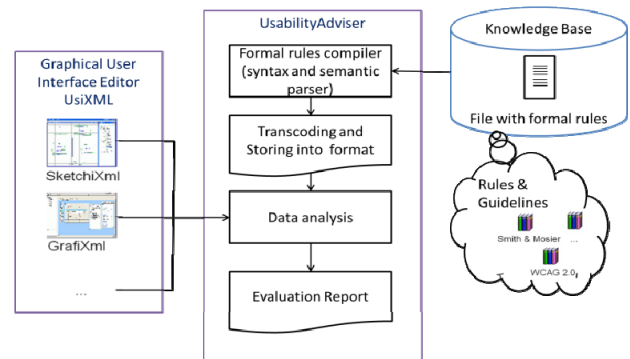


Figure 6. Global process for automatic evaluation.

## SUMMARY AND NEXT STEPS

We proposed an approach for automated UI evaluation with a cognitive architecture, which is usable in industrial application. It uses a model driven approach, and is connectable to tools that are already in use in the industry, like Matlab or Scade, which allows re-use of the models defined by the system designers. UsiXML provides a model driven development for the industry. Up to now, these tools have not been connected together for automated UI evaluation. In order to implement the proposed method in a prototypical tool, a rendering engine is needed as a connection between the cognitive model, UsiXML and the Design

Tools. We started to implement such a rendering engine in a first version. A main open issue is the connection between the design tools and UsiXML. As tools like Matlab or Scade use the mechanism of Events for interaction, an extension to UsiXML with a mapping to such events could be the solution. The rendering engine could then be extended to trigger such events when there is interaction with the UI elements, e.g. on button clicks, and to transfer events back to certain changes in the UI (e.g., opening of a dialog).

## ACKNOWLEDGMENTS

The research leading to these results has received funding from the European Commission Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 211988 (Project HUMAN, [www.human.aero](http://www.human.aero)), as well as funding from ARTEMIS JU under grand agreement n° 269336 (Project D3CoS, [www.d3cos.eu](http://www.d3cos.eu)). Juan Gonzalez would like to acknowledge of the ITEA2-Call3-2008026 USiXML (User Interface extensible Markup Language) European project and its support by Région Wallonne DGO6.

## REFERENCES

1. Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S. A., Lebiere, C., and Qin, Y. An Integrated Theory of Mind. *Psychological Review* 111, 4 (October 2004), pp. 1036-1060.
2. Anderson, J. R. *Learning and Memory*. John Wiley & Sons, Inc. (2000)
3. Bellamy, R., John, B. E., Richards J., and Thomas J. Using CogTool to model programming tasks. In *Proc. of Evaluation and Usability of Programming Languages and Tools PLATEAU'2010*. ACM Press, New York (2010), Article 1, 6 pages
4. Bossche, P. vanden. Développement d'un outil de critique d'interface intelligent : UsabilityAdviser, M.Sc. thesis, Université catholique de Louvain, Louvain-la-Neuve (2006).
5. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J. A Unifying Reference Framework for Multi-Target User Interfaces. *Interacting with Computers* 15, 3 (2003), pp. 289-308.
6. Coyette, A., Vanderdonckt, J., and Limbourg, Q. SketchiXML: A Design Tool for Informal User Interface Rapid Prototyping. In *Proc. of International Workshop on Rapid Integration of Software Engineering techniques RISE'2006* (Geneva, 13-15 September 2006). N. Guelfi, D. Buchs (Eds.), Lecture Notes in Computer Science, vol. 4401. Springer-Verlag, Berlin (2007), pp. 160-176.
7. Guerrero Garcia, J., Lemaigre, C., González Calleros, J. M., and Vanderdonckt, J. Model Driven Approach to Design User Interfaces for Workflow Information Systems. *Journal of Universal Computer Science* 14, 19 (2008), pp. 3160-3173.
8. John, B.E. Using Predictive Human Performance Models to Inspire and Support UI Design Recommendations. In *Proc. of CHI'2011*. ACM Press, New York (2011)
9. Lüdtkke, A. and Osterloh, J.-P. Simulating Perceptive Processes of Pilots to Support System Design. In *Proc. of IFIP TC13 Int. Conf. on Human-Computer Interaction INTERACT'2009*. Springer, Berlin (2009), pp. 471-484.
10. Lüdtkke, A., Cavallo, A., Christophe, L., Cifaldi, M., Fabbri, M., and Javaux, D. Human Error Analysis based on a Cognitive Architecture. In *Proc. of Int. Conf. on Human-Computer Interaction in Aeronautics HCI-Aero'2006* (Seattle, 20-22 September 2006). F. Reuzeau, K. Corker, G. Boy (Eds.). Cépaduès-Éditions, Toulouse (2006) pp. 40-47
11. Lüdtkke, A., Osterloh, J.-P., Mioch, T., Rister F. and Looije, R. Cognitive Modelling of Pilot Errors and Error Recovery in Flight Management Tasks. In *Proc. of IFIP TC13.5 Working Conf. on Human Error, Safety and Systems Development HESSD'2009* (Brussels, September 2009). Lecture Notes in Computer Science, vol. 5962. Springer-Verlag, Berlin (2010), pp. 54-67.
12. Molina, J.P., Vanderdonckt, J., Montero, F., and González, P. Towards Virtualization of User Interfaces based on UsiXML. In *Proc. of Web3D 2005 Symposium, 10th International Conference on 3D Web Technology* (Bangor, 29 March-1 April 2005). ACM Press, New York (2005), pp. 169-178
13. Rasmussen, J. Skills, Rules, Knowledge: Signals, Signs and Symbols and other Distinctions in Human Performance Models. *IEEE Transactions: Systems, Man and Cybernetics, SMC-13* (1983), pp.257-267.
14. Vanderdonckt, J., and Beirekdar, A. Automated Web Evaluation by Guideline Review. *Journal of Web Engineering* 4, 2 (2005), pp. 102-117.
15. Weber, L., Baumann, M., Lüdtkke, A., and Steenken, R. Modellierung von Entscheidungen beim Einfädeln auf die Autobahn. In A. Lichtenstein, C. Stöbel, C. Clemens (Hrsg.), 8. Berliner Werkstatt, Mensch-Maschine-Systeme. VDI Verlag, Düsseldorf (2009), pp. 86-91.
16. Wortelen, B. And Lüdtkke, A. Ablauffähige Modellierung des Einflusses von Ereignishäufigkeiten auf die Aufmerksamkeitsverteilung von Autofahrern. In *Mensch-Maschine-Systeme*, A. Lichtenstein, C. Stöbel, C. Clemens (Hrsg.), 8. Berliner Werkstatt, VDI Verlag, Düsseldorf, pp. 80-85.

# User Interface Generation for Maritime Surveillance: An initial appraisal of UsiXML V2.0

Charles R. Robinson<sup>1</sup>, Frédéric Cadier<sup>2</sup>

<sup>1</sup>THALES Research and Technology, Campus Polytechnique  
1, Av Augustin Fresnel 91767 PALAISEAU Cedex (France)  
charles.robinson@thalesgroup.com

<sup>2</sup>Institut TELECOM, TELECOM Bretagne, Technopôle Brest Iroise,  
CS 83818, 29238 Brest Cedex3 (France)  
frederic.cadier@telecom-bretagne.eu

## ABSTRACT

In the capacity of being among the first to implement the latest version of UsiXML, a language for user interface development and implementation, this article describes some initial work in its application to a real world situation. After presenting a general approach for the use of this standard, a scenario is described from the domain of Maritime Surveillance. With a model driven focus, UsiXML recommends creating a model of the tasks as the first stage for generating user interfaces. A discussion is provided on our implementation of this particular stage and the available tools that were used.

## Author Keywords

Task model, user interface generation, real-life applications, maritime surveillance, model-driven engineering, UsiXML.

## General terms

Application, Design, Interfaces.

## INTRODUCTION

In general the use of standards for development means assurance that the end product will have particular characteristics. Their use can be for a great range of reasons such as:

1. Safety, security or performance.
2. The ability to compare different products.
3. Guidelines for development processes, providing best practices for high quality.
4. Interoperability between different products and reuse for new applications.

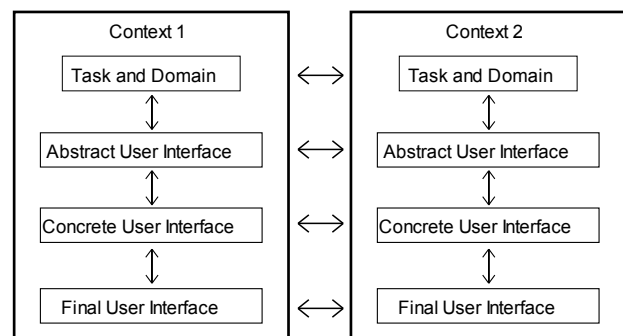
UsiXML has been created to be a standard for the purpose of user interface development and deployment. It enables several of the above properties, such as using the model driven engineering (MDE) approach of the OMG (Object Management Group). The structure of the UsiXML language provides the interoperability and flexibility to realise the  $\mu 7$  concept that means interfaces may be developed with multi-device / user / linguage / organisation / context / modality and multi-platform applicability.

The ability to dynamically present the same information through different devices and in different environments is crucial to being able to respond quickly to threatening situ-

ations and to support emergency services. Whilst UsiXML is still being fine tuned, and its associated tools being developed, this article discusses the first stages of applying UsiXML in the context of maritime surveillance. This is a suitable domain for the application of UsiXML because there are many people involved who are tasked with different responsibilities, operating at different terminals and requiring tailored methods for the presentation of similar information.

## OVERVIEW OF USIXML

UsiXML is currently funded by ITEA2 for the definition, validation and standardisation of a user interface definition language (UIDL). It has evolved over the past ten years and is the result of several joint international projects [1] including Cameleon [2] and Salamandre [3]. UsiXML specifies four levels that should be attained in order to realise an interface for an end user. Similar to the MDE approach, UsiXML includes equivalents to the task layer, platform independent layer and platform specific layer. In UsiXML, the task layer encompasses task and domain models. The platform independent model corresponds to the Abstract User Interface level of UsiXML and the platform specific level is represented by the Concrete User Interface. The remaining level of UsiXML is the Final User Interface which is the implementation of the modelled interface for the end user (Figure 1).



**Figure 1. The four stages of a user interface. Translation or translation is used to move between stages or to other contexts.**

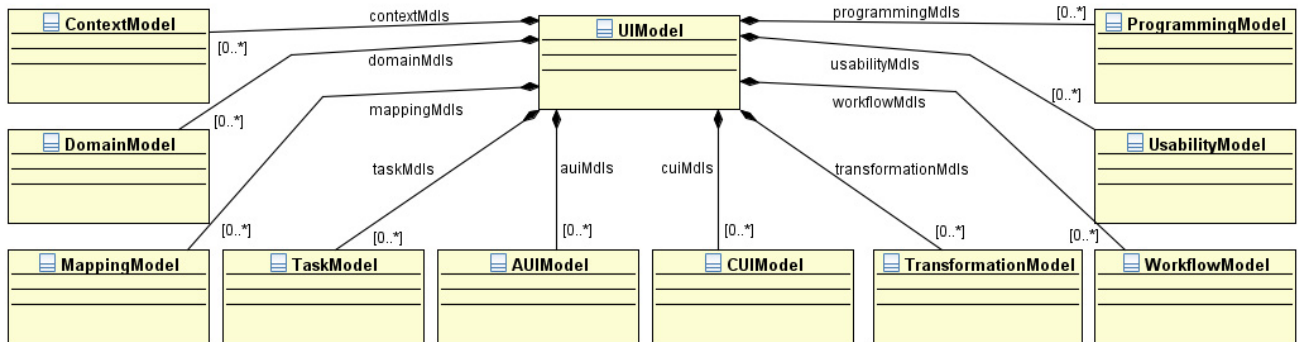


Figure 2. The models that may be used for generating an interface.

In addition to these four levels, there exist models for context, workflow, transformation between the four levels and mapping (translation) to the same level in a different context. Two further models were introduced recently for usability and programming. A UML representation of the different models that can be used to build the final interface is shown in Figure 2. It is important to note that not all are necessarily required for creating a user interface - it depends on the application.

#### A MARITIME SURVEILLANCE SCENARIO

Maritime surveillance is the monitoring of activity at sea, usually with the assistance of aircraft. It has many applications including the detection of illegal trafficking, immigration and piracy, monitoring of fishing lanes and search and rescue activities. There can be many personnel involved in such operations, often each person having different information requirements, or requiring a different representation of the same information.

The initial scenario that we draw on uses an aircraft, similar to a Dassault Falcon, and two unmanned aerial vehicles (UAVs) like the Watchkeeper. The aircraft and UAVs are equipped with video and EO/IR sensors (electro-optical and infra-red) that capture information about objects of interest. The vehicles are in direct communication with a ground command post. At such a command post we consider four types of operator that require a user interface in order to participate in the surveillance mission:

1. *TACCO*: Tactical operations, this person leads the mission by directing the surveillance activities.
2. *SENSO*: Operate sensors on the piloted aircraft in order to analyse nearby targets.
3. *UAV-Ctrl*: Responsible for tasking and control of UAVs, either through commands in autonomous mode or remotely piloting the vehicle.
4. *UAV-Mis*: Sensor operations of the UAV to analyses targets that are nearby.

The physical system in the ground control centre is made up of a multi-touch table for the TACCO and all cooperative tasks, and some individual workstations for the SENSO and UAV operators.

The TACCO operates at the table, where he has a global tactical view of his zone of operation and all the objects within this range detected by radar on-board the vehicles. His role is to maintain a so-called tactical situation awareness by monitoring these objects, checking that they are correctly identified by sensor operators and engaging mission specific action when needed, depending on what has been identified.

The aircraft and UAVs are used to assist with the identification. The TACCO and the UAV-Ctrl cooperate at the table to divide the tactical area into a zone assigned to the plane and one or two zones with an assigned drone. Following this, the TACCO and UAV-Ctrl define navigation patterns that will allow the vehicles to cover the whole area, taking the sensors ranges into account.

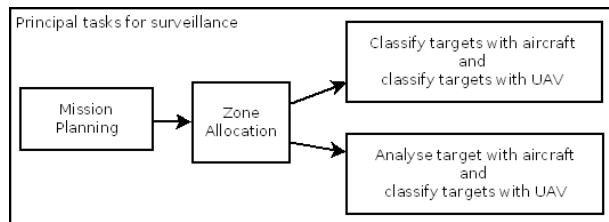
Objects are then identified by the SENSO or UAV-Mis when the manned or unmanned aircraft, respectively, approach a target. In turn, this identification is validated by the TACCO. Should an object be found requiring more investigation, the aircraft is reassigned to gather more data on the object of interest. The drones are then directed to survey other locations, where, for the time of this specific investigation, the UAV-Ctrl and UAV-Mis are responsible for maintaining the tactical situation.

#### CREATING A TASK MODEL WITH USIXML

The first step taken, and perhaps the most challenging, has been to create a generalised model of the flow of operations that take place in order to realise the aforementioned scenario. Figure 3 groups these operations into the main activities: mission planning, allocation of zones to the three aircraft and then either general surveillance of targets in these areas, or the manned aircraft conducts an in-depth investigation of a particular target. Each of these



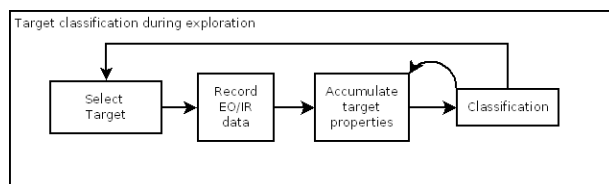
activities contains a set of operations, all of which will require representation in UsiXML. With four operators involved, a separate Task Model is required to be developed for each one. Not all the sub-operations or activities from the generalised model will be represented in each user model as the operators have different requirements.



**Figure 3. Main operations in this maritime surveillance scenario.**

As an example, we describe our implementation of the Task Model for the SENSO, focusing on the operation of classifying nearby targets (or target identification). It is shown initially as a higher level flow of operations (Figure 4). Later we break this down its constituent subtasks and represent it in UsiXML using the Task Editor (TE).

The cycle of operations, or workflow, that the SENSO carries out when involved with the surveillance activity of target identification is depicted in Figure 4. This activity follows on from the zone allocation and it is the responsibility of the SENSO to direct the sensors at each target the aircraft approaches and obtain sufficient information to classify and profile this target. The same role also applies to the UAV-Mis, but in the context of the unmanned aircraft. For clarity we shall only refer to the SENSO in this section. The operations shown in Figure 4 represent the principal tasks, each being composed of subtasks that will need to be realised on the interface of the SENSO in order to carry-out an operation.



**Figure 4. Operations for target identification during general surveillance. An activity of both the SENSO and UAV-Mis.**

The TE for UsiXML initially represented the tasks and subtasks in a tree structure, with subtasks branching out from higher level tasks. A temporal link could be placed between each successive subtask in a particular node to indicate their order of execution. Properties of a task, such as *optional* or *looping*, can be described by adding a

decoration component containing relevant fields. The characteristics of, and relationships between, the various tasks are stored using the UsiXML format in an XML file. This tree structure was useful for visualisation purposes, and as such using the initial TE, Figure 6 presents the previously described principal tasks and subtasks that need representation for the SENSO.

In the current framework of UsiXML, there are circumstances where one must add virtual tasks to the model in order to represent some properties of a group of user tasks. That is, elements that are used as placeholders, having no bearing on an actual task. For example, a task that loops may be decorated with an iterative property. However, when several tasks need to be encompassed within a loop, one needs to create a task to represent the ‘looping’ behaviour. This group of tasks are then placed as subtasks of the looping task. This is also the case where a temporal link from one task affects several other tasks.

The TE has since evolved to a more compact form, representing the tasks within a box structure. The outer box represents the top level task where lower level subtasks are boxes contained within. This alternative expression of our Task Model is shown in Figure 7 and Figure 8. Two particular changes with the new version are that tasks now have their decorations and temporal associations described in the higher level task. This provides one with the capability to express different properties or task relationships for different circumstances. A case in point being situations that require some tasks to be carried out in a particular order at system start-up but may run concurrently afterwards. In the example provided, the outer loop represents the task of exploration/target identification and will have this iterative property set at the higher level in the task model. For demonstrative purposes we name the inner cycle ‘Loop’, where the user has the option to return to task “Accumulate track properties”. However, generally loops have a purpose and it is this that should be used as the name (in this case the purpose being to change incorrect identifications).

## FUTURE WORK AND THOUGHTS ON USIXML

In terms of our application of UsiXML, the next step, using the recently released Domain Editor tool, will be to create the Domain Model. This model may be thought of as a template that describes the attributes within the system that the user may manipulate. One defines here the classes of interface components, their functions and required variables. Given this structure, UsiXML draws on UML (the Unified Modelling Language) to represent the Domain Model.

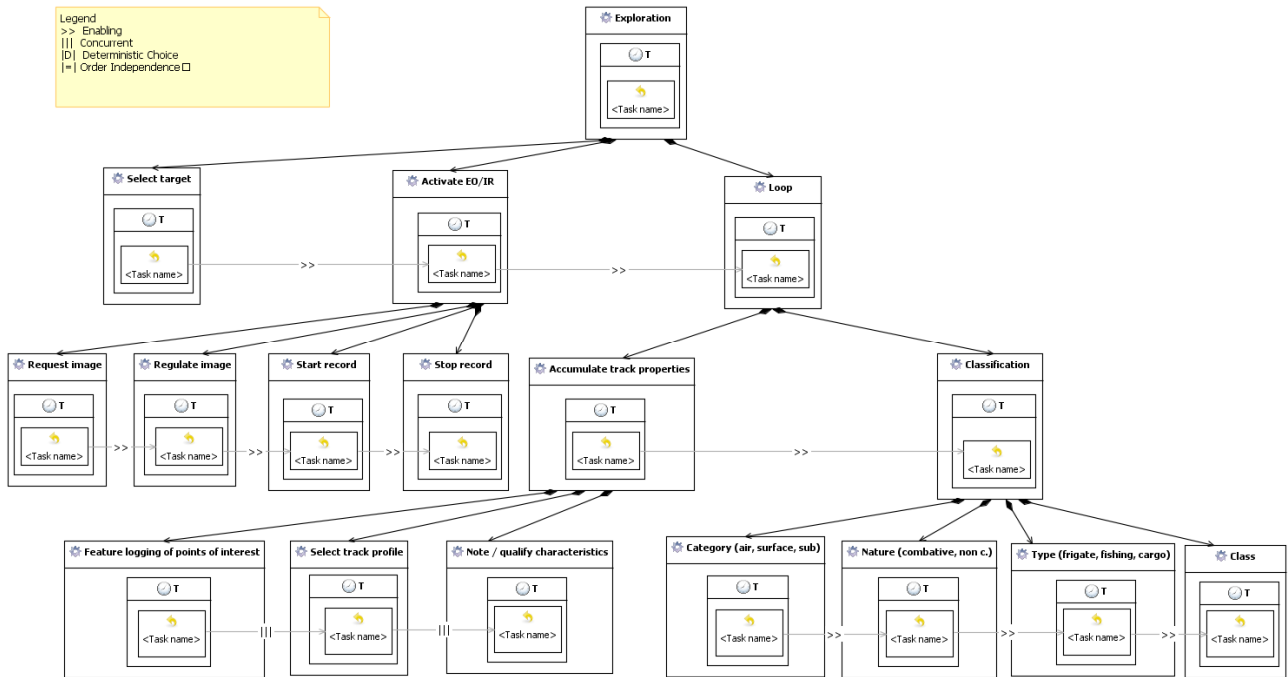


Figure 5. UsiXML model of the tasks required by the SENSO for general surveillance.

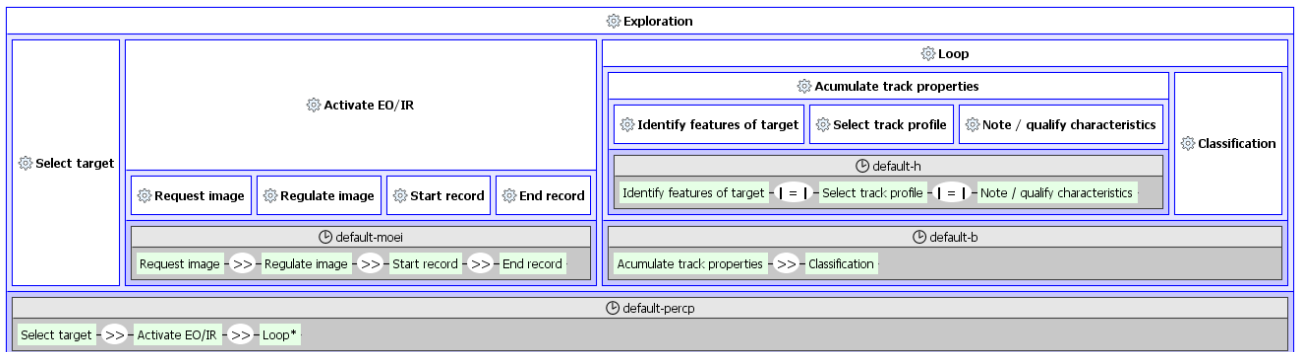


Figure 6. Task Model of the operations carried out by the SENSO during exploration/surveillance of the aircraft. For manageability, the TE provides the capability to hide subtasks, this has been done for the task “Classification”.

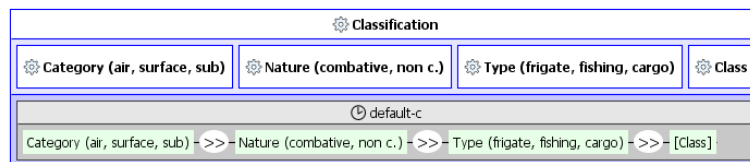


Figure 7. The Classification task - A subtask of the Exploration/Target identification Task Model.

When the Domain Model has been completed we will need to define the transformation rules to convert the first level UsiXML models to the second level, the AUI model. A transformation engine will be needed to gather the appropriate information from the initial level and populate the AUI model with the relevant data. Out of the options available, the general consensus within the UsiXML community has converged on the use of ATL (the Atlas Transformation Language) to define the necessary rules. Turning to thoughts of UsiXML, being immersed in the language at the moment, one finds it provides a most intuitive structure for interface development. In the current project, the underlying models of UsiXML have undergone some serious refinement and development over the past years and is the reason why new versions, or completely new tools are currently appearing. There are still some features that it would be useful to see addressed, or developed further.

Regarding the UsiXML Task Editor, the new box layout provides important benefits over the tree-structure. This includes a more efficient representation and the ability to describe different temporal relationships or task properties for different situations. However, in larger models it can be difficult to visualise the overall picture. The TE offsets this to an extent by providing the developer with the ability to hide any subtasks that are not currently being considered. Perhaps the UsiXML workflow model, currently being developed, will be better able to address this aspect of design.

It also seems a bit restrictive in the TE to have a SISO relationship between tasks (single input and single output). Of course there are ways one can overcome this to an extent. For example, to have one task (T1) that enabled three tasks (ABC), we could group these three tasks inside another task (T2). We could then set an enabling link between T1 and T2 and set A, B and C as concurrent. However, when a series of such instances are required, the model can quickly become rather unwieldy, with many tasks simply acting as containers. It would also be difficult to model one task having a different affect on several tasks.

Another tool that will be desirable in the near future would provide the means to select tasks from a library of previously designed interfaces. This would then auto-complete the associated links to all the required models. This then provides the ability for the user to automate generation of the associated subtasks, domain functions, etc. up to the point of code generation for the FUI.

With respect to the UsiXML language in general, one thing that may have become apparent during the course of this article is that after we considered the overall flow of operations, a separate task model was developed for each operator. There is no mechanism within UsiXML to define a global task model with multiple users, that has the capacity to derive the individual user task models. In general each

user interface is rather a stand-alone entity. As seen in the paper by Frey *et al.* [4], it is tricky to represent tasks that require external data to commence, or tasks that are enabled by the completion of the tasks of other users. In the short-term we plan to introduce a central system as a fifth user that can act as a repository for the information exchange among the operators. It would be desirable in the longer term to have a feature that included techniques or methods for representing the effects of events outside a particular interface. This would represent tasks occurring at other interfaces, or tasks separate from the computerised element such as a task requiring direct communication between users for resolving unforeseen problems.

With the current foundation of UsiXML, there is good groundwork in place for developing other technologies related to user interfaces. For example agent-related technology will have an important role, particularly in situations such as maritime surveillance. One can envisage them taking the form of decision aids, or indeed representing sensor platforms or vessels within a zone, augmenting or simplifying an operators interface as the environment changes. Another technology that will benefit from the UsiXML framework is the development of techniques and methods that learn about the user and are able to adapt interfaces for different operators.

## CONCLUSION

UsiXML is a UIDL that provides a model driven approach for the creation and application of user interfaces with high degrees of flexibility. An international project is currently in progress to bring this language to the point where it has suitable quality and robustness for standardisation. In this article we focus on the first level of the UsiXML user interface development process and introduce the Task Editor. Using the latest version of UsiXML, we apply it to the generation of four user interfaces for performing the maritime surveillance activity of target classification and investigation. These interfaces provide communication channels for the operation of sensor platforms and different levels of awareness of the tactical situation. While there remain some features it would be nice to see integrated into the UsiXML framework, it nevertheless provides a solid foundation and freedom for flexible interface design and implementation.

## ACKNOWLEDGMENTS

This work is being undertaken with the support of the ITEA2-Call3-2008026 UsiXML (User Interface Extensible Markup Language – [www.itea2.usixml.org](http://www.itea2.usixml.org)) European project. Special thanks go to David Faure (Thales), Olivier Grisvard (Télécom Bretagne) and PY-Automation for their expert knowledge and contribution to the implementation of the UsiXML demonstrator that has been discussed in this paper.

## REFERENCES

1. Limbourg, Q. and Vanderdonckt, J. UsiXML: A User Interface Description Language Supporting Multiple Levels of Independence. In *Engineering Advanced Web Applications*, M. Matera, S. Comai, S. (Eds.). Rinton Press, Paramus (2004), pp. 325-338
2. Chesta, C., Paternò, F., and Santoro, C. Methods and Tools for Designing and Developing Usable Multi-Platform Interactive Applications. *PsychNology Journal* 2, 1 (2004), pp. 123-139.
3. Vanderdonckt, J. A MDA-Compliant Environment for Developing User Interfaces of Information Systems. In *Proc. of 17th Conf. on Advanced Information Systems Engineering CAiSE'05* (Porto, 13-17 June, 2005). O. Pastor & J. Falcão e Cunha (Eds.). Lecture Notes in Computer Science, vol. 3520. Springer-Verlag, Berlin (2005), pp. 16-31.
4. García Frey, A., Céret E., Dupuy-Chessa, S., and Calvary, G. QUIMERA: A Quality Metamodel to Improve Design Rationale. In *Proc. of the 3rd ACM SIGCHI Symposium on Engineering interactive computing systems EICS'2011* (Pisa, June 13-16). ACM Press, New York (2011).



# UsiXML'2011

UsiXML'2011, the 2<sup>nd</sup> International Workshop on User Interface eXtensible Markup Language, was held in Lisbon, Portugal (September 6, 2011) during the 13th IFIP TC13 International Conference on Human-Computer Interaction Interact'2011 (Lisbon, September 5-9, 2011).age. This edition is devoted to software support for **any** User Interface Description Language.

A User Interface Description Language (UIDL) is a formal language used in Human-Computer Interaction (HCI) in order to describe a particular user interface independently of any implementation. Considerable research effort has been devoted to defining various meta-models in order to rigorously define the semantics of such a UIDL. These meta-models adhere to the principle of separation of concerns. Any aspect of concern should univocally fall into one of the following meta-models: context of use (user, platform, environment), task, domain, abstract user interface, concrete user interface, usability (including accessibility), workflow, organization, evolution, program, transformation, and mapping. Not all these meta-models should be used concurrently, but may be manipulated during different steps of a user interface development method. In order to support this kind of development method, software is required throughout the user interface development life cycle in order to create, edit, check models that are compliant with these meta-models and to produce user interfaces out of these methods.



EUREKA 1985  
2010  
25  
Doing business through technology

Published by  
Thales Research & Technology France,  
September, 2011



Avec le soutien de la DGO6  
Département des Programmes  
de Recherche

direction générale de la compétitivité  
de l'industrie et des services

*Supported by  
DGO6, Département des  
Programmes de Recherche,  
Service Public de Wallonie, Belgium  
& by  
DGCIS - Ministère de l'Économie, des Finances  
et de l'Industrie, France  
September, 2011*

The UsiXML project is the ITEA 2 project # 08026.  
ITEA 2 is the Eureka Project # 3674

ISBN 978-2-9536757-1-9



9 782953 675719

EAN 9782953675719

THALES

