



Graceful Degradation: a Method for Designing Multiplatform Graphical User Interfaces

By Murielle Florins

A thesis submitted in fulfilment of the requirements for the degree of

Doctor of Philosophy in
Management Sciences
of the Université catholique de Louvain

Examination committee:

Prof. Jean Vanderdonckt, Advisor
Prof. Manuel Kolp, Examiner
Prof. Benoît Macq, Examiner
Prof. Gaëlle Calvary, Université J. Fourier, Reader
Prof. Philip Gray, Reader

Acknowledgement

I would like to express my thanks to:

- My advisor, Professor Jean Vanderdonckt, for his constant support and enthusiasm regarding my work.
- Professors Gaëlle Calvary, Philip Gray, Benoît Macq, and Manuel Kolp for accepting to participate to the jury of this thesis.
- My colleagues from IAG school of management at Université catholique de Louvain. Special thanks to Marco Saerens for his help on statistical analysis and Benjamin Michotte for his development efforts on the GD plug-in.
- My family and friends.

This thesis was realized thanks to the support of:

- The Salamandre Project, funded by the ‘Initiatives IIP’ program of the Ministry of Walloon Region (DGTRE, Belgium) under contract No. 001/451 (in association with the ARTHUR project, under convention WDU 9914094-14624 of Walloon Region).
- The ISYS research unit at IAG.
- The SIMILAR network of excellence supported by the 6th Framework Program of the European Commission, under contract FP6-IST1-2003-507609 and the CAMELEON research project under the umbrella of the European Fifth Framework Programme (FP5-IST4-2000-30104).

Table of Contents

ACKNOWLEDGEMENT	III
TABLE OF CONTENTS.....	1
TABLE OF FIGURES	5
TABLE OF TABLES	8
CHAPTER 1 INTRODUCTION.....	9
1.1 Motivation: the Challenge of Developing User Interfaces for Multiple Platforms	9
1.2 Thesis	10
1.2.1 Thesis statement	10
1.2.2 Definitions / motivations	11
1.2.3 Graceful degradation	15
1.2.4 Focus	16
1.3 Reading Map.....	17
CHAPTER 2 STATE OF THE ART	19
2.1 A structuring theoretical framework.....	19
2.2 Approaches to the development of user interfaces for multiple platforms.....	22
2.2.1 The traditional development approach	23
2.2.2 The unique portable code approach.....	25
2.2.3 The transcoding approach.....	29
2.2.4 The multireification approach	30
2.2.5 The abstraction-reification approach.....	36
2.3 Single-authoring	38
2.4 Comparison of the approaches.....	38
2.4.1 Production costs.....	39
2.4.2 Completeness.....	42
2.4.3 Level of control	42
2.4.4 Usability	43
2.4.5 Cross-platform consistency	44
2.4.6 Guidance.....	45

2.5	Global comparison and conclusion	45
CHAPTER 3 LANGUAGE AND MODELS.....		48
3.1	UsiXML	49
3.2	Task Model.....	51
3.3	Domain Model.....	51
3.4	AUI Model.....	52
3.5	CUI Model.....	55
3.6	Platform Model.....	57
3.7	Interactor Model.....	59
3.7.1	Requirements	59
3.7.2	State-of-the-art of meta representations of widgets.....	61
3.7.3	Interactor model in the GD approach	66
3.8	Mapping Model.....	69
3.9	Conclusion.....	70
CHAPTER 4 EFFECTIVE KNOWLEDGE FOR GRACEFUL DEGRADATION		71
4.1	Typology of rules using the Unified Reference Framework.....	71
4.1.1	GD rules at the Final User Interface level	72
4.1.2	GD rules at the Concrete User Interface level.....	72
4.1.3	GD rules at the Abstract User Interface level.....	80
4.1.4	GD rules at the Tasks & Concepts level.....	83
4.1.5	Discussion	85
4.2	Formalization.....	85
4.2.1	Introduction	85
4.2.2	Specification of GD rules in OCL: examples.....	86
4.3	Discussion and conclusion.....	89
CHAPTER 5 MULTILEVEL APPLICATION OF RULES: EXAMPLE OF THE SPLITTING RULE.....		92
5.1	State of the art of pagination techniques.....	92
5.1.1	Pagination of Web pages	92
5.1.2	Pagination of content expressed in a user interface description language	93

5.2	Splitting at the Concrete UI level	94
5.3	Splitting at the Abstract UI level.....	96
5.3.1	Preliminaries.....	97
5.3.2	Principles	99
5.3.3	Description of the algorithm.....	103
5.4	Conclusion.....	105
CHAPTER 6	TOOL SUPPORT	106
6.1	A knowledge base of GD rules.....	106
6.1.1	Selection criteria.....	107
6.1.2	Structure of the knowledge base.....	107
6.1.3	Support of the adaptation process in the knowledge base.....	117
6.2	A tool support for GD rules.....	118
6.2.1	A plug-in to the GrafiXML editor	118
6.2.2	Functionalities	119
6.2.3	Scenario supported by the tool	121
6.2.4	Support of the adaptation process in the GD plug-in	125
6.2.5	User testing.....	125
CHAPTER 7	CASE STUDIES	128
7.1	ARTHUR.....	128
7.1.1	Introduction	128
7.1.2	Application of GD Rules	131
7.1.3	Conclusion.....	134
7.2	Semi-automatic adaptation of the rules: a hotel booking system	134
7.2.1	The hotel booking system.....	134
7.2.2	Production of the target UIs	137
7.2.3	Conclusion.....	137
CHAPTER 8	VALIDATION.....	140
8.1	Theoretical validation	140
8.1.1	Production costs.....	140
8.1.2	Completeness.....	142
8.1.3	Level of control	142
8.1.4	Usability	143
8.1.5	Cross-platform consistency	143
8.1.6	Guidance.....	144
8.1.7	Conclusion.....	144
8.2	Empirical validation.....	146

8.2.1	Goals of the experiment	146
8.2.2	Experimental UIs.....	147
8.2.3	Participants	150
8.2.4	Tasks.....	150
8.2.5	Questionnaires	151
8.2.6	Experimental procedure.....	151
8.2.7	Results	152
8.2.8	Conclusion	157
CHAPTER 9 CONCLUSION.....		158
9.1	Summary of results.....	158
9.1.1	Theoretical and conceptual contributions.....	158
9.1.2	Methodological contribution	159
9.1.3	Tools developed	159
9.2	Future work in prospect	159
9.3	Concluding remarks.....	161
REFERENCES.....		162
ANNEX A. CTT/USIXML TASK MODEL		179
ANNEX B. DISCUSSION OF USIXML'S PLATFORM MODEL ATTRIBUTES		181
ANNEX C. AN INTERACTOR MODEL FOR THE QTK TOOLKIT		191
ANNEX D. AN OVERVIEW OF OCL		197
ANNEX E. DESCRIPTION OF THE RULES TO BE IMPLEMENTED IN THE		200
GD TOOL		

Table of Figures

Figure 1-1 Constraints of some current hardware platforms (from [Pier04])	14
Figure 1-2 Graceful Degradation in a nutshell	16
Figure 2-1 The four abstraction levels in the Unified Reference Framework.....	20
Figure 2-2 Two sublevels in the Final User Interface	21
Figure 2-3 Relationships between components in the Unified Reference Framework	22
Figure 2-4 Traditional development of multiplatform UI	23
Figure 2-5 The web site of American Airlines: an example of traditional development of multiplatform UI.....	24
Figure 2-6 Development of portable UI using a virtual toolkit.....	25
Figure 2-7 Development of portable UI using browsers	25
Figure 2-8 Adaptation of the UI's look-and-feel using Java Swing	26
Figure 2-9 Automatic adaptation of a layout to the size of a container using a layout manager in Java Swing.....	27
Figure 2-10 Development of multiplatform UI using transcoding tools.....	29
Figure 2-11 Transcoding tools in the identity configuration.....	29
Figure 2-12 Transcoding tools in the direct configuration.....	29
Figure 2-13 Development of multiplatform UI in the multireification approach	31
Figure 2-14 The abstraction-reification approach	37
Figure 3-1 Constituent models in UsiXML.....	49
Figure 3-2 Meta-model of the UsiXML task model.....	51
Figure 3-3 Meta-model of the UsiXML domain model	52
Figure 3-4 Meta-model of the UsiXML Abstract User Interface.....	54
Figure 3-5 Concrete Interaction Objects in UsiXML: upper part of the hierarchy	55
Figure 3-6 UsiXML's platform model.....	59
Figure 3-7 Same toolbox / distinct environments: the DatePicker object on Pocket PC (left) and Smartphone (right).....	61
Figure 3-8 Same environment / distinct toolboxes: the TreeView object in Windows MFC (left) and Java Swing (right).....	61
Figure 3-9 Modelling of comets (from [Calv04])	65
Figure 3-10 Class diagram of the concepts involved in our interactor model.....	67
Figure 3-11 Hierarchy of task types	68
Figure 3-12 Mappings in UsiXML.....	70
Figure 4-1 Reorientation rule	73
Figure 4-2 Automatic application of a repositioning rule by an on-line reformatting service	74
Figure 4-3 Resizing rule applied in combination with image cropping	76
Figure 4-4 Component replacement due to unavailability	77
Figure 4-5 Candidate interactors for multiple choice.....	77
Figure 4-6 Candidate interactors for simple choice	77
Figure 4-7 Interactor substitution: example of composite substitution	78
Figure 4-8 Substitution rule (1)	79

Figure 4-9 Substitution rule (2)	79
Figure 4-10 Substitution rule (3)	80
Figure 4-11 Defining Abstract Containers for a simple information retrieval system.....	82
Figure 4-12 Internal redundancy due to splitting rule	83
Figure 4-13 Example of temporal ordering transformation	85
Figure 5-1 Unidirectional/bidirectional linear navigation	96
Figure 5-2 Unidirectional/bidirectional indexed navigation	96
Figure 5-3 Unidirectional/bidirectional mixed navigation	96
Figure 5-4 Fully-connected navigation	96
Figure 5-5 Priority ordering between the temporal operators in the task model	97
Figure 5-6 A task model and its priority tree representation	98
Figure 5-7 Task model for a simple IR system, with different distribution of tasks among interaction spaces.....	100
Figure 5-8 Splitting an interaction space containing a sequence of tasks, one of them being an optional task.....	101
Figure 5-9 Splitting an interaction space at the level of a concurrent operator.....	101
Figure 5-10 Splitting an interaction space containing concurrent tasks at different level in the hierarchy	102
Figure 5-11 An example of distribution of a disabling task	103
Figure 5-12 Scope of a temporal operator: illustration	104
Figure 6-1 Class diagram of the concepts of the knowledge base	109
Figure 6-2 Relational model of the database	114
Figure 6-3 Functionalities offered by the Web-based interface to the knowledge base ..	115
Figure 6-4 The consultation of rules in the interface to the database.....	116
Figure 6-5 Support of Dieterich's four adaptation stages in the knowledge base.....	118
Figure 6-6 The five groups of rules in the GD plug-in.....	119
Figure 6-7 Details panel linked to a selected rule	120
Figure 6-8 Access to the knowledge base from the GD plug-in	121
Figure 6-9 A very simple UsiXML specification in textual format and in the GrafiXML composer	122
Figure 6-10 Description of a rule in the GD tool	123
Figure 6-11 Selection of rules and parameters in the GD tool.....	124
Figure 6-12 Selection of the components to which the rule applies.....	124
Figure 7-1 Problems raised by ARTHUR's user interfaces	129
Figure 7-2 Task model of the ARTHUR prototype.....	130
Figure 7-3 Subtasks for the nursing records management	130
Figure 7-4 Example of a subtask detail	131
Figure 7-5 Desktop version of the ARTHUR prototype: source interface.....	131
Figure 7-6 An alternative for the PDA version of ARTHUR: deleting edition tasks	132
Figure 7-7 Interactor substitution in ARTHUR.....	133
Figure 7-8 Design alternatives for ARTHUR's PDA user interfaces (mock-ups in HTML)	134
Figure 7-9 The source UI of the hotel booking system in the GrafiXML editor	135
Figure 7-10 UsiXML code of the source UI	136
Figure 7-11 Preview of the source UI	136
Figure 7-12 Design alternatives for a small target device (preview in GrafiXML) - 1....	138
Figure 7-13 Design alternative for a small target device (preview in GrafiXML) - 2....	139

Figure 8-1 Screenshot of the Iacchos Web site: first source user interface of the experiment	147
Figure 8-2 Screenshot of the Maporama Web site: second source user interface of the experiment	148
Figure 8-3 “Splitted” version of the Iacchos Web site, with sequential navigation.....	149
Figure 8-4 “Splitted” version of the Maporama Web site, with fully-connected navigation	149
Figure 8-5 Preferences expressed on Iacchos / Maporama versions	153
Figure 8-6 Perceived similarity of the PDA versions with the desktop version in terms of functionalities	156
Figure 8-7 Perceived similarity of the PDA versions with the desktop version in terms of presentation.....	156
Figure A-0-1 The personal computing continuum (illustration from [Weis02])	188
Figure A-0-2 Handheld devices categories (illustration from [Weis02]).....	189

Table of tables

Table 2-1 Global comparison of development approaches on all criteria	46
Table 6-1 Decomposition of the moving rule	106
Table 6-2 Detail of the plug-in's five sections.....	120
Table 6-3 The CSUQ questionnaire.....	126
Table 8-1 Compared costs of developing UIs for multiple platforms	141
Table 8-2 Compared costs of modifying/adding a functionality on multiple platforms	141
Table 8-3 Compared costs of modifying/adding a format on UIs deployed on multiple platforms	142
Table 8-4 Compared completeness of the development approaches	142
Table 8-5 Compared level of control offered by the development approaches	143
Table 8-6 Compared usability of UIs targeted to very distinct platforms	143
Table 8-7 Compared cross-platform consistency between the UIs produced.....	144
Table 8-8 Global comparison of graceful degradation and other approaches on all criteria	145
Table 8-9 The 12 categories of subjects recruited for the experience	150
Table 8-10 Evaluation of the four PDA versions of the Iacchos Web site	152
Table 8-11 Evaluation of the four PDA versions of the Maporama Web site	153
Table 8-12 p-values for Wilcoxon Signed-Ranked test (H0: Global score on perceived usability is equal)	154
Table 8-13 p-values for Wilcoxon Signed-Ranked test (H0 : Global score on task completion time is equal).....	154

Chapter 1 Introduction

1.1 Motivation: the Challenge of Developing User Interfaces for Multiple Platforms

Computer-based information systems are an essential part of modern organizations. Users of these information systems often have to deal with a variety of platforms, mobile and fixed, from which they expect to have access to the same data and functionalities.

Computing platforms, that we will define as “any combination of hardware and software components on which the user interface will run”, can be very different: the devices, ranging through desktops, laptops, PDA’s and mobile phones, may differ in screen size, resolution or number of colours supported; the graphical toolkits can be different; the input devices can include a pointing device (mouse, stylus) or none, a full keyboard or a simple phone keypad, etc. New devices appear on the market at a very high rate and, although it is difficult to predict whether one particular material will gain popularity or not, the use of mobile devices and wireless technologies is unlikely to decrease in the next years.

Designing *Multiplatform User Interfaces*, i.e. interactive systems that provide access to information and services using different platforms [Seff04], is a difficult and time consuming task. Developers and researchers agree that the current methods, tools and languages do not entirely satisfy the challenges issued by multiplatform development. The development of Multiplatform User Interfaces (hereafter MUIs) suffers from the following limitations:

- *Lack of knowledge and experience*: developing a user interface that is tailored to a given platform requires knowledge and skills about the programming languages supported on the target platform, about the device capabilities, about the usability guidelines for this platform... but information on platform capabilities and usability guidelines is often unavailable or very scattered.
- *Lack of methodology*: if some methods for developing classical user interfaces have been proposed, for example MUSE [Lim94], DIANE [Bart88] or TRIDENT [Boda95], methods for building several versions of a user interface for multiple platforms at the same time are almost non-existent.

1. Introduction

- *Lack of tool support*: few tools are specifically dedicated to the design or coding of MUIs, whether we consider interface builders, prototyping tools, model-based generation tools, or user interface development environments in general.

These three major deficiencies entail further problems:

- *High development and maintenance costs*: user interface have always represented a significant fraction of the software development and maintenance effort, even for traditional applications designed with a single target platform in mind. The diversity of platforms, added to the ever increasing complexity of information systems, has still increased development costs.
- *Lack of consistency*: consistency is a basic principle of user interface design. In multiplatform systems, users expect to be able to rely on their experience of a given version of the system when using the same service on another platform. However, development is often performed by several teams (each platform requiring specific skills and experience), at different times (depending on the evolution of the customer's requirements). Consistency is thus seldom achieved.
- *Lack of adaptation / bad usability*: user interfaces must be usable, thus adapted to the platform on which they run. Adaptation to the target platform is often neglected, due to high development costs and, should the problem be taken into account, existing techniques sometimes provide results of poor quality, for instance when the adaptation is performed automatically via reauthoring systems.
- *Lack of reusability*: developers lack techniques that would allow them to reuse a user interface's components, logical structure and design on several platforms.
- *Lack of techniques combining genericity and flexibility*: developers lack techniques that would allow them to specify a user interface at an abstract, generic level, suitable for several platforms and contexts, while providing flexible, configurable adaptation to the specific target platforms.

1.2 Thesis

1.2.1 Thesis statement

We argue that developing consistent, usable and adapted user interfaces for multiple platforms simultaneously is a task that would benefit from

1. Introduction

- An *integrated approach* where all stages in the software life cycle are covered, from early requirements until prototyping and coding and where the same team or developer can work on several versions;
- An approach that *considers the entire multiplatform system at the initial design stage* with a focus on the continuity of the user experience;
- A *computer assisted approach* which automates some repetitive tasks while offering a good level of control to the designer;
- A *repository of design knowledge* which gathers observed practices and discusses their effects on usability.

Therefore, we will defend the following thesis:

The design and development of multiplatform user interfaces benefits from a *semi-automatic, model-based, transformational* approach which applies transformation rules to a *source model*, conceived for the least constrained platform, in order to produce one or several *target models*, adapted to *more constrained platforms*.

The concepts introduced above are reviewed and defined in the next section.

1.2.2 Definitions / motivations

1.2.2.a Semi-automatic approach

In order to conciliate computer-support and human control, we adopt a semi-automatic approach where:

- (1) Transformation rules are manually selected and parameterized by the designer, with a possibility to modify this configuration at any time.
- (2) Transformation rules are then automatically applied to reduce the design effort.

1.2.2.b Model-based approach

Model-based tools have been investigated since the late 1980's. The goal of these tools is to allow the designer to specify the user interface at a level that is independent from the implementation. The specification is usually shared between a set of components, called models, each model representing a facet of the interface characteristics. The number and type of these models is different from one approach to another, therefore the first part of this text will be dedicated to a precise description of concepts relevant to our approach.

The model-based approach has been the target of some major criticisms [Myer00] [Szek96] [Puer96]. The main shortcomings commonly cited are:

1. Introduction

- (1) *High threshold*: the programmers need to learn a new language in order to express the specifications of the UI.
- (2) *Low ceiling*: each of the model-based systems has strict limitations on the kind of UI they can produce and the generated UI are generally not as good as those that could be created with conventional techniques.
- (3) *Unpredictability*: it is difficult to understand and control how the specifications are connected with the final UI. Therefore, the results may be unpredictable.
- (4) *Lack of propagation of modifications*: changes made to one model or to the final UI are generally not propagated to the other levels of the specification.
- (5) *System dependent and private models*: a lot of models are strongly tied to their associated model-based system and can not be exported. Furthermore, some model specifications are neither publicly available, neither obtainable via a license.

Most of these problems can be addressed:

- (1) *High threshold*: most models can be built graphically in a design environment, which prevents users from learning the specification language. Even if the designers have to learn the specification language, the automation of a portion of the development should reduce the development efforts.
- (2) *Low ceiling*: we believe that this criticism only holds for a specific kind of model-based generation tool, which generates the user interface starting from very high level models (task model and/or domain model).
- (3) *Unpredictability*: our approach relies on an explicit set of rules, fully documented and accessible. It offers to the designer a full control on the selection of those rules. The results of the application of a rule may be previewed.
- (4) *Lack of propagation of modifications*: although the problem of the impact of a modification made on a given model on the other models remains a tricky one, we will attempt to determine the side effects on the other models entailed by the application of a given rule.
- (5) *System dependent and private models*: we will make use of a user interface description language publicly and freely available.

Model-based interfaces have also recognized advantages [Puer97]:

- (1) Advantages in terms of *methodology*:
 - It is a widely accepted software engineering principle to start a software development cycle with a specification stage [Ghez01].

1. Introduction

- The model-based approach supports a user-centred and UI-centred development life cycle: it lets designers work with tasks, users and domain concepts instead of thinking in engineering terms.
- (2) Advantages in terms of *reusability*:
- In a multiplatform context, model-based tools can provide automatic portability across the different devices.
 - The availability of a complete description of the interface in a declarative form allows the reuse of some interface components.
- (3) Advantages in terms of *consistency*:
- This approach ensures some form of consistency between the early phases of the development cycle (requirements analysis, specification) and the final product.
 - In a multiplatform context, it also guarantees a minimal consistency between the UI generated for different target platforms. This is not always possible when using traditional techniques where the development of each version of the UI is likely to be performed separately.

1.2.2.c Transformational approach

Besides being model-based, our approach is also *transformational* i.e. based on a catalogue of transformation rules. Similarly to the concept of schema transformation in database engineering [Hain02], we can define a transformation between source model $\mathcal{M}$ and target model $\mathcal{M}'$ as an operator which replaces a construct C in $\mathcal{M}$ by a construct C' in $\mathcal{M}'$, or inserts a new construct into $\mathcal{M}'$, or removes an existing construct, while preserving a set $\mathcal{P}$ of properties of $\mathcal{M}$.

The set $\mathcal{P}$ of properties we want to preserve includes:

- The *usability* of the user interface;
- The *cross-platform consistency* of the whole information system, i.e. the consistency between the various versions of the user interface.

1.2.2.d Source model/target model

In our approach, the source model $\mathcal{M}$ is a UI model of a given type t (e.g., presentation, task...) designed for a given platform (typically: a desktop) and the target model $\mathcal{M}'$ is a UI model of the same type t , targeted to more constrained platforms (for example, PDAs or smartphones). Centring the design process on a source interface designed for the least constrained platform is sometimes referred to in the literature as a “single authoring” method [Wong02] [Ding06].

Typically, the source platform, when it is a desktop, will also be the most frequently used platform and the first platform to be considered when starting the development of a multiplatform system.

1. Introduction

1.2.2.e Constraints of a platform

Our basic idea is that transformation rules will adapt a source interface to more constrained platforms (see illustration on Figure 1-1). The phrase *more constrained platforms* covers several parameters:

- *Decrease of the screen resolution and size*: this parameter has a strong influence on the structure and presentation of the user interface. Sometimes, even with a similar screen size, the available screen area may be more constrained when a part of the display is used for other purposes (e.g; virtual keyboard).
- *Increase of the minimal size of graphical objects and the minimal spacing between them*: on some platforms, the objects included in the interface are to be larger or more distant (e.g; touch screen interfaces).
- *Decrease of the number of available widgets*: not all toolkits are available on every platform. Furthermore, some platforms have reduced versions of the toolkit or simplified versions of the mark-up language.
- *Decrease of the usability of available widgets*: the usability of a given widget may vary from one platform to another, for example because of the absence of a keyboard on some platforms.

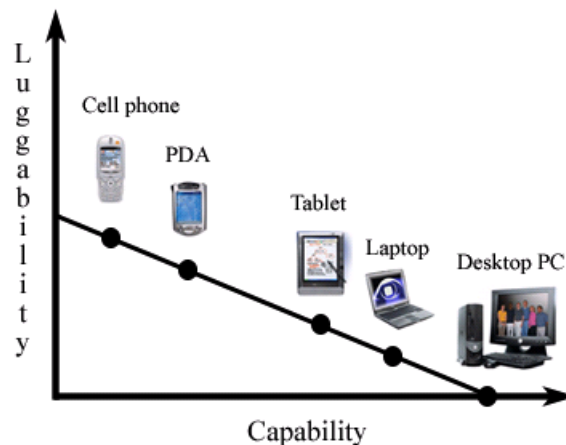


Figure 1-1 Constraints of some current hardware platforms (from [Pier04])

Other parameters such as the network capacity, the support of frames, images or colours, the presence and the type of pointing devices and keyboard, the storage capacity or the CPU speed can also be taken into account, but we will focus on the four parameters listed above for two reasons:

- (1) They raise interesting methodological problems because they force the modification of the UI at a high abstraction level and cannot be addressed only by mechanical solutions as transcoding, image cropping, or the use of multiplatform virtual toolkits.

- (2) Some technical characteristics of platforms such as the RAM memory, network bandwidth or disk space available are likely to be improved in the near future, while the display size will probably remain almost constant (see [Menk03b]).

Deciding which platform is the most constrained is not always easy:

- Some platform characteristics may be quite similar: is the difference between a screen size of 128x128 and a screen size of 128x160 significant enough to consider that we have distinct constraints? What should be the threshold?
- One platform could have a better performance with respect to one criterion and a less good performance with respect to another criterion. In such situation, it is difficult to decide which platform is the most constrained.

When it is doubtful whether a given platform should be considered as a source or a target platform, for example because a platform performs better with respect to one criterion and worse with respect to another one, our transformation rules should not be used, or should be used very cautiously. For example, if we need to design a multiplatform system that has to run on a PDA and on an interactive kiosk with a similar screen resolution, no platform is actually “less constrained”. Thus, applying our approach to adapt the PDA UI to the interactive kiosk or conversely does not seem a good idea. A possible design strategy for such a multiplatform system could consist in starting designing an artificial source UI conceived for a desktop, and to apply our transformation rules to that source interface.

1.2.3 Graceful degradation

As our transformation rules take as input an interface tailored for a given platform and produce smaller interfaces as output, we call the transformation process degradation.

As we want to produce highly usable interfaces adapted to the specific platforms while preserving the consistency between the versions, we qualify this degradation as a “graceful” one.

The phrase “graceful degradation” was born in the field of safety-critical systems such as aerospace. It denotes the ability of a system to continue providing service in proportion to the level of surviving hardware. The idea is that every system can malfunction at some point, either because a component has failed or is missing, or because the system experiences conditions it was not designed for (overload...) Under these bad conditions, the system is expected not to crash completely, but to continue providing its basic functions with a lower performance than would be expected normally. Our use of the term is slightly different since we do not consider the adaptation of services but the adaptation of the user interface presentation and since the adaptation is not triggered by a component failure but by the target platform capabilities.

We define *design by graceful degradation* as a *model-based, transformational* method to support the development of user interfaces for multiplatform systems. This method, applied at design time by the designer, is based on a set of transformation rules called *graceful*

1. Introduction

degradation rules (hereafter GD rules). These rules transform a source interface in order to produce specific interfaces targeted to *more constrained platforms*. The method is meant to respect a trade-off between, on the one hand, the usability of the user interface and, on the other hand, the consistency between the versions of the user interface. Figure 1-2 is a visual synthesis of our approach.

Even is the term may seem ambiguous, graceful degradation does not refer to a degradation of some quality of the user interface, but to a degradation of the potentialities of the target platform in terms of the constraints enumerated above (without other value judgment on the utility and quality of these small platforms). Therefore, this thesis will not contain a description of the ergonomic properties of a source interface that would be degraded on a target platform: when a satisfying design is produced, the usability of a target user interface could even be higher than the usability of the source user interface, notably in terms of guidance.

The “gracefulness” of the degradation refers to the trade-off mentioned above: our transformational approach should preserve at the same time the usability of the UI on the target platform and the cross-platform consistency.

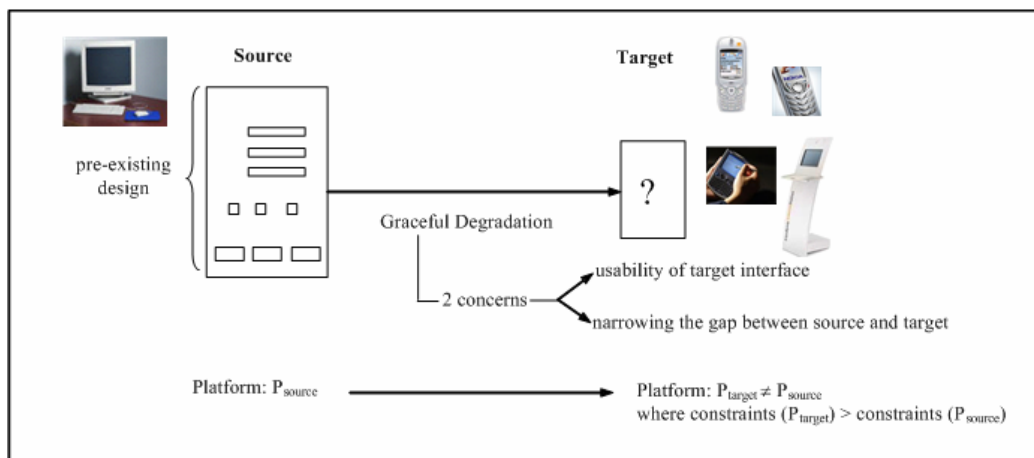


Figure 1-2 Graceful Degradation in a nutshell

1.2.4 Focus

- Our focus is information systems (IS), defined as “a set of interrelated components that collect (or retrieve), process, store, and distribute information” [Laud06]. Typically, this information is stored in databases. The importance of these IS is vital in nearly all companies and organizations. Different types of IS can be distinguished following the operational level they serve in the organization (strategic, management or operational level) and following their major functional area (sales and marketing, manufacturing and production, finance and accounting, and human resources). Typical examples of information systems (or

subsystems) are a payroll system, a registration system or a sales order information system. Examples of applications outside the category of IS are entertainment applications, embedded systems or control systems.

- The scope of this work is limited to Graphical User Interfaces (hereafter GUI), which are the standard interface to most information systems, are familiar to the majority of users, and are available on almost every platform. Hence, we do not consider nonvisual, multimodal or 3D user interfaces.
- The target audience of this thesis is, on the one hand, the HCI research community and, on the other hand, the professionals involved in the design and development of multiplatform user interfaces. In the remainder of this manuscript, we refer to these actors as “designers”.

1.3 Reading Map

The remainder of this thesis is structured as follows:

Chapter 2 presents current methods and tools which support the development of user interfaces for multiplatform systems (MUI). These approaches are classified using a theoretical framework called the CAMELEON framework, or Unified Reference framework, described here and used extensively in the remainder of the manuscript. They are compared and assessed using a set of criteria. The same list of criteria will help us to evaluate the usefulness of the method and tools proposed in this thesis.

Chapter 3 defines the model constituents that will be used through the whole work and the language chosen to represent these concepts (the user interface description language UsiXML).

Chapter 4 presents effective knowledge for Graceful Degradation. In this chapter, we establish a typology of rules using the CAMELEON framework and we discuss their formalization, relying on the UsiXML notation introduced in Chapter 3.

Chapter 5 will show the implication of multilevel application of rules on the example of the splitting rule, which permits to paginate content.

Chapter 6 will present the tools developed to document GD rules and to support their application. The first tool is a knowledge base that gathers, structures and organizes the rules. The second tool is a prototype aimed to demonstrate how GD rules can be applied automatically on a UsiXML specification.

1. Introduction

Chapter 7 illustrates the Graceful Degradation approach on case studies. The case studies cover the two types of scenarios envisioned for the use of GD rules: manual adaptation (the ARTHUR case study) and semi-automatic adaptation using the tool described in Chapter 6.

Chapter 8 will contain elements of validation, both theoretical and practical. Theoretical validation consists in assessing the GD approach using the quality criteria identified in Chapter 2. Empirical validation was realized by conducting an exploratory study implying twelve end users and analysing their appreciation of user interfaces produced by graceful degradation, in contrast to other methods (ad-hoc development and direct migration).

Chapter 9 will conclude by summarizing our contributions and exploring some avenues for future work.

Chapter 2 State of the Art

This chapter starts by presenting a theoretical framework for model-based, context-sensitive user interfaces. This framework then serves as a reference for structuring our description of current methods and tools for developing user interfaces for multiple platforms. These tools and methods are then analysed and compared using a set of quality criteria.

2.1 A structuring theoretical framework

The Graceful Degradation approach, as a model-based approach, relies on a set of constituents called *models*. These models describe various aspects of the user interface considered as relevant. The number and type of models is different from one approach to the other but, generally, three layers¹ of models [Szek96] can be distinguished, depending on the abstraction level:

- (1) The highest level consists of the *task and domain models*.
 - A *task model* is a description of the tasks that a user will be able to achieve in interaction with the system. This model can be understood as a generic representation of the envisioned scenarios that were elicited during the requirements analysis. It has generally a hierarchical structure and additional constraints and information about the tasks (such as ordering, launching conditions, associated objects and functions) can be added.
 - In the model-based approach, a *domain model* represents the set of objects that will have an impact on the UI (objects storing data that will be displayed or that will be modified by the user, or objects with methods that can be called from the UI or that can modify some aspect of the UI).
- (2) The second level represents the interface in terms of low-level interface tasks (such as selecting an element from a set or consulting a list of values), presentation units (abstractions of windows) and information elements (the data to be shown).
- (3) The third level specifies how the presentation units and their content will be rendered. It represents the interface in terms of toolkit primitives such as windows, dialog boxes, buttons or check boxes and also specifies the layout of those elements.

¹ In this manuscript, the term “layer” merely refers to the abstraction level of a model and is unrelated to the concept of layer in networking protocols such as TCP-IP or OSI.

2. State of the Art

Models at level 3 are often known under the terms of *presentation model* for the static aspects and *dialog model* for the dynamic aspects i.e. the description of the reactions of the system to the user's actions on the presentation elements.

Other types of models can be used:

- A *user model* stores the characteristics of the different types of users or user groups (their preferences, capabilities, role...)
- An *application model* describes functions belonging to the application functional core that are associated with objects in the domain.
- A *platform model* is a description of the combination of hardware (input and output devices, network connectivity, memory ...) and software (OS, available toolkits ...) on which the UI will run.

This distribution of models into hierarchical levels has been formalized into what is known as the CAMELEON framework or Unified Reference Framework [Calv03]. The Unified Reference Framework is intended to support the development of context-sensitive user interfaces in a model-based approach. We will refer to this framework throughout this manuscript, notably because the mark-up language we rely on (UsiXML) has been structured in accordance with it.

The notion of context in the framework encompasses three components: the user, the platform and the physical environment. Since we are only concerned by variations of platforms, we will use the terms “context of use” and “platform” indiscriminately.

The framework describes models at four abstraction levels (Figure 2-1) from the task specification to the running interface:

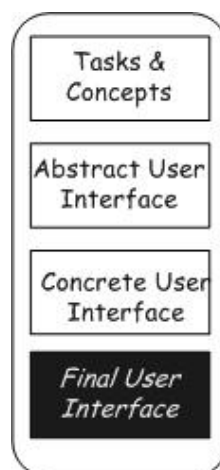


Figure 2-1 The four abstraction levels in the Unified Reference Framework

2. State of the Art

- (1) The *Tasks and Concepts* level describes the interactive system's specifications in terms of the user tasks to be carried out and the domain objects manipulated by these tasks.
- (2) The *Abstract User Interface* (AUI) 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,...) An interaction space is a grouping unit that supports the execution of a set of logically connected tasks.
- (3) The *Concrete User Interface* (CUI) is an expression of the UI in terms of “concrete interactors” (in fact, already an abstraction of actual widgets generally included in toolkits). The placement of these concrete interactors is also specified.
- (4) The *Final User Interface* (FUI) consists of source code, in any programming language or mark-up language (e.g. Java or HTML). It can then be interpreted or compiled. A given piece of code will not always be rendered on the same manner depending on the software environment (virtual machine, browser,...) For this reason, we consider two sublevels of the FUI: the source code and the running interface (Figure 2-2).

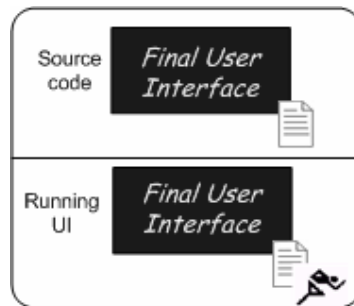


Figure 2-2 Two sublevels in the Final User Interface

Three types of relationships between these models have been defined (see Figure 2-3):

- (1) A relationship of *reification* links each level to the more concrete level just below.
- (2) A relationship of *abstraction* links each level to the more abstract level just above.
- (3) A relationship of *translation* between models at the same level of abstraction, but conceived for different contexts of use.

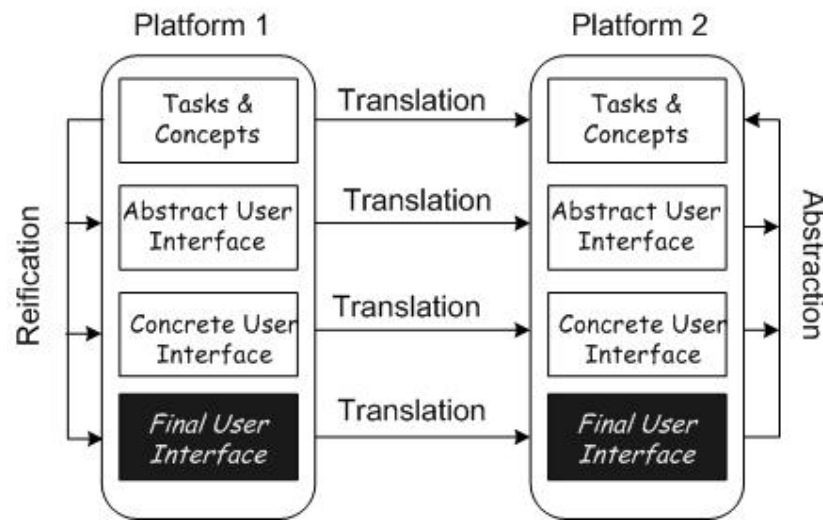


Figure 2-3 Relationships between components in the Unified Reference Framework

The Unified Reference Framework contains more than the four abstraction levels and three types of relationships. However, these notions are sufficient to describe and understand methods and tools currently used to develop user interfaces for multiple platforms at design time.

2.2 Approaches to the development of user interfaces for multiple platforms

Many techniques and tools have been used to develop UI for multiplatform systems. On the basis of the Unified Reference Framework, we can distinguish between five approaches:

1. The first approach or *traditional development approach* consists in coding a separate UI for each target platform.
2. The second approach or *unique portable code approach* consists in coding a unique UI able to run on the different platforms.
3. The third approach or *transcoding approach* takes as input the source UI's code and generates UI's for the target platform(s).
4. The fourth approach or *multireification approach* takes as input the source UI's specification and generates UI's for the target platform(s).
5. The last approach or *abstraction-reification approach* takes as input the code of one source UI, generates an intermediate representation at a higher abstraction level and produces as output the code of a UI ready to run on the target platform(s).

2.2.1 The traditional development approach

The more obvious method and probably the most commonly used method when producing UIs for multiplatform systems consists in developing a specific interface for each target platform (Figure 2-4).

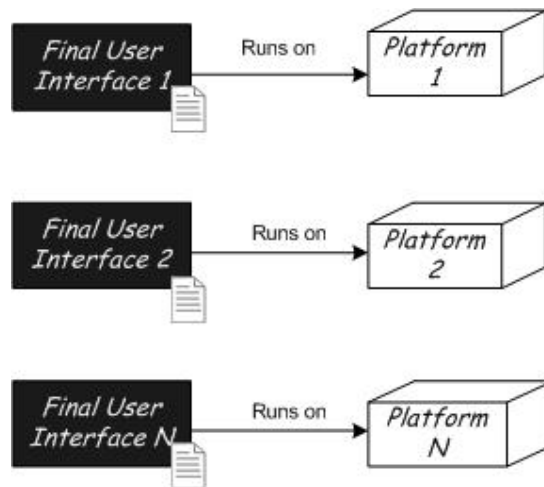


Figure 2-4 Traditional development of multiplatform UI

The traditional approach makes use of tools such as:

- Programming environments.
- User interface builders such as Visual C++, Visual Basic, Borland JBuilder, Glade, etc.
- Prototyping tools: [Szek94] drawing editors, sometimes enhanced with functions that permit simulating the interface's behaviour, such as Astound, Apple's Hypercard, MacroMind Director, Microsoft Visio; sketching tools (e.g.; DENIM [Lin02]); some of them enhanced with sketch recognition, what gives them functionalities similar to UI builders, like SILK [Land95] or JavaSketchIt [Caet02]. A state of the art report on sketching tools can be found in [Coye04]².

This approach:

- Often requires programming skills in different languages.
- Often requires knowledge of different toolkits.
- Does not factor out common aspects (features common to all versions) from specific ones (features specific to one version, to one platform).

² Some of the tools listed here are used for Web design (e.g; DENIM, SILK) or generate portable code relying on a virtual toolkit (e.g; JBuilder, JavaSketchIt) and could have been introduced in the next section (2.2.2). What we want to emphasize here is that a number of separate versions of the user interface are developed, whatever format is used for coding the final user interface.

2. State of the Art

- Does not guarantee any consistency between the different target specific UIs: this is left under the responsibility of the designers.

Figure 2-5 shows an example of separate development for two versions of the Web site of American Airlines (<http://www.aa.com>), the PDA version being developed by the AvantGo company: the PDA version proposes only a subset of the functionalities offered by the desktop version, only the navigation menu is visible on the home page while much more information was displayed on the desktop, the logo image is cropped,...



Figure 2-5 The web site of American Airlines: an example of traditional development of multiplatform UI

2.2.2 The unique portable code approach

Two techniques are widely used in order to achieve code portability: virtual toolkits (Figure 2-6) and generic clients (Figure 2-7).

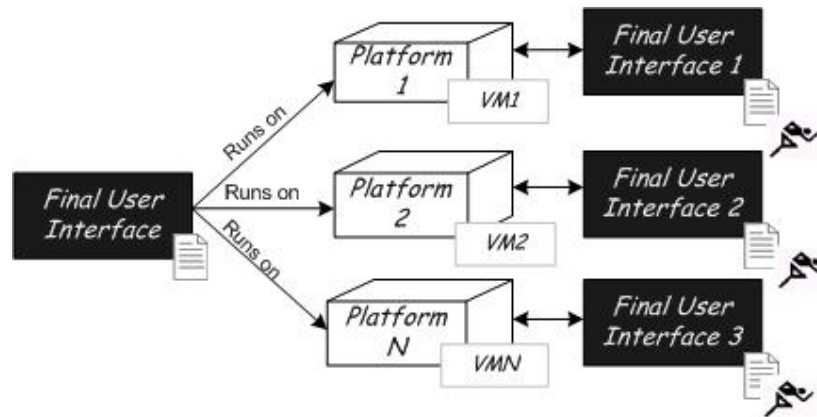


Figure 2-6 Development of portable UI using a virtual toolkit

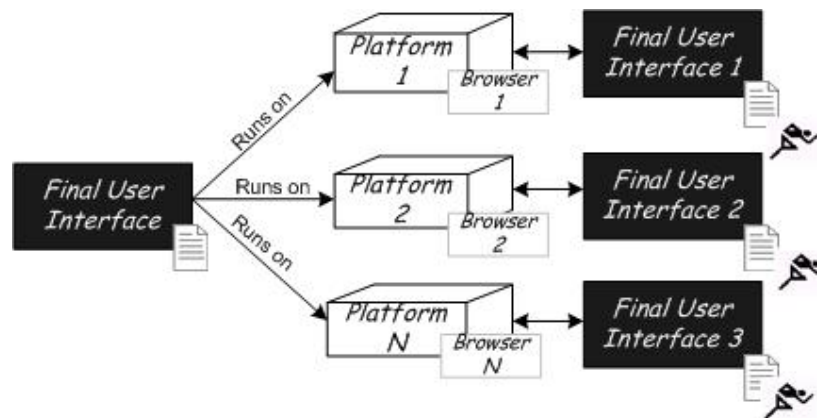


Figure 2-7 Development of portable UI using browsers

2.2.2.a Development of one single portable UI using a virtual toolkit

Virtual toolkits [Myer03] provide virtual widgets that can be mapped into the widgets of each toolkit. The interface code will run without change on different platforms and still provide a platform specific look-and-feel. Some virtual toolkits like Java AWT use the actual toolkits on the host machine while others like Galaxy³, Amulet [Myer97] or Java

³ Visix Software Inc. "Galaxy application environment". Now owned by Ambiência Information Systems.

2. State of the Art

Swing provide their own libraries that mimic the host platform look-and-feel. A mix of both techniques can be used such as in version 8.0 of Tk [Oust94], where many widgets are implemented with native platform widgets while others only emulate native look and feel. These virtual toolkits are considered to provide a limited form of adaptation (look-and-feel adaptation). Java Swing even permits setting explicitly the look-and-feel, as shown on Figure 2-8. Other multiplatform toolkits such as SUIT [Paus92], Garnet [Myer90] or the earlier versions of Tk use the same look-and-feel on all platforms and do not provide any adaptation perceivable to the user. Virtual toolkits like Tk and Swing where a layout manager dynamically calculates the placement of the objects also provide some adaptation of the UI presentation to the window's size. For example, Figure 2-9 shows how a layout manager in Swing adapts the placement of the components in reaction to the resizing of a container.

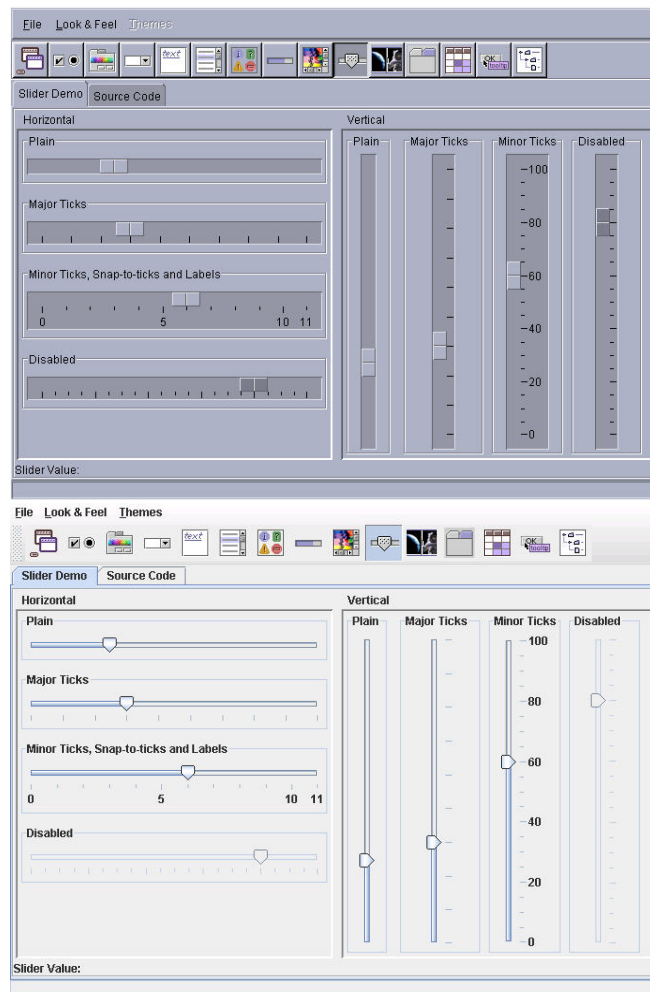


Figure 2-8 Adaptation of the UI's look-and-feel using Java Swing

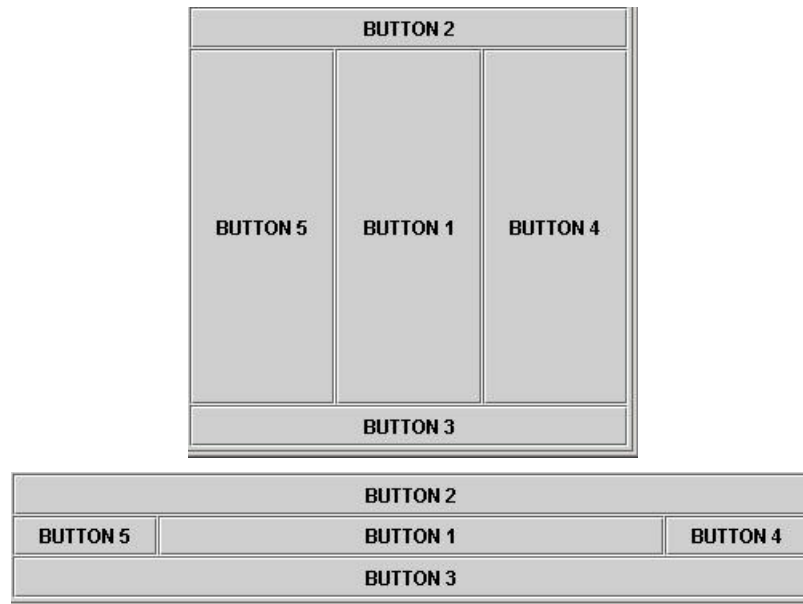


Figure 2-9 Automatic adaptation of a layout to the size of a container using a layout manager in Java Swing

More flexible toolkits have been proposed. Crease et al. [Crea00] have developed an extension to the Java Swing toolkit, with widgets able to adapt their presentation and handle different modalities at input and output, depending on user preferences or resource availability. Similarly, [Calv04] [Calv05] introduce a new kind of widget, the comet, which can possess different presentations and can be capable of self-adaptation. Comets can correspond to classical widgets (e.g. a button) or to custom widgets that can be much more sophisticated. The adaptation mechanisms provided by both toolkits go far beyond mere look-and-feel adaptation. For example, a polymorphic comet may own several descriptions at the Abstract User Interface level (the number of interactive spaces used to render the comet is different), and at the Concrete User Interface level (the comet has several alternate presentations). However, the comet mechanism only permits switching between pre-calculated alternate designs stored in the comet.

Additional techniques can be used together with virtual toolkits, in order to provide more than widget-level adaptation. For instance, [Ding06] proposes an authoring tool that transforms a Java user interface designed for a large platform to a more constrained platform, thanks to a transformation engine able to apply adaptation strategies such as scaling, widget substitution and splitting.

2.2.2.b Development of one single portable UI using generic clients

The second form of the “unique portable code approach” consists in specifying the user interface using a mark-up language, such as HTML. This code is rendered by a client program, called browser, available for a large range of platforms. As the mark-up languages were first designed for document display, they generally have to be combined with scripting languages such as JavaScript in order to control the UI behaviour.

A variant of this approach has been demonstrated by Wingman, a browser for the Palm Pilot PDA [Fox98]. Wingman relies on a three-tier client-proxy-server architecture. The proxy carries out most of the tasks usually devoted to the client, such as HTML parsing or tag-to-font mapping. In addition, it performs transformations such as image scaling and conversion to the Palm Pilot native format or image dithering.

HTML was conceived to be platform independent. However, the different browsers available often implement their own version of standards, which jeopardizes the task of authoring web sites for multiple platforms. This problem is topical again because of the important number of different browsers for small devices: Pocket IE (a version of Microsoft's browser for small devices), Opera (which has an established position in the market of small-screen devices), Minimo (a version of Mozilla), AvantGo, Blazer, Palm Web browser, ... Furthermore, one cannot expect the same Web page to be directly usable on such different platforms as a desktop, a PDA or a phone, due to differences in screen size notably. For this reason, some small-screen browsers such as Opera reformat Web sites to fit inside the screen width, which is beneficial in terms of adaptation to the device's capabilities but reduces the author's control on his/her pages. Such browser-side adaptation is made easier when Web sites authors respect design principles such as those issued by the W3C device independence working group (for example, the note on Authoring Techniques for Device Independence [W3C04b]).

Another problem with the generic client approach is the number of mark-up languages in use: besides the different versions of HTML, we could quote formats such as XML, WML, cHTML and iMode, Palm Web Clipping, ... One solution to this problem consists in developing one common XML document or HTML page and XSLT transformations to the other formats, which still requires a lot of authoring.

Good web development practices involve the separation of content, expressed as HTML or XML, and presentation, described in CSS or XSL stylesheets. Some adaptation of Web pages to the client's device can be provided using platform specific stylesheets. For example, parts of the content can be shrunk or hidden, or table items can be presented in sequence. The technique requires a lot of authoring. For this reason, research has been conducted on "multi-level" stylesheets [Dees04] factorizing the styling attributes common to several devices, in order to avoid writing a separate stylesheet for each target platform. Another limitation of stylesheets is that they do not allow performing transformations such as paging or changing form controls. Extensions to HTML have been explored that could support such adaptations. For example RIML [Zieg04] permits declaring parts of the code as "splittable" or not in order to achieve automatic pagination. Another example is the W3C recommendation XForms [W3C03]. XForms defines device-independent form controls (e.g. "input" or "choices") that are further rendered as appropriate on the target devices. However, if some small-screen browsers such as Opera already have support for CSS, it is doubtful whether they will ever accept new standards such as XForms.

2.2.3 The transcoding approach

Transcoding tools perform a transformation of a final user interface conceived for a given platform into another final user interface conceived for a distinct platform, without generating explicitly an abstract description of the UI (Figure 2-10).



Figure 2-10 Development of multiplatform UI using transcoding tools

Transcoding tools may be classified into two categories [Menk03]⁴:

1. **Identity configuration.** The identity configuration (Figure 2-11) corresponds to approaches where input and output are in the same standard format. A standard format is a widespread format that consumer devices can display without modification, such as HTML or WML. An example of this configuration is reauthoring an HTML document into another HTML document.

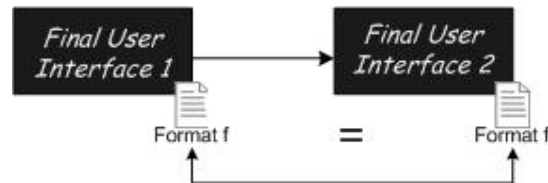


Figure 2-11 Transcoding tools in the identity configuration

2. **Direct configuration.** The direct configuration (Figure 2-12) corresponds to transcoding approaches where the input and output formats are standard formats. This technique is especially used for adapting Web content to other mark-up languages (e.g. HTML to WML, to TinyHTML for PalmPilots, to CHTML for iMode or to Voice XML).

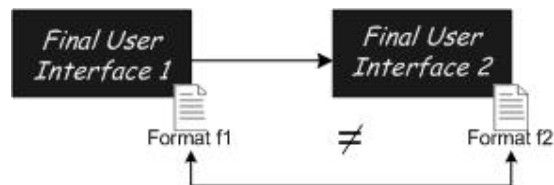


Figure 2-12 Transcoding tools in the direct configuration

⁴ Menkhaus and Fischmeister's classification contains two additional categories. "Hybrid configuration" includes tools such as MUSA or UIML, and roughly corresponds to the multireification approach described in next section. "Indirect configuration" corresponds to the abstraction-reification approach described below.

2.2.3.a Digestor

Digestor [Bick97] is a tool in the identity category. It has been implemented as a proxy server. It dynamically adapts Web pages to the screen size specified by the user using a best-first algorithm and a set of page transformations: redistribution of text between several pages (parts of the text are elided from the document and hyperlinks linking to the elided content are created), image and font reduction...

2.2.3.b Artail and Raydan's reauthoring approach

[Arta05] describes another approach in the identity category. The server-based tool proposed automatically reauthors Web pages for rendering on small-screen devices. The tool is composed of a "preprocessor" and a "handler". The preprocessor automatically generates CSS files for each page on the web server. These CSS files include a section intended for rendering elements on desktops, another section meant for handheld devices, as well as a common section. The handler is called when the user requests a given page. Relying on the characteristics of the client's device found in the http header, the handler passes the page through several algorithms, which operate text transformation (first sentence elision), table conversions and image size reduction before sending the transformed page and the related CSS style sheet to the client. The state of the art section of Artail and Raydan's paper discusses several similar techniques for transforming Web pages.

2.2.3.c IMBWebsphere

IMBWebsphere⁵ is a server-based tool in the direct category. It performs real-time transcoding of HTML to multiple mark-up languages (Palm OS HTML, Voice XML, WML, iMode). It provides adaptation such as image reformatting and rescaling, filtering content, fragmenting data. Transformations can be customized in several ways.

2.2.4 The multireification approach

In the multireification approach, developers produce a specification of the user interface instead of code. This specification can take different forms:

1. Detailed description(s) of the concrete user interface's widgets, layout and behaviour (Concrete User Interface level in the CAMELEON framework) e.g. LiquidUI [Phan00], AUIT [Grun03], SEESCOA [Luyt03], WebML [Ceri00]
2. More abstract description(s) expressed in terms of toolkit independent widgets, or even modality independent widgets (Abstract User Interface level) e.g. TIDE [Ali03], the AUIML toolkit [Clar00]
3. High level abstractions expressed in terms of tasks and domain concepts (Tasks&Concepts level) e.g. TIDE [Ali03], ArtStudio [Thev01], and TERESA [Pate02]. Most often, a single representation is provided at the tasks and concepts

⁵ WebSphere Transcoding Publisher http://www-306.ibm.com/software/pervasive/transcoding_publisher

level, but annotations indicating what should be rendered on which platform are added to the task model or the domain model.

The specification is then transformed into code, often producing intermediate representations at lower abstraction levels. Figure 2-13 shows multiplatform reification at distinct abstraction levels.

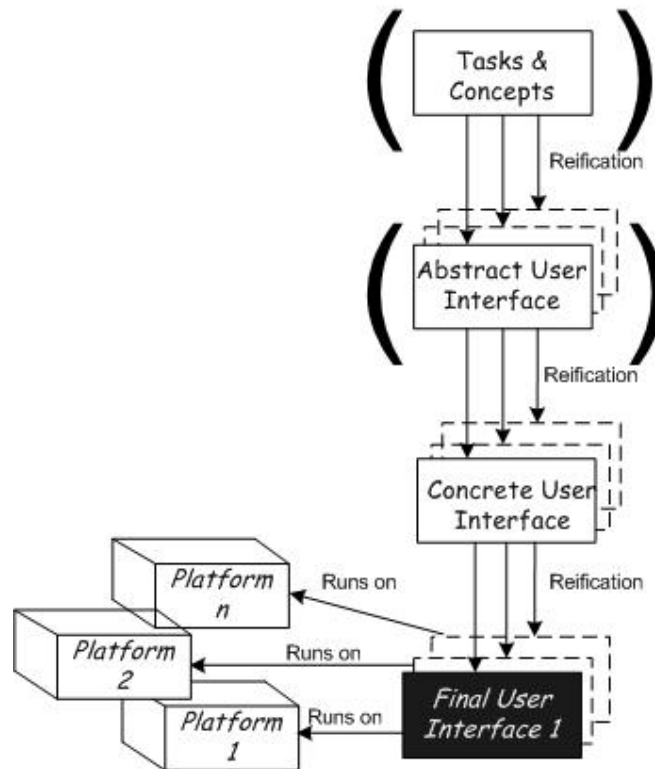


Figure 2-13 Development of multiplatform UI in the multireification approach

We will not list the whole range of user interface description languages (UIDLs) and related tools, but only a few representative approaches. UIDLs are numerous and more complete reviews can be found elsewhere [Souc03].

2.2.4.a LiquidUI

Harmonia's LiquidUI⁶, an authoring tool for the UIML language [Phan00] is a good example in the first category. UIML or "User Interface Mark-up Language" is a XML-based language that allows the description of device-independent user interfaces. An UIML interface specification includes three components:

- The logic component, that provides a way to communicate with the application independently from the protocols, method names or server location,

⁶ Harmonia: <http://www.harmonia.com>

- The presentation component, that provides a way to render the UI independently from the actual widgets and their properties and event handling,
- The interface component that describes the interaction between user and application in a platform independent way.

The interface component is subdivided into four subcomponents: the structure, the style, the content and the behaviour. The interface is then specified using a platform specific vocabulary. So, there is a separate vocabulary for AWT, Swing, WML, HTML... Each platform-specific specification can then be rendered using LiquidUI. The LiquidUI suite includes a Java renderer, an HTML renderer, a VoiceXML renderer and a WML renderer. Other renderers for UIML have been developed outside Harmonia: renderers to C++, QT, Visual Basic...

2.2.4.b TIDE

Another tool based on UIML is the Transformation-based Integrated Development Environment (TIDE). TIDE [Ali03] uses four abstraction levels: a task model, a description of the UI using UIML with a generic vocabulary that is common for a device family (e.g. desktop or WML), a UIML description with a platform specific vocabulary and the final interface. The tool highlights the mappings between the abstraction levels (the task model is not included yet), letting the designer controlling them.

2.2.4.c The Abstract User Interface Language Toolkit

IBM's Abstract User Interface Mark-up Language (AUIML) Toolkit [Clar00] relies on the same principle as LiquidUI. The AUIML specification is built using a graphical editor. Renderers for Java Swing and HTML are available. The AUIML vocabulary is platform independent. Although little information on AUIML is publicly available, AUIML's building blocks seem to be elements describing the data type, the function (action, choice...) or the grouping, which places the AUIML Toolkit in the category of multireification tools working on the Abstract UI level.

2.2.4.d MONA

The MONA research project [Simo05] has investigated a tool supporting the development of multimodal user interfaces for mobile devices. The reification process starts from a single presentation model, or "root interface", written in an ad-hoc user interface description language. The core of this language is a set of platform- and modality-independent widgets, associated to "content sets" which are collections of multiple alternative contents for different modalities. Widgets are associated to properties (style, priority...) and behaviours, described declaratively. The layout is specified by nested groups of widgets and alignment constraints between the widgets of each group, generating an adaptive flow-layout. Groups of widgets semantically linked can be declared. This tool is meant to stay experimental, as the objective of the authors is migration to existing Web standards such as XFORM or CSS, or at least to solutions built around standards.

2.2.4.e SEESCOA XML and Dygimes

SEESCOA XML was first designed as a serialization language aimed to describe at run-time an existing interface using the reflection mechanism of Java [Luyt03]. The XML description provides an abstraction of the user interface using Abstract Interaction Objects. The XML document can then be transported to another system (for example, via the Internet). Once arrived on the target system the XML document can be converted to a working user interface. For every target system, an XSLT document is defined which maps the AIO's defined in the abstract user interface description to CIO's for that particular system. Transformation rules used with SEESCOA XML were thus platform specific rules (one XSLT document per target) and allowed only changes in the mappings between CIO's and AIO's.

The work on SEESCOA XML has been later included into the Dygimes framework [Luyt04]. Dygimes is a framework for generating multi-device user interfaces at runtime. It takes as input a CTT task model, annotated with UI "building blocks" i.e., fragments of presentation, expressed in terms of AIO's, attached to the tasks of the task model. The sets of tasks that will be presented together are then calculated using an "Enable Task Sets" (ETS) algorithm, similar to the algorithm in [Pate00]. Heuristics also permit grouping several ETS together. A dialog model, under the form of a State-Transition Network, is automatically generated from the CTT task model [Luyt03b]. The user interface is then generated, using the ETS to determine the content of the windows and the STN to generate the navigation between those windows. Each window corresponds to the merging of the building blocks attached to the tasks of the ETS. At this level, the UI is expressed in SEESCOA XML. The transformation of this XML description into a final interface is supported by the rendering engine UiBuilder. UiBuilder is part of the Dygimes system, and is able to render a SEESCOA XML document in Java AWT, Java Swing, Java kAWT (a subset of AWT for Palm OS) and HTML.

2.2.4.f Adaptable User Interface Technology

AUIT [Grun03] describes the UI at the Concrete UI level as a grid layout containing screen elements (widgets, images, etc.) and other grids. The grid structure can be used to divide a large interface into multiple, smaller interfaces for display on small-screen devices. The AUIT tags are device mark-up language independent and are transformed at run-time into HTML or WML mark-up. The layout is determined depending on the device, user and user task context.

2.2.4.g WebML

The Web Modeling Language (WebML) [Ceri00] is a specification language for Web sites. It is a mark-up language, but each concept is associated with a graphic notation. WebML involves four distinct descriptions:

- A UML class diagram-like *domain model*, describing relevant entities and relationships,
- A *hypertext model*, which consists of two sub-models: a *composition model*, which specifies which pages compose the hypertext, and which content units make up a

page. Different types of content unit exist: some units display information on a single object or collection, while others are used to browse a set of objects (searching, scrolling,...) and a *navigation model* which expresses how pages and content units are linked,

- A *presentation* model expresses the layout and graphic appearance of pages, independently of the target platform's language, using stylesheets. These style sheets can be generic (they apply to all pages) or specific (they apply only to units describing specific concepts),
- A *user model*, used for personalization purposes.

A code generator processes WebML specifications and translates them into some concrete mark-up language (e.g. HTML or WML). At the same time, it maps the abstract references to content units into concrete data retrieval instructions in some server-side scripting language (e.g., JSP or ASP).

2.2.4.h ArtStudio

ArtStudio ([Calv01] and [Thev01]) is a generation tool that makes use of various descriptions: a task model, a concept model, a platform model, an interactor model, an abstract interface model and a concrete interface model. The task model in ArtStudio is a CTT structure with additional parameters, such as references to the functions that will be launched before or after the execution of the task or references to the concepts involved in the task. The concept model is an UML specification. The abstract user interface (AUI) represents the first step of the reification process from the models at a high abstraction level (tasks and concepts) to the final running interface. An ArtStudio AUI consists of a set of workspaces where each elementary workspace corresponds to a leaf task and each compound workspace corresponds to a task at a higher level in the task hierarchy. The static part of the AUI is completed by a navigation scheme based on the logical and temporal relationships between the tasks. The navigation scheme determines which transitions between workspaces will be allowed by the UI. The generation of the concrete user interface (CIO) takes as input the AUI, the platform model and the interactor model and consists in a reification of the workspaces into windows or other container widgets, of the concepts into interactors and of the navigation scheme into navigation interactors.

2.2.4.i TERESA

TERESA ([Pate02] and [Mori03]) generates UI for multiple devices from a single CTT task model. For each task, the objects that will be handled as well as the suitable target are mentioned. It is also possible to specify the suitable target at the object level. The task model is then filtered to produce a platform specific task model. This task model is further transformed into an Abstract User Interface (AUI), i.e. a set of abstract presentations and their dynamic behaviour. This transformation uses the Enable Task Sets (ETS) algorithm [Pate00], which produces the set of tasks enabled at the same time. These ETSs can be modified using various heuristics. Once the tasks have been

distributed among the presentations, each presentation is defined in terms of abstract interaction objects linked by operators (grouping, ordering, hierarchy, relation). In the last step of the generation process, each abstract interactor is mapped to a concrete interactor available on the target platform and each operator receives a concrete representation.

2.2.4.j SGUI

SGUI [Chu04] is a user interface development tool for cross-platform user interfaces. It also supports migration between platforms. The tool is composed of a library of abstract widgets and events and of a transformation manager. The transformation manager uses as input a platform independent presentation model. This model is a tree structure combining a task model and a presentation model (the upper part is a task model, the leaf nodes are widgets). Annotations can be attached to each node. These annotations specify the *layout* of the node (using a Grid Bag layout), the *task preference* (describes whether a task is suitable to a given platform), a *priority* (nodes with a higher priority will be placed first in the display panels), a *splittability* (indicates whether the subnodes can be spread over multiple pages) and an *importance* (optional or mandatory). SGUI optimizes the layout generated for a given platform by resizing widgets, operating widget substitutions, applying a different layout manager (Flow layout), or changing the repartition of widgets between screens. The transformations in SGUI are totally automatic (there is no human control on the process). An obvious limit of this approach is that task model and presentation model need to have a compatible structure: it is impossible to specify that two non contiguous tasks in the task model must appear in the same presentation unit, except if all tasks between them also belong to that presentation.

2.2.4.k PUC

The personal universal controller (PUC) is a universal remote control for home appliances such as media or tape players [Nich02]. The user interfaces of the PUC are automatically generated from higher-level descriptions. The specification mostly includes elements of the domain model, under the form of state variables. Examples of state variables are the current radio station, or the current track of the CD player. Each state variable is associated to a type and, optionally, to a label. Elements are grouped through group trees and dependencies between interface components can be specified. The automatic generation of UIs comes with a “Smart Template” mechanism [Nich04]. A Smart Template is a kind of domain specific design pattern, which defines the set of state variables (e.g. “time”, “volume”) and functions (e.g. “play”, “record”...) typical for a given kind of appliance. Multiple combinations of states and commands are allowed in the template definition, which allows a single template to be applied across slightly different appliances. At the rendering stage, the elements specified in the Smart Template are then rendered using platform-specific controls and intelligent layout and resizing mechanisms. PUCs built using Smart Templates are cross-platform consistent (they are generated from the same domain model), while respecting the platform-specific conventions the users are accustomed to. Of course, the Smart Template mechanism

seems difficult to adapt to all kinds of graphical user interface, since it requires an a priori knowledge of the usual functionalities of the target platform.

2.2.4.1 Supple

Supple [Gajo04] treats interface generation as an optimization problem, where the rendered interface must meet the constraints of the device while minimizing the estimated cost for the user. Supple requires three declarative descriptions of the user interface: a platform model, a user model, represented in terms of user traces and an “interface model”, which roughly corresponds to the domain model in the CAMELEON framework, with some additional constraints expressed on mappings. This “interface model” includes a set of interface elements, defined in terms of data types (primitive type, collection...), the set of interface constraints specified by the designer (e.g. “all lights have to be rendered by the same widget”) and optional additional information about each element (label, indication that the element is read-only, list of likely values...) The platform model is composed of the set of available widgets, the set of platform-specific constraints (functions that map element-widget assignments to either true or false) and platform-specific functions for evaluating the suitability of given widgets in particular contexts. The user is modelled in terms of his/her user traces, where a trace is a set of trails i.e. a sequence of elements manipulated by the user. A legal solution in Supple is a solution that maps each interface element to a widget while satisfying interface and device constraints. An optimal solution is a legal solution that minimizes the expected cost, defined as the sum of the costs of each user operation recorded in the trace. Supple’s rendering algorithm is an A* search. So, the precise Concrete User Interface generated by Supple depends not only of the specifications given by the designer, but also of the user’s specific experience with the system (Supple is auto-adaptive). So, the final layout of a user interface generated by Supple is neither controlled by the designer, nor immutable for a given user.

2.2.5 The abstraction-reification approach

The abstraction-reification approach extracts an abstract description from a UI’s code and generates the code of new UIs adapted to the target platform(s) e.g. PIMA, Vaquita/ReversiXML. Figure 2-14 schematizes this approach.

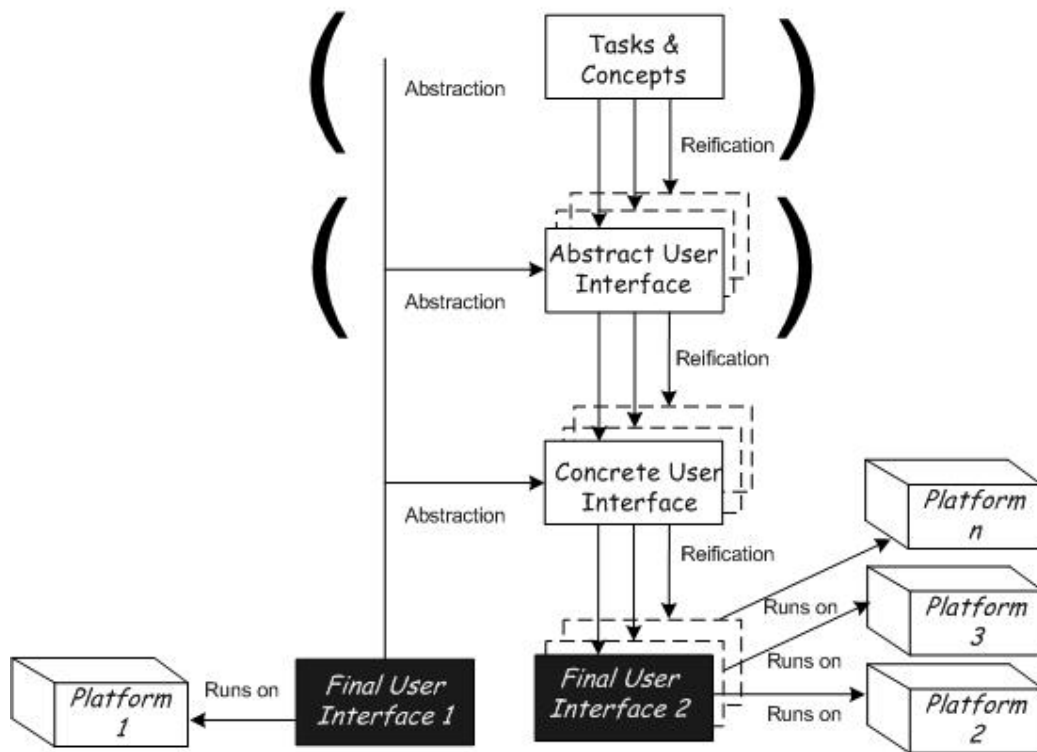


Figure 2-14 The abstraction-reification approach

2.2.5.a PIMA

The PIMA system [Bana04] is a tooling environment for building and deploying applications targeted to run on several devices. The first step when designing a PIMA application consists in creating a generic application model. This generic description can be created by hand or extracted from a specific instance of an application interface using PIMA's reverse engineering facilities (a generalization process in PIMA's framework). The generic representation is then converted into multiple device-specific representations by the specialization process. The third process or tweaking consists of handcrafting the automatically generated device-specific representations.

2.2.5.b Vaquita/ReversiXML

ReversiXML [Boui04] (formerly Vaquita) is a tool that reverse-engineers HTML pages into UsiXML models, both at the Abstract User Interface (AUI) and the Concrete User Interface (CUI) levels. ReversiXML not only creates a generic representation of the HTML source, it also adapts this representation following the characteristics of the computing platform where the UI is intended to migrate (retargeting). Abstraction and retargeting heuristics in Vaquita and ReversiXML can be entirely customized by the designers.

The reification stage consists in getting a new running interface from the UsiXML specification, by reification to the FUI level or by rendering of the CUI model.

Reification is not covered by ReversiXML, but by other tools in the UsiXML suite: the GrafiXML editor, which is able to generate XHTML and Java code, and the interpreter InterpiXML.

2.3 Single-authoring

Transversally to the five development approaches described above, several tools (2.2.2.a, 2.2.4.d, 2.2.4.i) promote a single authoring⁷ solution, which consists in centring the design process on a source interface designed for the least constrained platform. However, our own approach differs significantly from the other single authoring approaches investigated:

- In contrast to our semi-automatical approach, the SGUI tool described by Wong and Chu (2.2.4.i) automatically generates user interfaces starting from a hierarchical model combining task and presentation.
- Similarly, the single-authoring approach for multimodal interfaces investigated within the MONA project (2.2.4.d) relies on the automatic adaptation of a root user interface to several platforms.
- More similar to our vision, the tool developed by Ding (2.2.2.a) adopts a semi-automatic approach but takes as input a user interface in Java, enhanced with annotations indicating order or grouping relationships. The transformation rules implemented in the tool are either applied when selected by the designer, or applied automatically by the transformation engine. Those rules are similar to our own transformation rules but, of course, their scope is limited to Java interfaces and they can not rely on high level semantic information, such as the information that can be drawn from a task model, for example.

2.4 Comparison of the approaches

On the basis of the Unified Reference Framework, five categories of approaches to the development of UIs for multiple platforms have been identified, and representative tools have been described. This section will compare these five categories, using a set of criteria that were found relevant:

- The *production costs* include not only the development cost of the first release of the system but also the cost of system maintenance: modifying or adding functionalities, modifying or adding formats.
- The *completeness* describes the possibility of obtaining any type of UI using the approach.

⁷ This phrase is sometimes employed with other meanings in the literature. For example, [Brau04] defines single autoring as an approach that relies on “a single, device-independent user interface description, which can be mapped to a good concrete UI for each feasible target device”.

- The *level of control* offered to the developers includes:
 - The “granularity” of the developer's input (i.e.; the level of detail he/she can access). Fine-grained approaches allow the designer to control the smallest details of widgets attributes values, while coarse grained approaches only permit to specify tasks and concepts, for example.
 - The predictability of the final user interface rendered.
- The *usability* of the each UI produced considered separately, i.e., their observance of commonly accepted ergonomic recommendations.
- The *cross-platform consistency* describes the relations between versions of a user interface on different platforms or devices, i.e.; similarities/differences between versions in terms of tasks offered, concepts represented, presentation, dialog...
- The *guidance* refers to the means available to help the designer in his/her work of building a user interface for multiple platforms.

2.4.1 Production costs

The production costs criterion is crucial for the acceptance of any development technique. The cost of producing software is not only devoted to the production of the first system: it is widely estimated that 70% of the cost of software is due to maintenance activities [Meye97]. The largest part of maintenance activities consists in adding or modifying functionalities due to changes in user requirements (41,8%). The next maintenance activities in terms of costs are due to changes in data formats (17,6%) or hardware changes⁸.

A truly accurate evaluation of software production costs as a function of the chosen approach would require benchmark tests that are far beyond the scope of this thesis. We have little knowledge, for example, of the relative effort required to *specify* a UI vs. *coding* it. Nevertheless, some general comparisons can be made at this point. We will consider:

- The cost of developing a first version of the user interface for N platforms;
- The maintenance cost when functions are modified or added in response to changes in user requirements;
- The maintenance cost when formats change (for example, modification of 1 language, 1 graphical toolkit...)

⁸ Figures from Lientz, B.P. & Burton Swanson E.(1980). Software Maintenance Management. Boston: Addison-Wesley, cited by [Meye97].

2. State of the Art

2.4.1.a Cost of developing user interfaces for N platforms

Approach	Estimation of the development cost
Traditional development	High: the design effort is directly proportional to the number of platform families (i.e. platforms with similar capabilities) and the coding effort is directly proportional to the number of system versions. Furthermore, different programming languages may be used.
Unique portable code using virtual toolkits	Low: one single interface is designed and coded.
Unique portable code using generic clients	Low: one single interface is designed and coded. The cost increases when using stylesheets and/or XSLT transformations (one stylesheet or XSLT transformation has to be produced per platform family), but even in this case, 2 languages only have to be used.
Transcoding approach	Low: one platform-specific UI is designed and coded; the transcoding tool takes care of the adaptation to other platforms.
Multireification approach with a specification of the UI at a high level in the Unified Reference Framework (Tasks & Concepts)	Low ⁹ : one single interface is specified, nothing is programmed.
Multireification approach with a specification at a lower abstraction level	Moderate: the cost is directly proportional to the number of platform families (one UI per platform has to be specified), but only one specification language has to be used.
Abstraction-reification	Low: one platform-specific UI is designed and coded; the tool takes care of the adaptation to other platforms.

2.4.1.b Cost of modifying or adding functionalities

Approach	Cost of modifying/adding 1 functionality in N platforms
Traditional development	High: modifications in N platform-specific versions required.
Unique portable code using virtual toolkits	Low: modification of the code in 1 single version required.

⁹ At least theoretically: a small experimental evaluation conducted on the TERESA multi-reification tool has shown no significant difference in the total development time necessary to develop UIs for two different platforms using the generation tool or directly using XHTML [Ches04].

2. State of the Art

Unique portable code using generic clients	Low: modification of the code in 1 single version required
Transcoding approach	Low: modification of the code in 1 single version required
Multireification approach with a specification of the UI at a high level in the Unified Reference Framework (Tasks & Concepts)	Very low: 1 single modification at the Tasks&Concepts level of the specification required.
Multireification approach with a specification at a lower abstraction level	Moderate: modification of the specification in M platform-specific versions required, where M is the number of platform families ($M \leq N$).
Abstraction-reification	Low: modification of the code in 1 single version required, the tool passes the modifications on to other platforms.

2.4.1.c Cost of modifying or adding formats

Approach	Cost of modifying/adding 1 format in N platforms
Traditional development	High: modifications across the code of all the platform-specific versions concerned.
Unique portable code using virtual toolkits	Moderate: modifications across the code of 1 single version.
Unique portable code using generic clients	Moderate: modifications across the code of 1 single version.
Transcoding approach	Moderate: modifications across the code of 1 single version + N-1 modifications of the transcoding tool.
Multireification approach with a specification of the UI at a high level in the Unified Reference Framework (Tasks & Concepts)	Low: N modifications of the generation tool, no change in the UI's specification.
Multireification approach with a specification at a lower abstraction level	Low: N modifications of the generation tool, no change in the UI's specification (except if the designer wants to take advantage of a new feature such as a new widget, for example)
Abstraction-reification	Low: 1 modification of the reverse-engineering tool if the source format is adapted + N-1 modifications of the generation tool

2. State of the Art

2.4.2 Completeness

While some approaches impose no restriction on the kind of user interface they can provide (they are complete), others are not suitable for every purpose. Completeness is thus an important criterion that characterizes the applicability domain of a given method.

Approach	Level of completeness
Traditional development	Maximum.
Unique portable code using virtual toolkits	Maximum.
Unique portable code using generic clients	High, if used in combination with scripting languages, but restriction on the type of dialog (form-based dialog mainly).
Transcoding approach	Moderate, practicability mostly demonstrated for Web-based UI's.
Multireification approach with a specification of the UI at a high level in the Unified Reference Framework (Tasks & Concepts)	Low, practicability mostly demonstrated for simple form-based interfaces (TERESA) or for UIs of control systems (ArtStudio, Supple, PUC).
Multireification approach with a specification at a lower abstraction level	Low/high: depends of the possibilities of the specification language: possibility of specifying any type of dialog, any type of presentation...
Abstraction-reification	Moderate, practicability mostly demonstrated for Web-based UI's.

2.4.3 Level of control

Control is an important factor of user satisfaction and is likely to have a great impact on the acceptance of a given approach. As mentioned above, this criterion includes:

- The level of detail the designer can access, which can range from a task's specification until the very details of a widget's presentation;
- The predictability of the final results, with respect to the input provided (code, specifications, parameters to transformations ...)

Approach	Level of control
Traditional development	Maximum.
Unique portable code using virtual toolkits	Very high, even if no control on the virtual machines' exact behaviour.
Unique portable code using generic clients	Very high, even if no control of the browsers' exact behaviour.

2. State of the Art

Transcoding approach	Moderate: the first version can be fully controlled by the designer, but the subsequent transformations are automatic (even if some transformations accept user-defined parameters).
Multireification approach with a specification of the UI at a high level in the Unified Reference Framework (Tasks & Concepts)	Very low, even if some systems allow user-defined parameters.
Multireification approach with a specification at a lower abstraction level	Moderate: the designer has a general control on the choice and presentation of widgets, but the details are hidden to him.
Abstraction-reification	Moderate: the first version can be fully controlled by the designer, but the subsequent transformations are automatic (even if some transformations accept user-defined parameters).

2.4.4 Usability

The usability of a user interface is a key factor for user acceptance of the product delivered. The usability can be assessed thanks to a range of methods (heuristic evaluation, user testing...), criteria and rules. Producing usable user interfaces for each platform-specific version of a multiplatform system is difficult, due to differences between the capabilities of the different devices and softwares.

Approach	Usability of the final user interfaces rendered
Traditional development	Potentially maximum, but has to be guaranteed by the good practices of the developers
Unique portable code using virtual toolkits	Low if the platforms are very different: no adaptation to the platform is provided (except portability and sometimes adaptation to the platform's look-and-feel), which generates usability problems (excessive scrolling, tasks inappropriate on the device, illegible fonts, inappropriate widgets...) Toolkits of self-adaptive widgets offer a solution to the last problem.
Unique portable code using generic clients	Low if the platforms are very different: no adaptation to the platform is provided (except portability). Stylesheets may improve the usability of the UI's, but they are still insufficient when very different target platforms are considered (very different screen size, or different modality).
Transcoding approach	Depends on the transformations provided: can range from simple code-to-code equivalences to complete reauthoring of the system.

2. State of the Art

Multireification approach with a specification of the UI at a high level in the Unified Reference Framework (Tasks & Concepts)	Potentially high usability and excellent adaptation to the target platform, depending on the heuristics of the generation tool.
Multireification approach with a specification at a lower abstraction level	Potentially high usability, as one specification per platform family is provided, but depends on good practices of the developers.
Abstraction-reification	Low if the platforms are very different: no adaptation to the platform is provided. Higher if the tool is able to adapt the recovered specification to the characteristics of the target platform (retargeting).

2.4.5 Cross-platform consistency

The cross-platform criterion is one important aspect of the usability of a cross-platform user interface. It can be defined as the capability to provide similar functionalities, similar operation procedures, similar data representations and same data sets in each version of the UI [Flor04].

Approach	Cross-platform consistency
Traditional development	Has to be guaranteed by the good practices of the developers.
Unique portable code using virtual toolkits	Maximum: 1 single UI.
Unique portable code using generic clients	Maximum: 1 single UI. Lower if stylesheets are used, especially if they introduce very different placements, colours...
Transcoding approach	High: the general structure of the UI is respected. Lower if reauthoring techniques are used.
Multireification approach with a specification of the UI at a high level in the Unified Reference Framework (Tasks & Concepts)	High, but depends on the generating heuristics of the tool.
Multireification approach with a specification at a lower abstraction level	Very good between platforms of the same family, but has to be guaranteed by the good practices of the developers for platforms from distinct families.
Abstraction-reification	High: the general structure of the UI is respected. Lower if retargeting is used.

2. State of the Art

2.4.6 Guidance

Designing user interfaces for a variety of computer devices at the same time is a complex task. Therefore, designers should benefit from some assistance in the design process, at least if different versions of the user interface have to be produced (if one single code or specification is provided, no guidance is needed). Guidance should have an impact on two criteria mentioned above: usability and cross-platform consistency.

Approach	Guidance
Traditional development	None.
Unique portable code using virtual toolkits	–
Unique portable code using generic clients	–
Transcoding approach	–
Multireification approach with a specification of the UI at a high level in the Unified Reference Framework (Tasks & Concepts)	–
Multireification approach with a specification at a lower abstraction level	None.
Abstraction-reification	–

2.5 Global comparison and conclusion

The table below (Table 2-1) provides a synthetic view of how each approach can be evaluated against the eight criteria listed previously (development costs, cost of modifying functionalities, cost of modifying formats, completeness, level of control, usability, cross-platform consistency and guidance).

2. State of the Art

Criteria	Development costs	Maintenance costs (modification of functionalities)	Maintenance costs (modification of formats)	Completeness	Level of control	Usability if the platforms are very different	Cross-platform consistency	Guidance
Approach								
Traditional development								
Virtual toolkits								-
Generic clients								-
Transcoding								-
Multireification approach (high level specification)								-
Multireification approach (lower level specification)								
Abstraction-reification								-

Legend

	= good for that criterion
	= medium for that criterion
	= bad for that criterion
-	= irrelevant

Table 2-1 Global comparison of development approaches on all criteria

A general conclusion that can be drawn from the observation of this table is that no approach is free from serious drawbacks:

- (1) Traditional development of MUI is very expensive, both for the development of the first versions of the user interfaces and for their maintenance. Furthermore, ad-hoc development of platform specific versions does not guarantee any form of consistency within the multiplatform system.

- (2) “One fits all” approaches based on virtual toolkits, generic clients or transcoding are cheaper solutions, but their common weakness is a lack of adaptation of the user interfaces produced to their target platform when these user interfaces are ported to platforms with very distinct capabilities.
- (3) Each one of the three model-based approaches (multireification from high-level specifications, multireification from low-level specifications and abstraction reification) has a serious limitation:
 - Multireification from high-level specifications such as task models and domain models neither offer a solution for all types of GUIs (completeness), nor permit anticipating what exact design will be generated from the specifications (control).
 - Multireification approaches based on lower-level specifications, at least when they cover several kinds of target platforms, such as UIML, are more expensive to develop and maintain, generate user interfaces that are less similar (cross-platform consistency) and offer no guidance to the designer.
 - The abstraction-reification approach neither offers any guarantee on the quality of the final results, nor is completely satisfying in terms of control and completeness (in fact, little is known about the possibilities of abstraction-reification tools, which are not numerous and mostly targeted to the reverse-engineering of Web pages).

In conclusion, we think that there is a place among the enumerated techniques for another model-based approach:

- Providing a better level of control and a larger coverage than multireification from high-level specifications or abstraction-reification, thanks to designer-controlled explicit transformation rules applied to a source user interface specification.
- Generating cross-platform consistent user interfaces, starting from detailed specifications semi-automatically built from the source user interface, which represents an advantage in terms of guidance and development costs in comparison to multireification from low-level specifications.
- Specifically targeted to multiplatform systems where the capabilities of the platforms are very different, which is not the case of abstraction-reification.
- Conceived as a trade-off between cost and quality criteria on one side, and conflicting quality criteria such as usability and cross-platform consistency on the other side.

The two basic ingredients for the model-based method we introduce in this thesis, i.e; design by graceful degradation, are a source interface specification and a catalogue of transformation rules. So, the next chapters will be devoted to the description of the chosen specification language and its underlying concepts (Chapter 3) and to a presentation of the transformations rules (Chapter 4).

Chapter 3 Language and Models

Our method to develop user interfaces for multiple platforms is model-based. Therefore, it requires the use of a user interface description language (hereafter UIDL). Our method is also transformational, as it consists of specifying a source UI, designed for the least constrained platform and then applying transformation rules to it to produce specific UIs targeted to more constrained platforms. These transformation rules will process different layers of the specification, according to the abstraction levels defined in the Unified Reference Framework described above. For this reason, the UIDL we will use needs to be structured in several layers. Until now, only a few UIDLs meet this requirement: XI ML [Puer02], the last versions of UIML [Ali03] and UsiXML.

We have chosen the last:

- UsiXML is the UIML developed in our team, which offered opportunities to collaborate on its design.
- It relies on a theoretical framework aimed to address the development of user interfaces for multiple contexts of use (the Unified Reference Framework described above).
- It has the additional advantage to possess a graphical syntax for a majority of constituent models, which is an advantage in terms of communication and acceptance. Furthermore, graphical editors are available that permit editing these models.
- Contrary to XI ML, UsiXML's language specifications are freely available and not protected by copyright.
- Unlike XI ML, UsiXML can be rendered in Java, HTML and XUL; even if the UsiXML renderers are still at the development stage and can not compete with the numerous implementations of UIML.

This chapter presents UsiXML and the conceptual content of this language. We will focus on the representation of tasks, domain objects, presentation and dialog, platforms and interactors. The platform and interactor models represent a specific contribution of this thesis and are thus described in more detail.

3.1 UsiXML

The user interface description language UsiXML ([Limb04]) allows designers describing various aspects of a user interface, while using the same language. Depending on the needs, a designer can adopt distinct viewpoints on the same user interface. In the early stages of design, he might choose to specify only high level functionalities (tasks) or domain objects. Later, he might want to give a very detailed description of the dialog and presentation. These views on a user interface, called models in UsiXML, are organized in abstraction layers, following the Unified Reference Framework.

A UsiXML specification is thus a combination of models. None of these models is mandatory and every combination of models is allowed. UsiXML is equipped with eight main types of models, as illustrated on Figure 3-1: a task model, a domain model, an AUI model, a CUI model, a mapping model, a context model, a resource model and a transformation model.

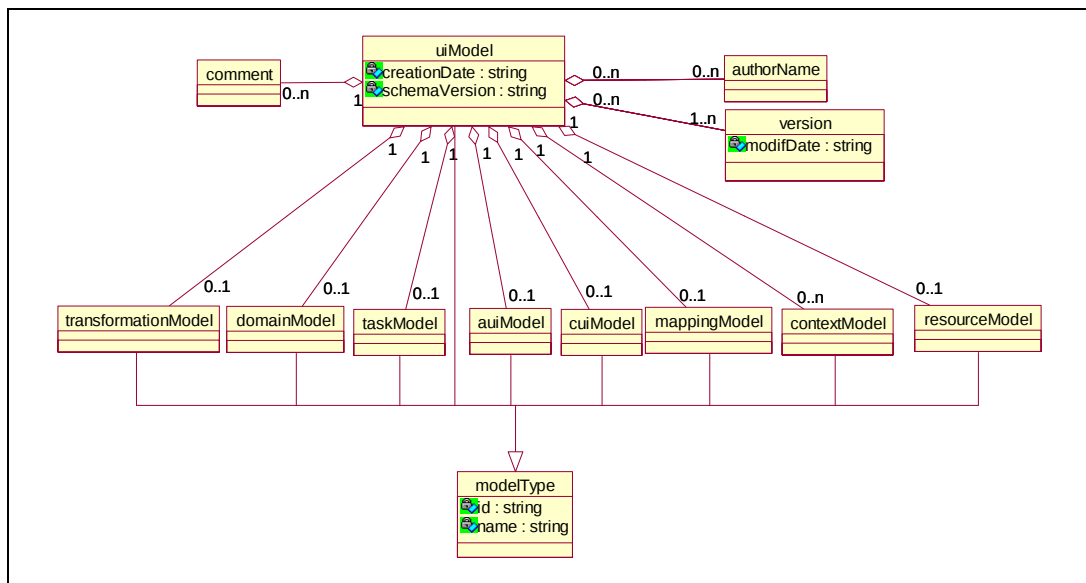


Figure 3-1 Constituent models in UsiXML

The *task* and *domain models* both belong to the Tasks&Concepts level of the Unified Reference Framework. The task model is a description of the tasks carried out by a user in interaction with the system, while the domain model is a description of the objects and classes viewed or manipulated by the user.

The *AUI model* (Abstract User Interface) lies at the next abstraction level in the Reference Framework. It is used to specify which group of tasks and domain concepts will be presented together (for example, in the same window or card).

The *CUI model* (Concrete User Interface) is a detailed specification of the appearance and behaviour of the UI's elements.

The *mapping model* serves to establish relationships between models or elements of models (for example, between a task belonging to the task model and the widget of the CUI that permits the execution of this task).

The *context model* consists of three submodels: a user model, an environment model and a platform model:

- The *user model* decomposes the user population into user stereotypes, described by attributes such as the experience with the system or with the task, the motivation, etc.
- The *environment model* describes any property of interest of the global environment where the interaction takes place. The properties may be physical (e.g., lighting or noise conditions) or psychological (e.g., level of stress).
- The *platform model* captures relevant attributes related the combination of hardware and software where the user interface is intended to be deployed.

The *resource model* contains elements (title, tooltip, mnemonic...) specific to a given context (for example, the user's language). Resources are linked to objects of the CUI or AUI model.

At last, the *transformation model* permits the specification of transformation rules under the form of graph transformation rules, taking advantage of the underlying graph structure of UsiXML. A graph transformation is expressed as a pair {LHS, RHS}, where LHS is the Left Hand Side of the rule and RHS is the Right Hand Side of a rule. LHS expresses a graph pattern that, if it matches the host graph, will be modified to result in another graph called resultant graph, according to what is specified in RHS [Limb04b]. This formalism supports different types of transformations: abstraction (e.g.; recovering an AUI model starting from a CUI model), reification (e.g.; generating a CUI from a task model and a domain model) and translation (e.g.; adapting a CUI designed for one specific context of use to another context of use). We will not rely on this formalism in this thesis, for two reasons:

- Some GD rules are inherently difficult to express using graph transformations. For example, it is far more easy and intuitive to express layout transformations by describing the algorithms used to generate the results then by giving a precise description of the pre- and post-conditions of the rule as patterns defined on a graph. In particular, the difficulty in ordering the substeps of a given rule is a serious obstacle both for layout transformations rules and for the splitting rules presented on Chapter 5.
- Even for simple transformations, such as modifying fonts size for example, relying on graph transformations has a negative impact on performance, because the process requires the collaboration of different tools, the use of several internal formalisms and several steps: (1) Firstly, models are built using a graphical editor. These editors (IdealXML [Mont05], GrafiXML) possess an internal representation of the model, and export it in UsiXML (2) The UsiXML models are imported within AGG (Attributed Graph Grammar tool [Ehri99]), a graphical environment for specifying

and executing graph transformations where the rules are applied to the graph structure (3) The resulting models are exported from AGG to UsiXML.

The next sections will be dedicated to a precise definition of the conceptual content of the models that are relevant in the framework of Graceful Degradation: task, domain, AUI, CUI, platform, interactor and mappings. The interactor model is a separate model that is not part of UsiXML. It permits producing meta-descriptions of the toolkits available on a given platform. We will not make use of the other UsiXML models.

3.2 Task Model

A task model, as defined above, is a description of the tasks that a user will be able to accomplish in interaction with the system. This description is a hierarchical decomposition of a global task, with constraints expressed on and between the subtasks. The task model of UsiXML (see Figure 3-2) is an (slightly) extended version of ConcurTaskTree (CTT) [Pate00]: a hierarchical task structure, with temporal relationships specified between sibling tasks. A complete description of this task model can be found in Annex A.

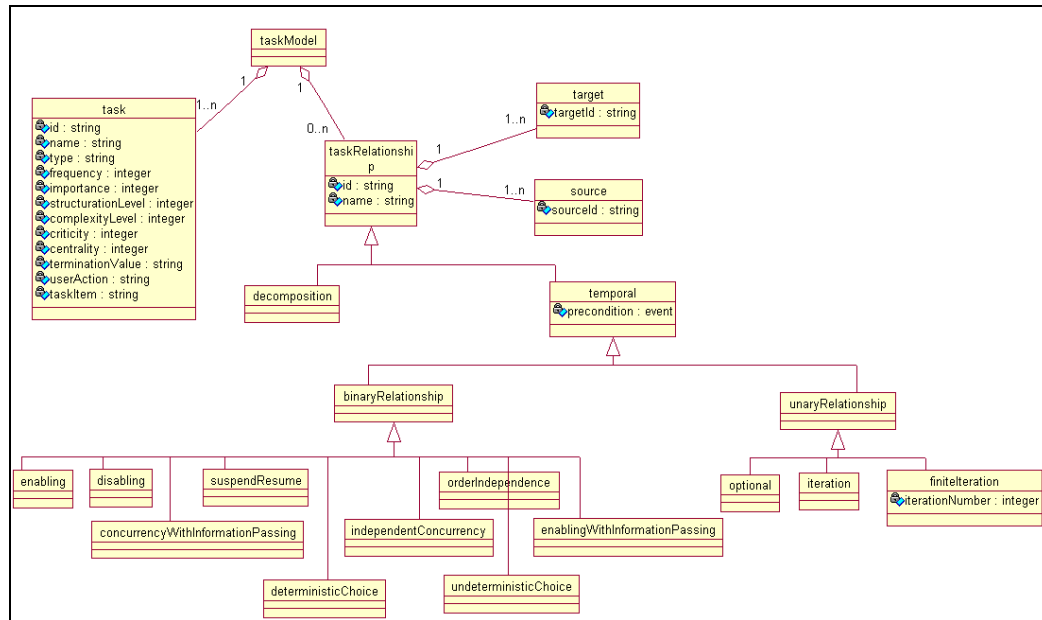


Figure 3-2 Meta-model of the UsiXML task model

3.3 Domain Model

UsiXML relies on UML class diagrams and objects diagrams [Rati97] for its domain model. The main concepts in a UsiXML class diagram, as represented on the meta-model on Figure 3-3 are classes, objects, attributes, methods and relationships.

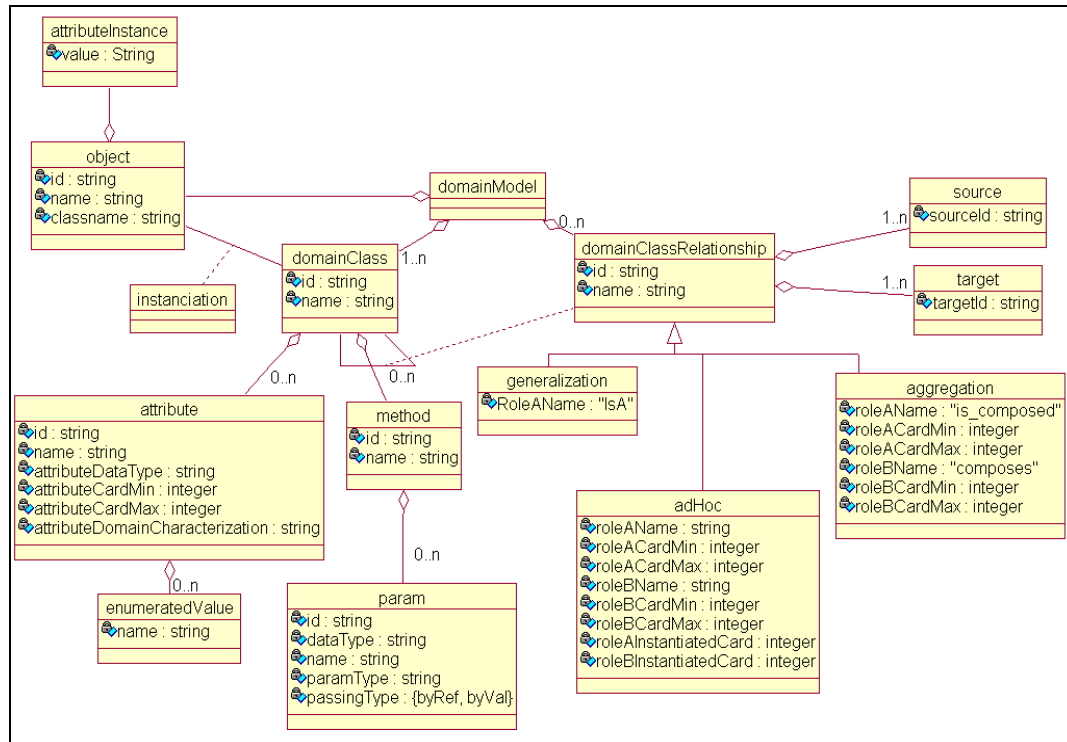


Figure 3-3 Meta-model of the UsiXML domain model

3.4 AUI Model

An AUI model is an expression of the rendering of the domain concepts and tasks in a way that is independent from any modality of interaction. In UsiXML, the AUI (see meta-model on Figure 3-4) is populated by Abstract Interaction Objects and AIO Relationships.

Abstract Interaction Objects (AIO's) are elements populating the AUI. They may be of two types: *Abstract Containers* (ACs) and *Abstract Individual Components* (AICs).

Abstract Containers (ACs), also named *interaction spaces* or *presentation units*, define grouping of tasks that have to be presented together, in the same window or page for example. An abstract container contains other AIO's. It may be reified into graphical containers like windows or dialog boxes.

Abstract Individual Components (AICs) are individual elements populating an abstract container. AICs are an abstraction of widgets found in most toolkits (for example windows, buttons or vocal output widget in auditory interface).

An AIC may be composed of multiple *facets* describing the type of interactive tasks it is able to support. Each facet describes a particular function an AIO may assume. Four main facets have been identified:

1. An *input* facet describes the type of input that may be accepted by an AIO.

2. An *output* facet describes what data may be presented to the user by an AIO.
3. A *navigation* facet describes the possible container transition a particular AIO may enable.
4. A *control* facet describes possible methods of the domain model that may be triggered from an AIO.

Some AIO's may assume several facets at the same time (for instance, an AIO may display an output while accepting an input from a user).

AIO relationships are abstract relationships between two distinct AIO's. Our description of these relationships is more precise and complete than what can be found in the current UsiXML specification (introduction of new constraints, of new types of relationships). These propositions are intended to be included in the next UsiXML release.

AIO relationships indicate the existence of some spatio-temporal or logical setting among AIO's. A given pair of source and target AIO's can be linked by several AIO relationships. The operators between the abstract interaction objects in the TERESA tool [Pate02] or the abstract constraints expressed between components in some constraint-based automated layout systems [Lok01] are examples of the use of AIO relationships in the literature. Different types of AIO relationships can be defined:

- *Decomposition relationships* allow specifying a hierarchical structure of abstract containers.
- *Spatio-temporal relationships* are modality-independent constraints between AIO's, using the temporal relationships defined by Allen [Alle83]. When UsiXML is used for specifying GUIs, they are redundant with the graphical relationships defined at the Concrete User Interface level: for this reason, we will not make use of Allen relationships in the context of this thesis.
- *Abstract grouping* is an abstract relationship between two or more AIO's of the same abstract container that need to be grouped together, regardless of the actual layout that will be defined at the Concrete User Interface level.
- Conversely, *abstract separation* is an abstract relationship between two AIO's of the same abstract container that need to be separated from each other (for example, by a blank space or a separation line in graphical user interfaces, by a beep in auditory user interfaces...)
- *Differentiation* is an abstract relationship between two AIO's that should be differentiated from each other. For example, an "erase all" button could be differentiated from its neighbours, in order to avoid confusions.
- *Is-title-of* is an abstract relationship between one output AIO that represents a title and the AIO it describes.

3. Language and Models

- *Hierarchy* is an abstract relationship between two or more AIO's that form a hierarchy. For example, a series of titles in a document could be linked with a hierarchy relationship.
- *Abstract adjacency* is an abstract relationship between two AIO's that have to be adjacent (which is not possible to specify using Allen relationships).
- The *Order* relationship specifies some kind of ordering between two or more AIO's (list...)
- *Dialog control* relationship allows a specification of a flow of control between the abstract interaction objects in terms of LOTOS operators.

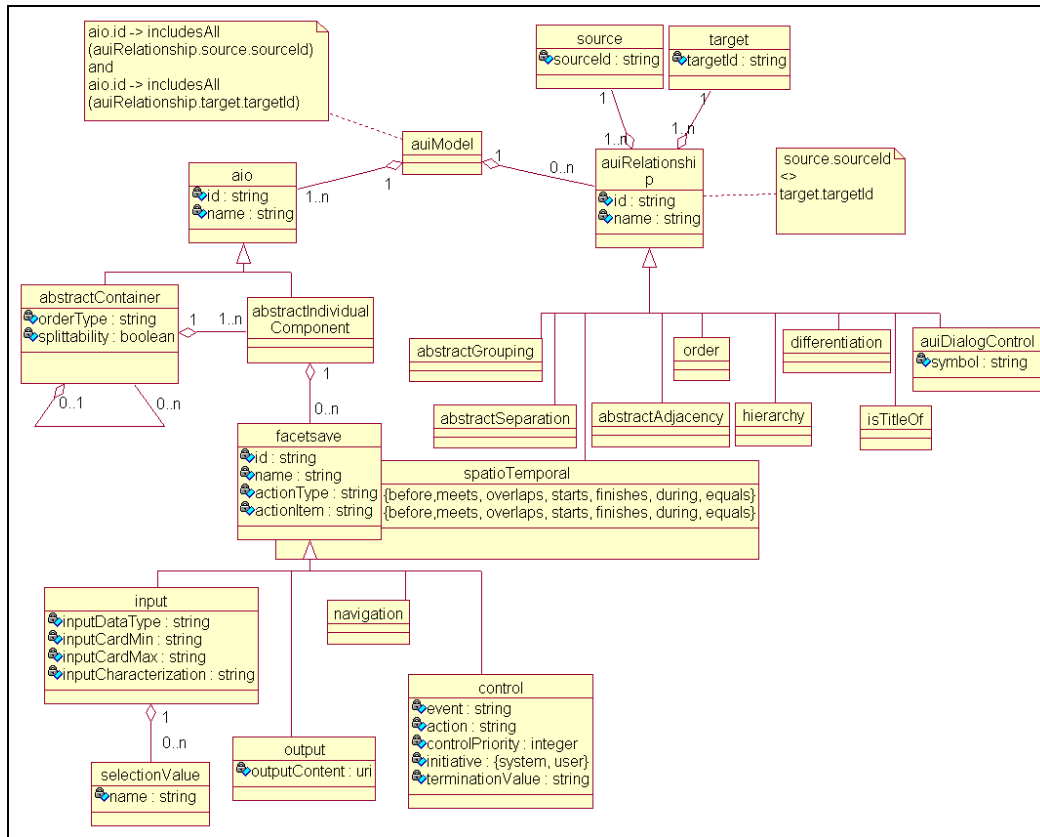


Figure 3-4 Meta-model of the UsiXML Abstract User Interface

3.5 CUI Model

A CUI Model represents a concretization of an AUI Model. A CUI is populated by *Concrete Interaction Objects* and *Concrete User Interface relationships* between them.

Concrete Interaction Objects (CIO's) are the building blocks of the CUI. They are an abstraction of widget sets found in popular toolkits such as Java AWT/Swing or HTML4.0. UsiXML distinguishes between *graphical CIO's* and *auditory CIO's*. In the context of this thesis, we will only consider graphical CIO's. UsiXML further classifies graphical CIO's in two categories: *graphical containers* and *graphical individual components* (Figure 3-5).

A *graphical container* is a graphical CIO that can contain other CIO's, including other containers. UsiXML's metamodel contains a list of 11 types of containers: dialog box, menu bar, menu pop-up, tabbed dialog box and tabbed item, table and cell, tool bar, status bar, window and box.

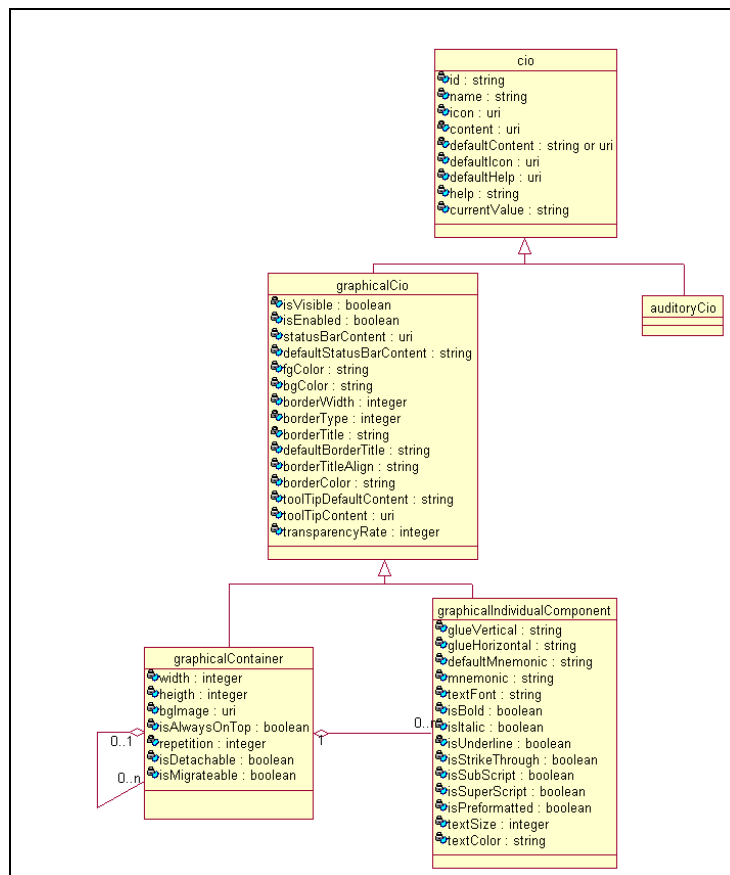


Figure 3-5 Concrete Interaction Objects in UsiXML: upper part of the hierarchy

A *graphical individual component* is a CIO that permits observing or manipulating domain objects, or to call domain methods. Graphical individual components are a direct abstraction of widgets found in popular toolkits. For example, UsiXML's *checkBox*

component corresponds to `<INPUT TYPE = CHECKBOX>` in HTML 4, `JCheckBox` in Java Swing or `Checkbox` in Tcl/Tk. The list of graphical individual components in UsiXML includes *text component*, *video component*, *image component*, *button*, *toggle button*, *radio button*, *checkbox*, *combobox*, *listbox*, *spin*, *menu items drawing canvas*, *colour picker*, *date picker*, *file picker*, *hour picker*, *progression bar* and *slider*.

Concrete Interaction objects are linked by Concrete User Interface relationships. Again, they are divided into *auditory relationships* and *graphical relationships*. *Dialog control relationship* can be defined between both types of interaction objects.

Graphical relationships express different types of constraints between a source graphical CIO and a target graphical CIO:

- *Relative positioning constraints* specify a positioning relationship between two components. Most of these constraints are a concretization of Allen relationships for graphical UI's: insertion, left-of, right-of, superiority, inferiority. Other constraints were impossible to express at the AUI level: left-indentation, right-indentation, horizontal adjacency and vertical adjacency.
- *Graphical transitions* specify a transition between two containers. Transition types are open, close, minimize, maximize, suspend/resume.
- *Alignment relationships* specify a relationship between two components and a guide extending their edges (vertical alignment, horizontal alignment) or crossing their centre either horizontally (horizontal centred alignment) or vertically (vertical centred alignment). Except centred alignment, these relationships have direct correspondences at the AUI level (i.e; they can be expressed in terms of Allen relationships).
- *Adjacency relationships* indicate that there is no interpolated component between two graphical CIO's, either in the horizontal direction (horizontal adjacency) or in the vertical direction (vertical adjacency).

Dialog control relationships allow a specification of a flow of control between the concrete interaction objects, independently from the task model, using LOTOS operators. Dialog control relationships at the CUI level are a refinement of the dialog control relationships defined at the AUI level.

Relative positioning constraints (e.g; left-of, inferior-to...) between two components can also be specified by the type of *box* that contains the CIO's. *Boxes* are the basic layout mechanism in UsiXML. A box can contain other boxes or graphical individual components. Boxes are characterized by:

- Their type: horizontal, vertical, grid.
- Their relative width and height with respect to their parent container.
- Information on their resizable and their minimum width and height.
- Optional balance constraints.

- A “splittable” attribute that indicates whether the box may be redistributed between several abstract containers.

UsiXML's Concrete User Interface is a hybrid model that contains at the same time information on the presentation of the UI and on its behaviour. At the CUI, each CIO can be linked to a *behaviour*.

A *behaviour* is the set of reactions of the user interfaces to *events* such as user interactions, changes in the system state, period of time elapsed... These events trigger *actions*, such as a method call or a transition to a target container, provided that certain *conditions* are met.

3.6 Platform Model

UsiXML's platform model, as well as the related interactor model presented below, has been developed in the context of this doctoral research.

A UI *platform model* is a description of the *platforms* that may run this UI. A *platform* is a specific combination of hardware (screen, keyboard, memory...) and software (OS, graphical toolkit, browser ...). A platform model should contain values for each feature of the platform that influences the design of the user interface (the screen size and resolution, the graphical toolkit...)

Unlike user models, which are widely used in systems such as Adept [Wils96], Trident [Vand97], FUSE [Lonc96], CT-UIIMS [Mart90], Mobi-D [Puert97], Tadeus [Schl97] or WebML [Ceri00], platform models are almost non-existent in model-based user interface management systems. To the best of our knowledge, only two tools in the multireification category (2.2.4), ArtStudio [Thev01] and TERESA [Mori03], contain a platform model, but this model is very rudimentary: in ArtStudio, the platform model only specifies the target language and the screen dimensions and resolution, while TERESA only permits associating to tasks and objects an annotation indicating the set of platforms suitable to support it. The User Interface mark-up language XIIML [Puer02] is equipped with a platform model, named “Workstation model”, that should represent the computer platforms on which the user-interface may be instantiated, but the XIIML specification proposes only imprecise attributes such as “description” or “feature”.

On the other side, the Web community has shown a huge interest for specifying device capabilities, in order to perform adaptation of Web content delivered to platforms such as WAP phones or PDA's. A number of platform modelling technologies have been developed [Butl01]. Some of these initiatives such as Universal Plug and Play, promoted by Microsoft, Media feature Sets, proposed by the Internet Engineering Task Force, or the W3C recommendation CC/PP [W3C04] (Composite Capability/Preferences Profile) define only syntax for describing device capabilities, allowing vendors to define their own vocabularies. Other initiatives such as SyncML, UA-PROF or Wireless Village Initiative define their own vocabularies. There are of course overlaps between those three vocabularies.

We have chosen UA-PROF as a starting point:

- UA-PROF possesses the richest set of attributes.
- Repositories of hardware devices descriptions in UA-PROF (UA-PROF “profiles”) are already available¹⁰. Partial vocabulary interoperability between the UsiXML platform model and UA-PROF will allow UsiXML users wishing to develop their own platform models to exploit these repositories, which is very valuable since information on hardware capabilities is often hard to collect.
- UA-PROF is CC/PP compliant, and CC/PP is a W3C Recommendation. Actually, UAProf is just a specific CC/PP vocabulary and not an alternative standard.

UAProf (User Agent Profile) is a WAP Forum specification that defines a framework for describing characteristics of WAP-enabled devices. This work is based on WAP-248-UAProf-20011020-a, Version 20 October 2001. Our platform model was built in two phases:

1. Selection of a subset of UA-PROF attributes that are useful for user interface design and adaptation, especially in the framework of graceful degradation. We only use the list of attributes defined in UA-PROF, without taking into account its syntax.
2. Addition of other useful attributes, inspired from the SyncML and Wireless Village Initiative or from our own experience.

Similarly to UA-PROF, a platform description in UsiXML is structured in five components (see Figure 3-6):

- The *Hardware Platform* component contains properties of the device's hardware (category, screen dimensions and resolution, keyboard and pointing devices, colour support...)
- The *Software Platform* component contains properties of the device's application environment, OS and installed software.
- The *Browser User Agent* component contains attributes related to the browser user agent running on the device (browser name and version, supported mark-up and scripting languages...)
- The *Network characteristics* component contains properties describing the network environment (capacity, cost...)
- The *WAP characteristics* component contains properties related to the WAP protocol, if any.

Each component is then described by a set of attributes. A discussion of the attributes included in UsiXML's platform model and a description of these attributes can be found in Annex B. Only a subset of the attributes in UsiXML's platform meta-model needs to be instantiated in a platform specification, depending on the needs.

¹⁰ For example http://w3development.de/rdf/uaprof_repository/ (December 5th 2005)

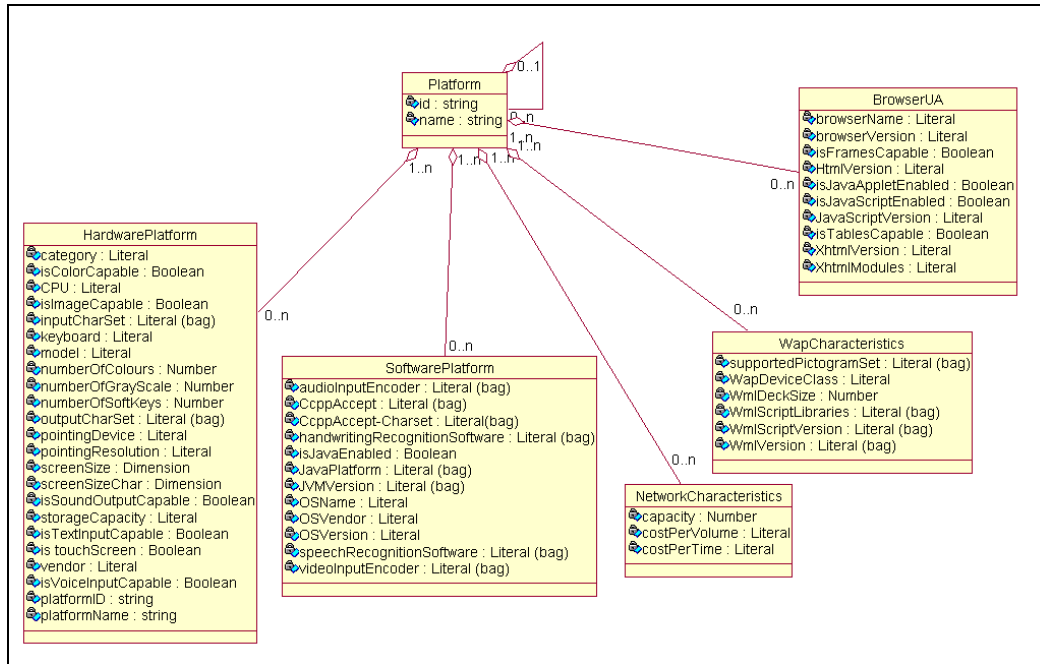


Figure 3-6 UsiXML's platform model

The model is aimed at covering classical platforms: advanced platforms composed of clusters of devices, or using mixed reality are not covered.

3.7 Interactor Model

An instance of an *Interactor model* is a meta-description of a given widget library or user interface description mark-up language. It is linked with one or several instances of the platform model. It describes the components that may appear at the AUI, CUI and Final UI descriptions. In the context of this thesis, this interactor model will be used for detecting the unavailability of a widget on a given target platform and specifying interactor substitutions.

3.7.1 Requirements

The current set of UsiXML metamodels suffers from the following limitations:

- The metamodel of the CUI is too restrictive: it proposes a pre-established set of CIO's, which renders impossible the modelling of a user interface containing other widgets (existing widgets not included in UsiXML, “ad-hoc” widgets created by a designer, or new widgets included in future toolkits).
- Conversely, the description of AIO's is not restrictive enough: the metamodel of the AUI allows describing AIO's composed of any number and type of facets, regardless of the possibility to reify these AIO's into actual widgets. It would be possible, for

example, to specify an AIO composed of 5 input facets, 0 output facet, 2 navigation facets and 1 control facet, which does not correspond to any widget in current toolkits.

- The mapping between CIO's and AIO's has to be explicitly provided by the designer. Now, there exist a limited number of correct mappings, and these possibilities are known beforehand: this information has to be stored and exploited.
- Similarly, the mappings between CIO's/AIO's and the data types/task types they support should be known beforehand, in order to allow the automatic selection and substitution of interactors: the current metamodels only permit deriving this information from the mappings established between CIO's/AIO's and domain concepts or tasks in a specific user interface's description.

To overcome the shortcomings listed above, an interactor model should satisfy the following conditions:

- It should be able to describe any widget, not just a predefined set of widgets.
- It should only allow CIO's and AIO's that possess a reification.
- It should make explicit links between the widgets found at the Final UI level, the CIO's populating the Concrete UI level, and the AIO's.
- It should contain information on
 - (1) The *role(s)* of the interactor i.e., the type of task it is able to support. For example, we would like to know that a button object has the role of “launching a command” and could be replaced by an object sharing the same role (for example, a menu item) on another platform.
 - (2) The *abstract data type(s)* it is able to handle, if any.
 - (3) The *rendering cost* of the interactor [Thev99], i.e. the quantity of physical resources needed for its instantiation. In our approach, the rendering cost consists essentially of the object's width and height on the screen and of the required interaction devices (pointer, keyboard). Other characteristics of the platform such as the RAM memory required could also be used. The object's width and height should not be expressed in terms of constant values, since interactors can often be resized. The rendering cost should be a platform dependent data.
 - (4) An evaluation of the *appropriateness* of the interactor for a given combination of role / abstract data type / platform.

3.7.2 State-of-the-art of meta representations of widgets

3.7.2.a CIO/AIO

Vanderdonckt and Bodart [Vand93] introduced a common way of representing interactors in model-based systems: the distinction between Concrete Interaction Objects (CIO) and Abstract Interaction Objects (AIO).

The CIO's and AIO's described here do not correspond to the CIO's and AIO's defined in sections 3.4 and 3.5. Vanderdonckt and Bodart's CIO's (also called widgets, controls or physical interactors) belong to the Final User Interface, while AIO's belong to the Concrete User Interface. In Vanderdonckt and Bodart's terminology, a CIO is thus a real object belonging to a particular toolkit. Its specification includes:

- (1) Its graphical appearance, determined by the graphical toolbox and the environment. Every change of graphical toolbox / environment may result in a difference in the control appearance, as illustrated on Figure 3-7 and Figure 3-8.
- (2) Its concrete attributes.
- (3) The concrete events it can receive or generate.
- (4) The concrete primitives that describe its reaction to events.



Figure 3-7 Same toolbox / distinct environments: the DatePicker object on Pocket PC (left) and Smartphone (right)

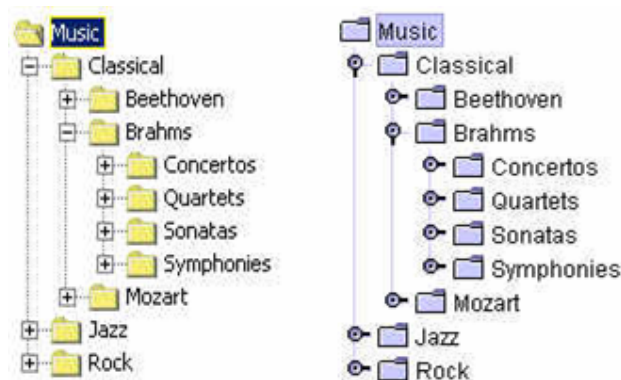


Figure 3-8 Same environment / distinct toolboxes: the TreeView object in Windows MFC (left) and Java Swing (right)

Abstract Interaction Objects (also called logical interactors) are an abstraction of all CIO's sharing the same behaviour, independently of the presentation differences in their target environment. They correspond to Graphical CIO's in UsiXML. They have abstract attributes, events and primitives. They have no graphical appearance, but each AIO is connected to one or more CIO's. For example, the "Button" AIO corresponds to a Button object on Macintosh, to an XmPushButton on X-Windows and to a Command Button on Open Look.

However, the AIO's proposed in [Vand93] do not allow specifying an interface at a high semantic level. For example, using these AIO, it is possible to specify in the presentation model that a Button object has to be used (instead of a XmPushButton or JButton), but it is not possible to declare a presentation object that will launch an application function, independently of the presentation of that object (menu item, button, icon in a menu bar,...)

Categories of AIO have been proposed by the authors:

- (1) Action AIO (e.g. menu, menu item, menu bar, drop-down menu, ...)
- (2) Scrolling AIO (e.g. scroll arrow, scroll cursor, scroll bar, frame, ...)
- (3) Static AIO (e.g. label, separator, group box, prompt, icon ...) A static AIO has no interaction function and thus nor events, neither primitives.
- (4) Control AIO (e.g. edit box, scale, check box, switch, push button, list box, table,...)
- (5) Dialog AIO (e.g. window, help window, dialog box, panel, ...)
- (6) Feed-back AIO (e.g. message, progression indicator, contextual cursor, ...)

However, these categories do not correspond to the roles we would like to represent in our interactor model:

- The classification criteria employed are not well defined and lead to surprising distinctions. For example, why should menu's and buttons belong to distinct categories since they both permit to call an application function? And why should a table belong to the control category and not the static one?
- The objects within a category are not mutually substitutable.

3.7.2.b Teallach's presentation model

The presentation model in the Teallach environment [Grif01] also contains a concrete and an abstract level. The concrete level contains actual widgets, based on Java Swing. The abstract level consists of a set of categories:

- (1) FreeContainer (e.g. window or dialog box). A FreeContainer is a top-level window. It is a container for other presentation items that is not itself contained by anything.
- (2) Container. It represents all containers that are not top-level windows. Typically, a container is used as a grouping mechanism for interaction objects.

- (3) Inputter (e.g. text field). An Inputter is used to transfer data from the user to the application.
- (4) Display (e.g. read-only table). A Display is used by the application to transfer data to the user.
- (5) Editor (e.g. updateable table). An Editor allows the user to alter existing data.
- (6) Chooser (e.g. a list box). A Chooser allows the user to perform a choice among a set of alternatives.
- (7) ActionItem (e.g. a button or menu item). An ActionItem allows the user to initiate some behaviour of the system.

Contrary to the categories proposed in [Vand93], the entities in Teallach's abstract model can be used to specify a user interface without commitment to specific presentations, allowing developers to postpone decisions on the actual widgets to be used until later in the design process. However, the typology is not detailed enough for our purpose, since only some substitutions between objects of the same category are possible. For example, a group of check boxes allowing the user to select an undetermined number of options can not be replaced by a group of radio buttons, which would restrict the user choice to only one option, even when both objects belong to the "Chooser" category.

3.7.2.c ACE's Selectors categories

An abstraction mechanism similar to Teallach's was developed earlier in the ACE system [John92]. ACE (Application Construction Environment) offers Selectors and Presenters as an alternative to widgets.

There are two subclasses of Selectors:

- The *Data Selectors*, that have mappings with elements of the domain (they display and set application variables).
- The *Command Selectors*, that display and invoke system operations.

Data Selectors encapsulate the semantics of choice. Each Selector is defined by

- A base-type i.e. the data type it can contain (e.g. Integers, Colours, ...)
- A domain i.e. a restriction on the values of the base-type that define a set of options. A domain can be specified by explicit enumeration (e.g. {red, green, blue}), by a range (e.g. [10..50]) or by a predicate function (e.g. Odd(n)).
- A value, that indicates the minimum and maximum number of possible choices among the options.

Data Selectors are given an appearance by a Data Selector Presenter. A Data Selector can be presented by multiple Presenters at the same time. The Presenter manages the layout, the look and the behaviour of the displayed objects, but the content of the object (option list, etc.) is managed by the Selector.

Command Selectors, on the other side, provide invocation access to a set of related commands. A Command Selector has:

- A domain (a list of related commands).
- A value that indicates which of the Selector's commands are active.
- A minimum and maximum number of commands that can be activated at the same time.

Command Selectors are given an appearance by Command Selector Presenters.

The Selector approach permits to compose interfaces at a semantic level, as it was the case with Teallach's abstract categories. It presents the additional advantage to connect explicitly the presentation objects to application semantics (domain values and operations linked to the commands). On the other side, all examples in the source paper are related to ActionItems or Choosers (in the Teallach terminology) and it is unclear whether Selectors are also an appropriate abstraction mechanism for objects in the other categories (Inputter, Display ...)

3.7.2.d ArtStudio's interactor model

The interactor model of ArtStudio (see section 2.2.4.h) was developed in a perspective of multiplatform generation. In ArtStudio, an interactor is described in terms of its *representational capacity*, *interactional capacity* and *usage cost*.

The *representational capacity* of an interactor corresponds to the data type it is able to render. ArtStudio defines two categories of interactors: the presentation interactors, that have a representational capacity, and the navigation interactors, that do not represent concepts but are systems composed of a view on a navigation spaces and control objects on that navigation space.

The *interactional capacity* of an interactor are the tasks that this interactor is able to support (e.g., specification or selection).

Finally, the *usage cost* of an interactor is expressed as the graphical footprint of the interactor on the screen (the display area it requires).

ArtStudio's interactor model satisfies most of our requirements:

- ArtStudio's interactors have toolkit independent definitions.
- Their role is represented by the task type(s) they can support. The few task types proposed by ArtStudio do not permit to define correct substitutions between interactors (the only types listed are consultation, specification, selection and activation), but this typology can be extended (see section 3.7.3).
- The abstract data type(s) the interactors can handle is specified when appropriate.
- The rendering cost of the interactors is expressed by the display area they require on the screen.

3.7.2.e The Comets

A new type of widgets has been introduced lately under the name of *comet* [Calv04] [Calv05]. A comet is a new type of widget specially designed to support software plasticity, i.e. the capacity of an interactive system to withstand variations of context of use while preserving its quality in use.

Comets are defined as *self-descriptive widgets*, able to export their *functional* and *non functional properties*. The functional properties, which correspond roughly to ArtStudio's concepts of representational capacity and interactional capacity, include the tasks the comet can support (e.g. selecting an option among a set of options) and the concepts it can manipulate (e.g. a banking account). The potential capacity of a comet to adapt itself is also considered as a functional property. Non functional properties are qualities the comet guarantees in a given interaction context. They may also refer to the properties of the comet's auto-adaptation process (e.g., continuity of the interaction) and they may be expressed in any reference framework (ISO, IFIP ...)

More sophisticated comets own the additional capabilities of *polymorphism* (they contain several presentations) and *self-adaptation* (they contain all the mechanisms necessary to adapt themselves i.e. they can recognize the context, calculate a reaction and apply it).

A comet can be defined at *different levels of granularity*: the functionalities it can cover range from functionalities of classical widgets (e.g. a button) to functionalities of a specific interactive system (e.g. viewing the time at two different locations).

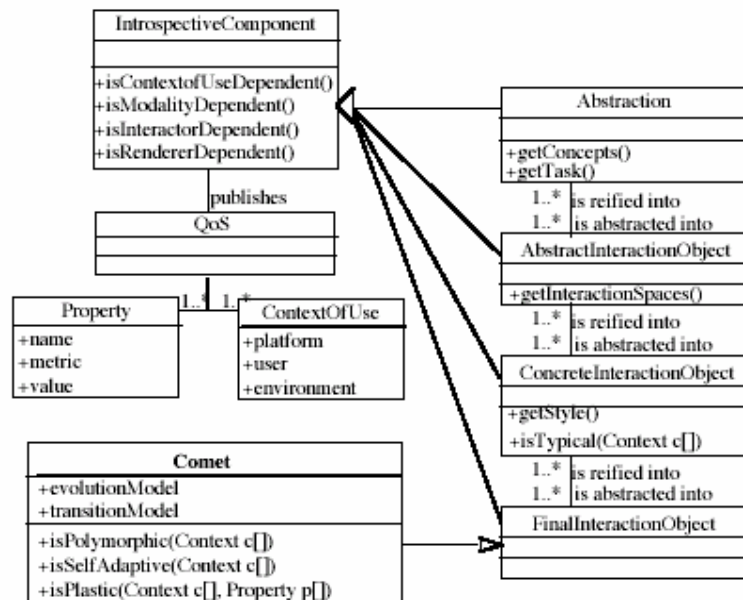


Figure 3-9 Modelling of comets (from [Calv04])

The description of a comet (see Figure 3-9) is structured in four abstraction levels, following the Unified Reference framework (section 2.1): Tasks&Concepts (description of the concepts and user tasks supported), Abstract User Interface level (description of the comet in terms of interactive spaces), Concrete User Interface level (description of the “style” of the comet and assessment of whether it is typical or not to use this style in a given context of use) and Final User Interface level (detects and communicates context changes and manages the comet’s interaction state). Each level publishes its quality of use for a given pair of property / context of use.

Like ArtStudio, comets have thus an explicit representation of the concepts and tasks supported. The notion of quality of use subsumes the concept of usage cost in ArtStudio, as it can be characterized by any property in any set of ergonomic recommendations. In contrast, in the context of graceful degradation, a widget’s quality is not only determined by the context of use, understood as a triplet user-platform-environment, but is also influenced by characteristics of the particular user interface to which it belongs (for example, whether it is densely populated or not, ...) In GD, the appropriateness of a widget should be calculated for a given concrete user interface, and not directly stored in the toolbox.

As a major characteristic, comets are auto-descriptive. This feature could be useful if we wished to apply graceful degradation rules (such as widget substitution rules) on the fly, at run-time. When considering only the application of the rules at design time, we can rely on an external description (a meta description of the widget library, not embarked in the toolkit itself).

3.7.3 Interactor model in the GD approach

Our own description of interactors is based on ArtStudio and shares some features with basic comets, although our own model is not meant to be used at run-time, but will serve to describe final interactors available in toolkits or mark-up languages.

Like in the comets' approach, we will describe interactors at distinct abstraction levels. Our description has no abstract interface level: we have considered interactors as no splittable units. Our description is thus structured in three levels only:

- (1) The higher level (corresponding to the Tasks&Concepts level for comets) describes *Extended AIO*'s. Like Abstract Interaction Objects defined in section 3.4, Extended AIO's are independent from the modality and composed of facets, but their description contain details impossible to express using only the four facets described above. Examples of Extended AIO's are a Number Inputter, a Date Chooser, an Action Item, a Separator or a Container.
- (2) At the second level (which would be the concrete interface for comets), the *Graphical CIO*'s are a reification of the Extended AIO's for GUIs. They correspond to the Graphical Concrete Interaction Objects defined in section 3.5. Examples of graphical CIO's are an edit box, a radio button or a list box.

3. Language and Models

The third level (final user interface) represents the actual widgets, or *Final widgets*. This level is platform dependent and modality dependent.

Figure 3-10 shows a detailed representation of the concepts involved in our interactor model. Let's review the concepts of that diagram.

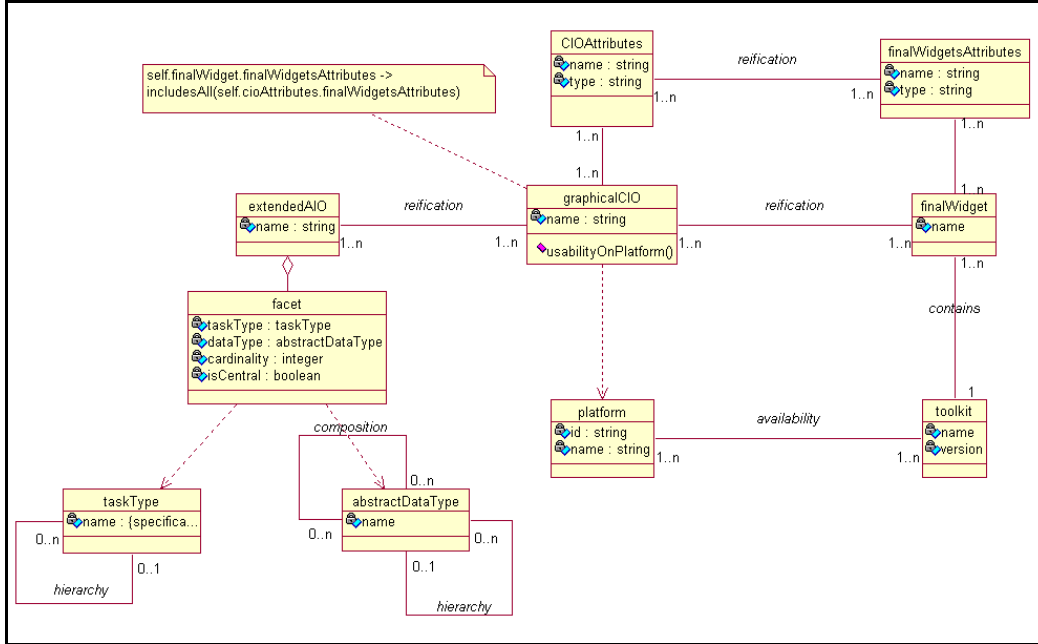


Figure 3-10 Class diagram of the concepts involved in our interactor model

Extended AIO's are described by their *name* and composed of *facets*.

The *names* are Teallach-based, with extensions. Examples of names for a-modal AIO's are:

- For the interactors with an interactive role (i.e. attached to a task type): Editors, Choosers, Inputters, Choosers, Displays, Internal Navigator, Action Items, ..., together with the attached data type if any (e.g. Text Inputter, Date Inputter, ...)
- For the static interactors (i.e. not attached to any task type): Separators ...

Facets are described by the *task type(s)* they support, the *abstract data types* they can handle and a *cardinality*. Some extended AIO's are composed of several facets (e.g. a text editor is composed of a specification facet and to a consultation facet at the same time). Other extended AIO's (e.g; Separators) do not possess facets at all, because they neither permit to realize an interactive task, neither present data.

The *task types* are structured into a hierarchy of inheritance, shown on Figure 3-11. A facet supporting a given task type can also support the descendants of that task type (e.g. a facet that supports a specification task can also support a selection task or a unique selection task). This property is important if we consider the possibility of interactor substitution in a GD process.

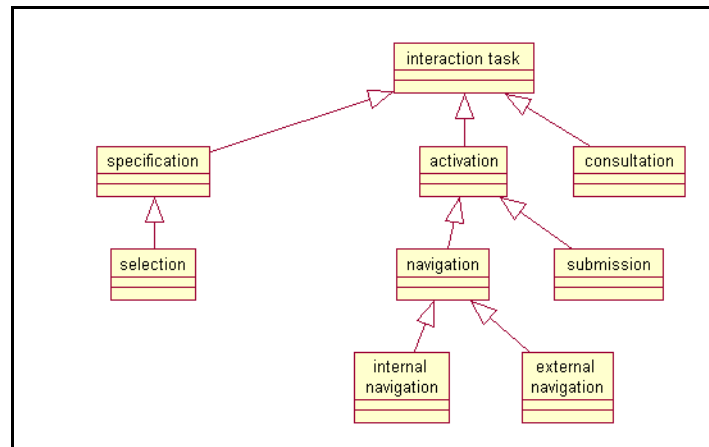


Figure 3-11 Hierarchy of task types

The *abstract data types* are also structured in a hierarchy. Such as for tasks, a facet that supports a given abstract data type can also support the descendants of that data type (e.g. a facet that can represent text can also represent a city, if we consider “Cities” to be a subclass of “Text”). Some abstract data types can be considered as an aggregation of several simpler data types (e.g. a date is the composition of a month, a day and a year).

The last attributes of a facet are its *cardinality* and its *centrality*. The cardinality indicates the number of concepts the interactor can handle. Interactors that have no representational function because they are not linked to any abstract data type have no value for this attribute. The centrality is a Boolean attribute indicating if the facet is a central component in the interactor's description or not. For example, a Chooser is composed of a selection facet, which is central, and a consultation facet, which is secondary.

At the second abstraction level, Graphical CIO's are the abstractions of widgets predefined in the toolkits. They are described by their *name* and are linked with a list of *CIOAttributes*. They have a *usability* on a given platform.

The *names* correspond to the subtypes of **graphicalIndividualComponent** and **graphicalContainer** defined in UsiXML's CUI model (section 3.5). For the graphical CIO's not included yet in UsiXML, the names are based on the terminology developed in [Vand98]. Examples of names are ScrollBar, MenuItem, Button, or CheckBox.

The *CIO attributes* contains the generic attributes of the graphical CIO. A generic attribute is the generalization of a property of the final widgets linked with that CIO, independently of the actual names and particularities of these attributes at the Final User Interface level. Examples of attributes are the object's colour, label, etc.

A graphical CIO should be able to evaluate its *usability* on a given platform. For example, the usability of an Edit box is higher on a platform that is equipped with a physical keyboard than on a platform that has only a virtual keyboard, and the usability of menu's and scrollbars is lower on an electronic white board than on a workstation, because the user can not reach them easily from any position. A CIO's usability depends also on user

characteristics and preferences or on the context of use (Was the user trained in touch typing? Is he/she standing or sitting? Will he/she have the possibility to use both hands?) However, some general recommendations can be made for an average, typical user, in a typical context of use. For example, a general recommendation could be to prefer pop-up pie menus to classical pull-down menus on very large displays [Jaba03] or to prefer selection to text entry when only a phone keypad is available.

At the Final User Interface level, toolkit-dependent *Final widgets* are characterized by a *name* and a list of final attributes. They belong to a given *toolkit*, which is available on one or several *platforms* (see 3.6).

The interactor model described above will allow us to perform interactor substitution in a graceful degradation process, as will be shown in section 4.1.2.b. In order to assess the validity of this model, we have used it to represent the widgets of a toolkit of limited size: QTK (a Tcl/tk based toolkit linked with the Oz language). The resulting meta-description of QTK can be found in Annex C.

3.8 Mapping Model

UsiXML's mapping model is a collection of links, named *mappings*, between elements of the interface specification. UsiXML proposes a non exhaustive list of mappings, illustrated on Figure 3-12:

- (1) *isReifiedBy*: relationship between elements of a model $\mathcal{M}_i$ and a model $\mathcal{M}_j$, where $\mathcal{M}_i$ has a lower level of abstraction than $\mathcal{M}_j$. Links an abstract object (for example an AIO) to the more concrete object that reifies it (for example, a CIO).
- (2) *isAbstractedInto*: inverse of the *isReifiedBy* relationship.
- (3) *isTranslatedInto*: relationship between elements of a model $\mathcal{M}_i$ and a model $\mathcal{M}_j$, where $\mathcal{M}_i$ and $\mathcal{M}_j$ are at the same abstraction level, but belong to distinct descriptions targeted to distinct contexts of use.
- (4) *manipulates*: relationship between a Task Model and a Domain Model. Maps a task to a domain concept (class, object or attribute).
- (5) *isExecutedIn*: relationship between a task model and an Abstract User Interface or Concrete User Interface. Links a task and the interaction object (CIO or AIO) allowing its execution.
- (6) *observes*: relationship between a Concrete/Abstract User Interface and a Domain Model. Links an interaction object to the domain concept it permits to observe.
- (7) *updates*: relationship between a Concrete/Abstract User Interface and a Domain Model. Links an interaction object to the domain concept it permits to update.

- (8) *triggers*: relationship between a Concrete/Abstract User Interface and a domain model. Links an interaction object and the method of the Domain Model this interaction object triggers.
- (9) *hasContext*: relationship between any User Interface model (Task, Domain, AUI, CUI) and the Context Model it is related to.

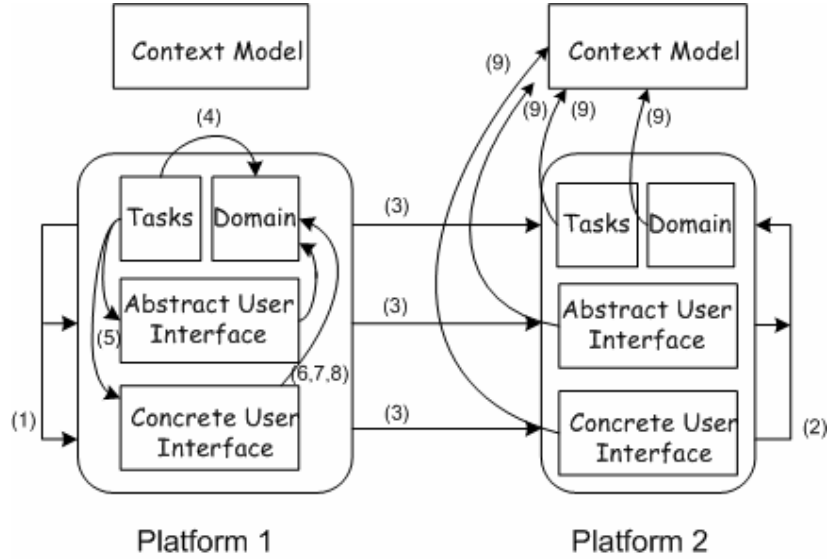


Figure 3-12 Mappings in UsiXML

3.9 Conclusion

One major characteristic of the UIDL described above is its hierarchal structure in four abstraction layers. The different models belonging to each abstraction layer (Task, Domain, AUI, CUI) were defined independently of this thesis [Limb04][Limb04b], whereas UsiXML's platform model represents a contribution of this doctoral research. Because we have identified some limitations in the descriptions of interactors in UsiXML (3.7.1), we also propose an interactor model, which could be used to enrich the description of UsiXML's graphical CIO's and AIO's or as an alternative construct in a future version of the language.

The language and models defined in this chapter will be used through the remainder of this thesis. The GD rules presented in Chapter 4 will be expressed in UsiXML. The splitting rule described in the next chapter will take advantage of UsiXML's capacity to freely combine models at different abstraction levels. A tool performing semi-automatic transformations on UI specifications in UsiXML will be introduced in Chapter 6.

Chapter 4 Effective knowledge for Graceful Degradation

After the state of the art of development approaches for multiplatform systems of Chapter 2, we have shown that there is a place for an approach that would realize a trade-off between development and maintenance costs on the one side and level of control, completeness, usability and guidance on the other side. We have called that approach Graceful Degradation of user interfaces. The two basic ingredients of Graceful Degradation are a UIDL, used to specify source and target UI, and a set of transformation rules, used to adapt the source UI to more constrained platforms. Chapter 3 consisted in the presentation of the chosen specification language: UsiXML. Chapter 4 will present a collection of GD rules, classified using the Unified reference framework presented in section 2.1 and formalized in UsiXML.

4.1 Typology of rules using the Unified Reference Framework

The basic ingredient of the GD approach is a collection of transformation rules [Flor04b]. Graceful Degradation rules have been identified by the observation of the user interfaces of a large number of applications running on several devices, most of them publicly available on the Web or described in scientific publications. The observed transformations were then abstracted and logically organized in terms of the CAMELEON framework.

The rules are intended for two uses: semi-automatically transforming user interface specifications in UsiXML, and manual application by human designers. For this reason, this typology is not limited to rules actually implementable when working with models in UsiXML.

In the terms of the Unified reference framework, GD is a translation process. The GD rules are thus transformations between a source model and a target model at the same abstraction level. In this section, we present a classification of GD rules according to:

- (1) Their *abstraction level*: GD can be applied at different abstraction levels in the Unified reference framework, namely the 'Tasks&Concepts level, the Abstract Interface level, the Concrete Interface level and the Final Interface level;
- (2) Their *source element(s)* in the layer they belong to: GD can be applied on distinct elements of a given layer (for example, on Graphical CIO's or on graphical relationships in the Concrete User Interface).

The abstraction level of a rule is the highest level in the hierarchy of models of the reference framework where the rule can be attached. Of course, every rule applied at a high abstraction level in the framework has an impact on lower levels: if we delete a task in the task model, this change would be propagated at the AUI, CUI and FUI levels when coupling GD translation with a generation process. For this reason, we begin by presenting rules belonging to the lowest abstraction level.

4.1.1 GD rules at the Final User Interface level

Ultimately, each GD rule will result in a modification at the code level (the Final User Interface). A lot of transformations can also be applied directly on the code, without need of any higher level description. These rules are generally platform specific and do not require a model-based approach to be developed. Nearly every non model-based technique aimed to adapt the UI to a platform (in transcoding tools for example) is a transformation rule at the final UI level.

Some transformations at the Final User Interface level are impossible to express in our framework: they have no impact on higher levels and, in a model-based approach; the Final User Interface is generated, not specified. These transformations are thus beyond our scope. For example:

- The substitution of an image by an image in a compressed format (e.g. substituting a BMP by a JPEG).
- The reduction of the colour number
- Zooming techniques

Other transformations can be applied on a higher level description of the UI (they will be included in the classification below), but can just as well be applied at the code level:

- Removal of images
- Reduction of font sizes
- Interactor substitution rules or moving rules when directly expressed between two specific platforms
- etc.

4.1.2 GD rules at the Concrete User Interface level

As explained above, The Concrete User Interface level (CUI) is a detailed specification of the appearance and behaviour of the UI's elements. A CUI is populated by Concrete Interaction Objects (CIO's) and Concrete User Interface relationships. In the case of graphical user interfaces, a CUI is restricted to *graphical CIO's* and *graphical relationships*.

Two important kinds of GD rules can be applied at the CUI level:

- Rules that transform the graphical relationships between the graphical CIO's (graphical relationship transformations)

- Rules that modify the size, number and nature of the graphical CIO's (graphical CIO's transformations).

4.1.2.a Graphical Relationships Transformations

There are two types of rules that can be applied to graphical relationships:

1. Reorientation rules.
2. Moving rules.

4.1.2.a.1 Reorientation rules

Reorientation rules modify the orientation of an object without other change in size or position. They are mainly useful when switching from landscape to portrait mode or conversely. They can only be applied to a small set of objects (tables and table labels, sliders, toolbars...) Figure 4-1 shows an example of a reorientation rule applied to an accumulator widget (i.e. a component transferring items from the left list of possible values to the right list of accumulated selected items).

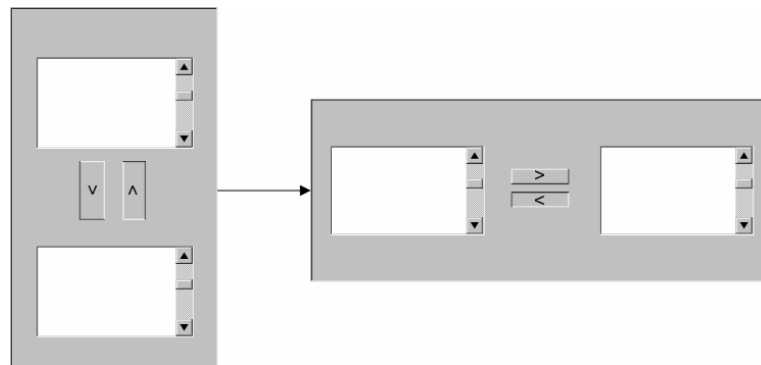


Figure 4-1 Reorientation rule

4.1.2.a.2 Moving rules

Moving rules modify the localization of a graphical object, i.e. the position of the object in its container, defined by the graphical relationships where it is involved. Moving rules are useful in several cases:

- The components do not fit in one dimension (horizontally or vertically) when there is blank space left in the other dimension.
- The components do not fit horizontally and we want to avoid horizontal scrolling.
- Some ergonomic rule or convention of the target platform has to be respected (for example, menus on an IPaq should better be placed on the bottom of the screen, and on a wall display, should not be placed out of reach of a short person).

4. Effective knowledge for Graceful Degradation

Repositioning graphical objects vertically is especially useful, since a majority of users dislike horizontal scrolling, considered as a tedious and disorientating task (Palm OS does not even support horizontal scrolling at all). For example, repositioning elements in one or more columns or placing graphical objects beneath their labels instead of to the right of their labels are widely used transformations. A good illustration of the use of moving rules is the Skweezer¹¹ Web service, which reformats Web pages in order to avoid left-to-right scrolling when accessing the Web with a handheld device. Figure 4-2 shows the result of the application of a moving rule on our research project Web page.

**UCL** Belgian Laboratory of
Computer-Human Interaction
ESPO/AG/ISYS - BCHI

Title
SALAMANDRE



Acronym
A salamandre is an animal showing some capabilities to adapt itself to the environment.

Duration
2001-2005

Funding
Initiatives III Research Program, DGTRE, Ministry of Walloon Region

Goals
SALAMANDRE is interested to produce multiple User Interfaces (UIs) for different platforms. Our methodology, the *Graceful Degradation Approach*, is based on a set of rules that modify the UI specifications at different abstraction levels. Guidelines related to different computing platforms will be gathered in a database. In collaboration with the Telecommunications Unit (TELE), mobile and multi-platforms UIs are considered in a case study in telemedicine.

Université Catholique de Louvain | Information System Unit | School of Management | Faculty of Economical, Social and Political Sciences | kchi

Title
SALAMANDRE



Acronym
A salamandre is an animal showing some capabilities to adapt itself to the environment.

Duration
2001-2005

Funding
Initiatives III Research Program, DGTRE, Ministry of Walloon Region

Goals
SALAMANDRE is interested to produce multiple User Interfaces (UIs) for different platforms. Our methodology, the *Graceful Degradation Approach*, is based on a set of rules that modify the UI specifications at different abstraction levels. Guidelines related to different computing platforms will be gathered in a database. In collaboration with the Telecommunications Unit (TELE), mobile and multi-platforms UIs are considered in a case study in telemedicine.

Figure 4-2 Automatic application of a repositioning rule by an on-line reformatting service

¹¹ Trade Mark of Greenlight Wireless Corporation (<http://www.greenlightwireless.net/skweezer/>)

4.1.2.b Graphical CIO's Transformations

Beside GD rules that transform the graphical relationships between objects, another type of transformation can be applied at the Concrete User Interface level, namely modifications in

- The size of the graphical objects.
- The nature of these graphical objects. Object transformations can take three different forms: modification, substitution and removal.

4.1.2.b.1 Resizing rules

Resizing rules modify the dimensions of a graphical object. Theoretically, they could be applied to any UI component, but we have to take into account:

- The type of the graphical CIO: some interactors have fixed dimensions in most of the toolkits where they have been implemented (e.g. a radio item) while others may generally be resized (e.g. a button).
- The constraints imposed by the toolkits: a lot of toolkits do not allow the designer to give arbitrary dimensions to the widgets: for example, widgets in languages like HTML or QtK are automatically given the necessary size.
- The limits of human perception: for example, experimental usability results establish that an icon cannot be shrunk below the threshold of 8 x 7 pixels. Beyond this, it becomes illegible or impossible to distinguish.

When a component can be resized, the designer has to know the minimum width and height it can be shrunk to. For some widget types, the minimum width / height is influenced by factors that will only be determined at design time for a given application: e.g. the minimal width of a listbox depends on the length of the larger proposed choice, the minimal width of a button depends on the length of its label, etc.

Some resizing transformations are impossible to apply on a CUI expressed in UsiXML, since the language does not permit expressing information on the size of all graphical CIO's. So, resizing a button has to be applied at the FUI level, when allowed by the graphical toolkit. However, transformations such as reducing the size of text components (width of edit fields, height and width of text fields), limiting the number of items visible in a listboxes, modifying the size of fonts or resizing an image are easy to express in UsiXML. Some transformations may combine resizing and deletion of non central information: Figure 4-3 for example shows the result of applying scaling and cropping to a picture.

Note that “resizing” does not always mean “shrinking”, even in a GD context. For example, when the target platform must support touch interaction with the finger, or possess a lower pointing resolution, some graphical objects may need to be enlarged.



Figure 4-3 Resizing rule applied in combination with image cropping

One interesting method to address resizing has been proposed by Dragevic et al. [Drag05]. Their technique, called “artistic resizing” allows designers providing various versions at a given graphical object, at different key sizes, using their usual drawing environment. The system then interpolates the resizing behaviour of the object between these key sizes, while taking into account the need to apply deformations to objects in order to create an illusion of smooth resizing.

4.1.2.b.2 Modification rules

Modification rules act upon the appearance of a graphical object. The physical rendering of a semantic feature can be modified (e.g. the notion of ‘emergency’ could be represented by the red colour on a workstation and by a flickering on a mobile phone), or the font of a text, or the colour of an object.

Modification of the fonts is sometimes imposed by the fonts available on the target platform: mobile devices often own a very limited set of native fonts.

Similarly, modification of the colours is unavoidable when a smaller number of colours or grey scales is supported by the target platform. Furthermore, ergonomic guidelines fix a maximum number of colours per screen, in order to improve legibility. Switching colours also permits respecting platform specific colour conventions (linked to a given operating system, for instance).

4.1.2.b.3 Substitution rules

Substitution rules replace a graphical CIO by an alternate interactor that enables the same type of functionalities. A substitution rule can be activated for two reasons:

- *Unavailability*: when a graphical object is no longer available on the target platform, it has to be replaced by another one which is available on the target. For example, check boxes and radio buttons, non existing in WML language for mobile phones, are replaced by a list, as illustrated on Figure 4-4.

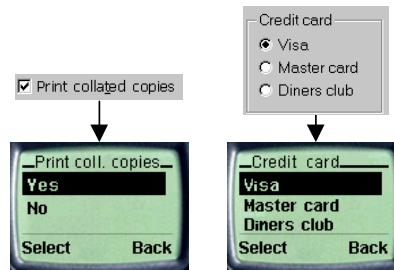


Figure 4-4 Component replacement due to unavailability

- Screen size inadequacy:** when a graphical object does not fit in the target platform because it takes too much screen size and when it is not possible anymore to resize it, it should be replaced. For example, Figure 4-5 shows possible substitutions for an accumulator, an interactor transferring items from the left list of possible values to the right list of accumulated selected items, thus allowing multiple selection among a closed list of items. In a first stage, the accumulator can be replaced with a smaller version of the same object (with the transfer buttons labels being reduced). In a second stage, when the accumulator is no longer affordable, the use of other interactors supporting multiple selection tasks has to be considered: a group of check boxes, a list box containing check boxes, a simple listbox or a list restricted to merely one item in the extreme case. Similarly, Figure 4-6 shows a set of substitutions for a simple choice task.

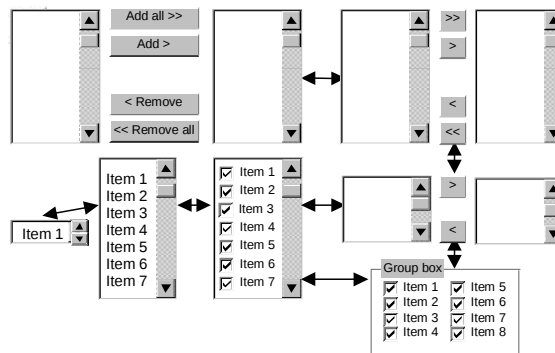


Figure 4-5 Candidate interactors for multiple choice

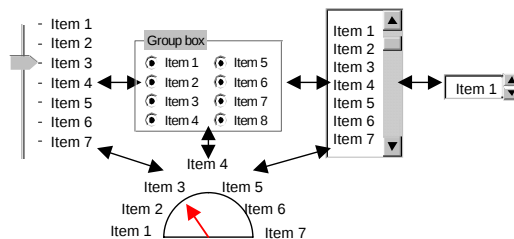


Figure 4-6 Candidate interactors for simple choice

Different types of substitution can be performed:

- *Simple substitution* ($1 \rightarrow 1$): a graphical object x on the source platform is replaced by a graphical object y on the target platform.
- *Merging* ($N \rightarrow 1$): a set of graphical objects on the source platform is replaced by a single graphical object on the target platform. E.g. a set of check buttons can be regrouped into an accumulator.
- *Splitting* ($1 \rightarrow N$): a single graphical object on the source platform is replaced by a set of graphical objects on the target platform. E.g. a tabbed panel could be replaced by a set of hyperlinks.
- *Composite substitution* ($N \rightarrow M$): a set of graphical objects on the source platform is replaced by another set of graphical objects on the target platform. E.g. a group of N action buttons is replaced by a menu and N menu items (for example, if the methods launched by the buttons correspond to subtasks of the same mother task in the task model, or belong to the same class of the domain model). Figure 4-7 shows an example of composite substitution that replaces a sequence of edit fields and their associated labels by a single edit field and a combobox permitting the selection of each of the labels in turn.

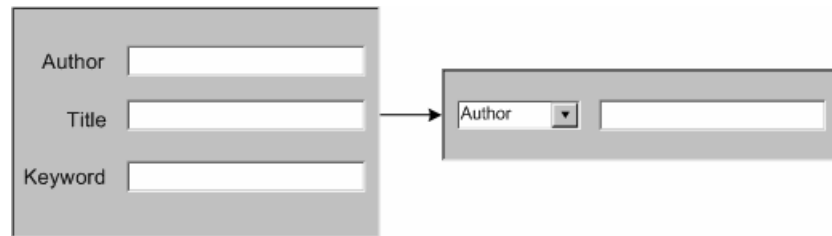


Figure 4-7 Interactor substitution: example of composite substitution

Not all alternatives have the same usability in a given context:

- Not all interactors are as easy to manipulate on a given platform. For example, a checkbox is difficult to select on touch screen platforms because of the dimension of the fingers.
- Some interactors offer better visual guidance for a given type of task. For example, an accumulator clearly denotes a multiple selection task whereas a simple listbox does not indicate whether multiple choice is allowed nor how to achieve this task. Only experienced users will know that they have to press a special key in order to select multiple items.
- Depending on the number of available choices, some interactors seem to be more appropriate than others. For example, ergonomics rules generally state that a group of checkboxes should be limited to 7 items in order to optimize the legibility, whereas an accumulator is perfectly suitable for higher cardinality.

4. Effective knowledge for Graceful Degradation

As mentioned above, interactor substitution rules can also be defined at the FUI level. However, rules applied on a FUI are platform specific: a separate set of transformations has to be defined for each pair of platform. On the other hand, the use of a model-based approach allows us to define platform independent substitution rules. These substitution rules presuppose the availability of a description of the interactors on the source and target platform following the format we have proposed in section 3.7.3.

The rules are to be applied as a 2 step process:

- (1) When a given graphical CIO is not available or suitable on a given platform, select a graphical CIO linked with the same extended AIO in the interactor model. For example, substitution of a Button by a MenuItem: these graphical CIO's are both linked to the Extended AIO “ActionItem”.
- (2) When there is no suitable graphical CIO linked to the same Extended AIO, select a CIO linked to another Extended AIO that supports:
 - A supertype of the original task type e.g., an Extended AIO supporting a selection task can always be replaced by another Extended AIO supporting a specification task (see Figure 4-8).

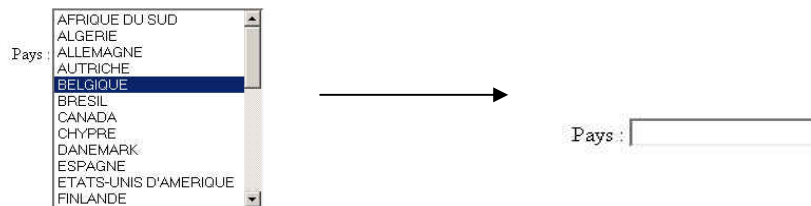


Figure 4-8 Substitution rule (1)

- And/or a supertype of the original abstract data type e.g., if we consider that a season is a subtype of the string data type, then we can replace a Season Selector by a Text Selector (see Figure 4-9).



Figure 4-9 Substitution rule (2)

- And/or data types corresponding to the decomposition of the original data type into simpler data types (and conversely) e.g., substitution of a Date Inputter by a Month Inputter, Day Inputter and Year Inputter (Figure 4-10). This kind of substitution has of course more visual impact

on the UI and is more likely to disturb users switching from the version of the interface on the source platform to the version of the interface on the target platform (less cross-platform consistency).

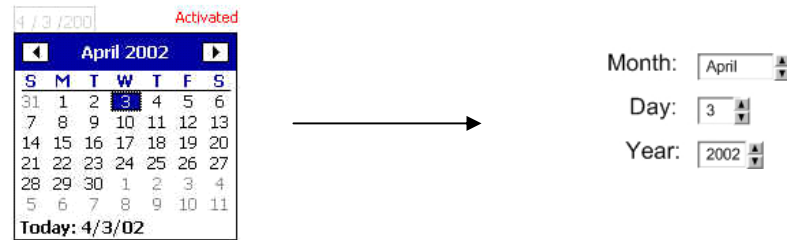


Figure 4-10 Substitution rule (3)

Both the mapping of the CIO to the domain model and the behaviour specification are kept constant.

4.1.2.b.4 Removal rules

Finally, the last type of GD rule that can be applied to graphical CIO's are *removal rules*, that merely delete a graphical object, due to space constraints on the target platform (e.g., removal of pictures on a mobile phone).

4.1.3 GD rules at the Abstract User Interface level

An Abstract User Interface defines grouping of tasks that have to be presented together, in the same window or page for example. In UsiXML, the AUI is populated by Abstract Containers and Abstract Individual Components.

Let's consider the simple example of an information retrieval system (Figure 4-11). The system's task structure is composed of two sequential subtasks. The first subtask is an interactive task consisting of the insertion of the search criteria. The second subtask is an interactive task consisting of viewing the results of the query. We can imagine several ways to map these interactive tasks to Abstract Containers:

- Both tasks mapped to the same Abstract Container: the Abstract Individual Components presenting the “Insert search criteria” task and the “View results” task are grouped together.
- Each task mapped to a distinct Abstract Container: the Abstract Individual Components presenting the “Insert search criteria” task and the “View results” task belong to separate containers.
- The first task mapped to a first Abstract Container, plus a mapping between both tasks and a second Abstract Container: the first container contains the Abstract Individual Component(s) presenting the “Insert search criteria” task, while the second container presents the “Insert search criteria” task and the “View results” task at the same time.

Figure 4-11 shows two representations for these Abstract Containers. The graphical representation on the right was produced using the graphical editor for UsiXML included in the IdealXML tool [Mont05]. This representation shows Abstract Containers and Abstract Components, with their facets. The representation on the left is a shortcut notation that shows the Abstract Containers thanks to outlines drawn directly on the task model. If we consider that a given task is always mapped to the same Abstract Component or set of Abstract Components in the AUI model, the two notations are equivalent.

When there are big differences between the platforms constraints (e.g. big differences in screen size and resolution), it will not be possible to maintain the same distribution of tasks among Abstract Containers between the system versions. For this reason, the most useful GD rules at the AUI level split the source Abstract Container into two or more Abstract Containers on the target platform. We call these rules *splitting rules*.

A possible side effect of the application of a splitting rule is the introduction of internal redundancy within distributed versions of a system: a task mapped to one single Abstract Container on the source platform could be mapped to two or more containers on the target platform. Figure 4-12 shows an example of internal redundancy caused by a platform change: the single “cancel” component on the source platform has to be duplicated on the target platform.

Besides splitting rules, another possible adaptation technique at the Abstract UI level is the *reorganization of tasks within the same Abstract Container* (modification of the spatio-temporal relationships between the Abstract Individual Component mapped to these tasks). The reason for internal permutation between tasks can be that we want to present tasks in the order of frequency of each task and that we expect that a task frequency will change on the target platform (e.g. the consultation of an address book could be more frequent on a mobile phone than on a workstation).

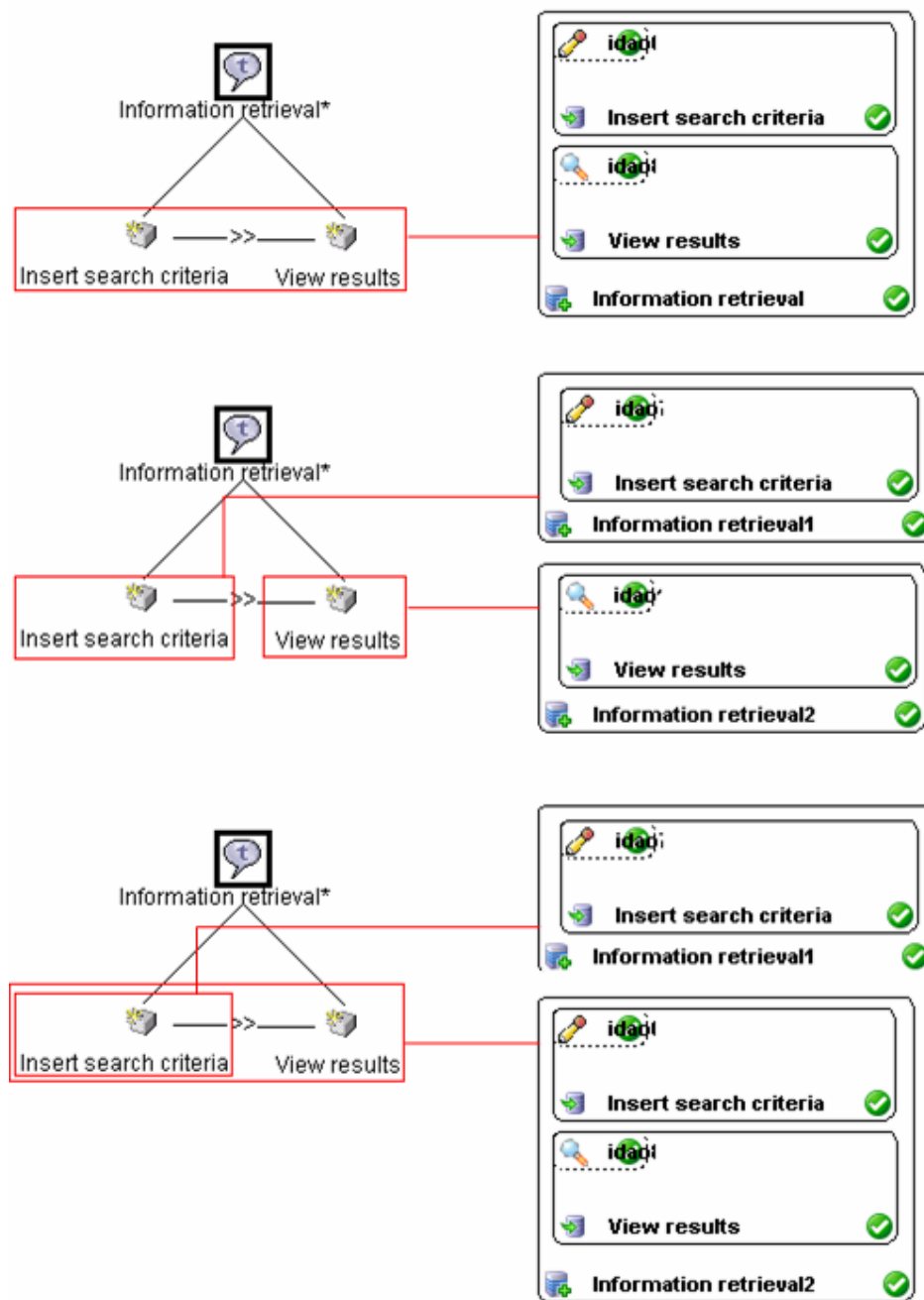


Figure 4-11 Defining Abstract Containers for a simple information retrieval system

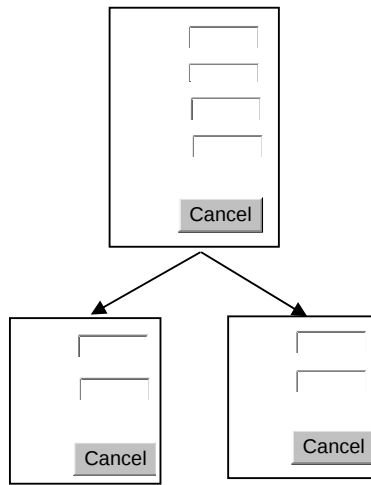


Figure 4-12 Internal redundancy due to splitting rule

4.1.4 GD rules at the Tasks & Concepts level

At the Tasks & Concepts level, GD rules can be applied to general functionalities (high level tasks, that correspond to the user's general goals), to the procedures that the user must follow in order to achieve his/her general goals (low level tasks), to the temporal ordering between tasks and to the concepts.

4.1.4.a High Level Task Deletion

A high level task present on the source version may be removed from the target version, for different reasons:

- The task implies interaction capabilities that are unavailable or inappropriate on the target platform (e.g. tasks involving video streaming or manipulation of complex graphics are impossible to perform on a cellular phone, so are tasks of data storage when the quantity of data is huge).
- The task requires resources that are very scarce on the target platform so that the interaction could be interrupted due to lack of resources (e.g. a task manipulating an object requiring much RAM memory).
- The task requires such interaction capabilities that carrying out this task on the target platform could become very tedious (e.g. a task of word processing on a PDA, although partially possible, rapidly becomes impractical due to the limited entry capabilities like virtual keyboard or character-recognition).
- The typical context of use of the target version is inappropriate to the performance of that task (e.g. a task of graphical edition is inappropriate in a context where the user will be standing, when the target platform is an interactive kiosk for example).

4.1.4.b Procedure Transformations

Another type of transformation at the Tasks & Concepts level affects the subtasks necessary to achieve the same general functionality. Two types of GD rules modify a system subtasks: subtasks deletion rules and subtasks insertion rules.

4.1.4.b.1 Subtask Deletion

Subtasks can be deleted for different reasons:

- Some subtasks are unnecessary on the new platform (e.g., on a platform with a GPS system, it is no longer necessary to specify the user's location).
- Some subtasks require too many resources with respect to the constraints of the target platform (e.g. the cellular phone version of an information system dedicated to theatre will still enable the general task of booking theatre tickets, but not the subtask of viewing the free seats on a picture of the hall).

4.1.4.b.2 Subtasks Insertion

Causes for subtasks insertion involve:

- Insertion of a subtask because the target platform does not permit executing several tasks at the same time (e.g. on a mobile phone, as it is impossible to edit several items simultaneously, a selection task that would allow the user to choose which item he wants to modify should be added before any edition task mapped to more than one item).
- Insertion of a subtask because the display area on the target platform does not permit executing the same set of tasks within one presentation only, so that the tasks have to be split among several presentations, what may imply the insertion of additional navigation tasks between the new presentation spaces.

4.1.4.c Temporal Ordering Transformations

Examples of GD rules modifying the temporal ordering between tasks are:

- Sequentialization of tasks when the style of interaction changes (e.g. from a direct manipulation UI to a form-based UI). For example, Figure 4-13a depicts a task model where the first level of decomposition is regulated by a concurrent operator (the user can specify the payment method independently from entering the reference of the product). In Figure 4-13b, an additional constraint is set: the product has to be specified before entering the payment method.
- Conversely, some tasks that where sequential can become concurrent when the style of interaction changes.

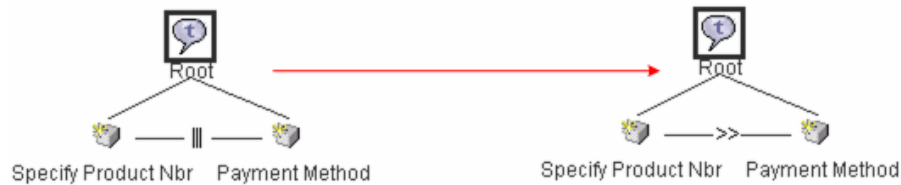


Figure 4-13 Example of temporal ordering transformation

4.1.4.d Concept Level Transformations

The domain model should be preserved as much as possible in order to permit interoperability across platforms (for example, several platform specific user interfaces should be able to share the same database). However, graceful degradation rules can modify the view given on some concepts:

- Information can be summarized or cut;
- Some attributes can be masked;
- Alternative shorter label or titles can be chosen;
- Numeric data (for example, tables) can be replaced by graphical representations (charts...);
- Text can be replaced by a graphical representation (icon instead of a menu item for example).

4.1.5 Discussion

The level of the CAMELEON framework where a rule can be applied (Tasks&Concepts, Abstract Interface, Concrete Interface or Final Interface) has not always been easy to identify, especially when considering pictures and other graphical content. In those cases, it is not always possible to decide whether an image is a presentational element belonging to the Concrete Interface, or whether it is content, belonging to the domain model: for example, resizing an image is similar to resizing a CIO, when cropping this image may be closer to text summarization. Our intuition is that the form and presentation of non textual elements are difficult to separate, and that the CAMELEON framework may be less suitable for describing such elements.

4.2 Formalization

4.2.1 Introduction

The main interest of formalizing GD rules is to avoid the ambiguity that could be linked to a description in natural language. In other words, we will make a descriptive use of the formal notations and do not intend to use them to support the verification or validation of the target user interfaces generated against formal properties.

A GD rule is as a model transformation. Each model has been described by a meta-model (UML class diagram). Each rule can thus be described as a transformation on the instances of this meta-model, using pre- and post-conditions on the meta-model instances. For most of the rules (but not all), pre- and post-conditions can be expressed easily with any language having the power of first order predicate calculus, plus some elements of set theory.

Several formal and semi-formal specification formalisms have been considered for specifying GD rules:

- Representation using a set of functions specified using pre- and post-conditions, where pre- and post-conditions are expressed using predicates.
- Specification using Z schemas
- OCL

The Object Constraint Language (OCL) [OMG05] [Warm99], used to specify constraints on UML models, has been selected because it is especially convenient for formalizing such rules:

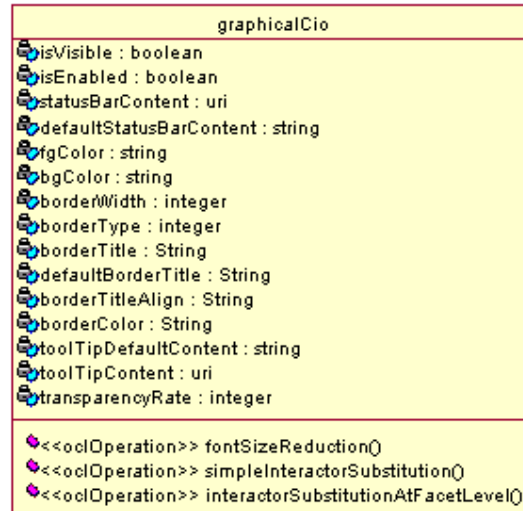
- The use of OCL guarantees coherence between the specification of the rules and the class diagrams described in Chapter 3 (class names, attributes, methods, multiplicity of associations...)
- Each class or association defined within a UML model can be referred to in OCL, which avoid defining artificial predicates only for the purpose of formalizing GD rules.
- Finally, OCL is an accepted standard in the UML community and is readable without need of a strong mathematical background.

An overview of OCL's main concepts can be found in Annex D.

4.2.2 Specification of GD rules in OCL: examples

Specifying GD rules with OCL is quite straightforward. Each rule can be specified directly in UsiXML's meta-model. Of course, some rules that can not be executed automatically without intervention of the designer (for example, deleting a task), can not either be given a formal specification. For each GD rule to specify, a corresponding UML operation is inserted in the class diagram, using the `oclOperation` stereotype. This operation is added into a class corresponding to the rule's source element. If a rule has several source elements, or makes use of several constructs in UsiXML's meta-description, the additional method is inserted into the most convenient host class. Let us examine some examples of how GD rules can be described in OCL.

Example 1: formalization of the FontSizeReduction rule



The font size reduction rule modifies objects of the class `graphicalCio` (see above). The OCL description of this rule consists merely in putting conditions on the `textSize` attribute:

rule name	FontSizeReduction
context	graphicalCio::fontSizeReduction()
pre	textSize > 6
post	textSize < textSize@pre textSize >= 6

Example 2: formalization of the SimpleInteractorSubstitution rule

After the very simple example above, let us examine a more complex rule involving several associated classes, belonging to different “models” of the UsiXML description. The interactor substitution rule is applied on `graphicalCIO`'s belonging to the CUI model and uses:

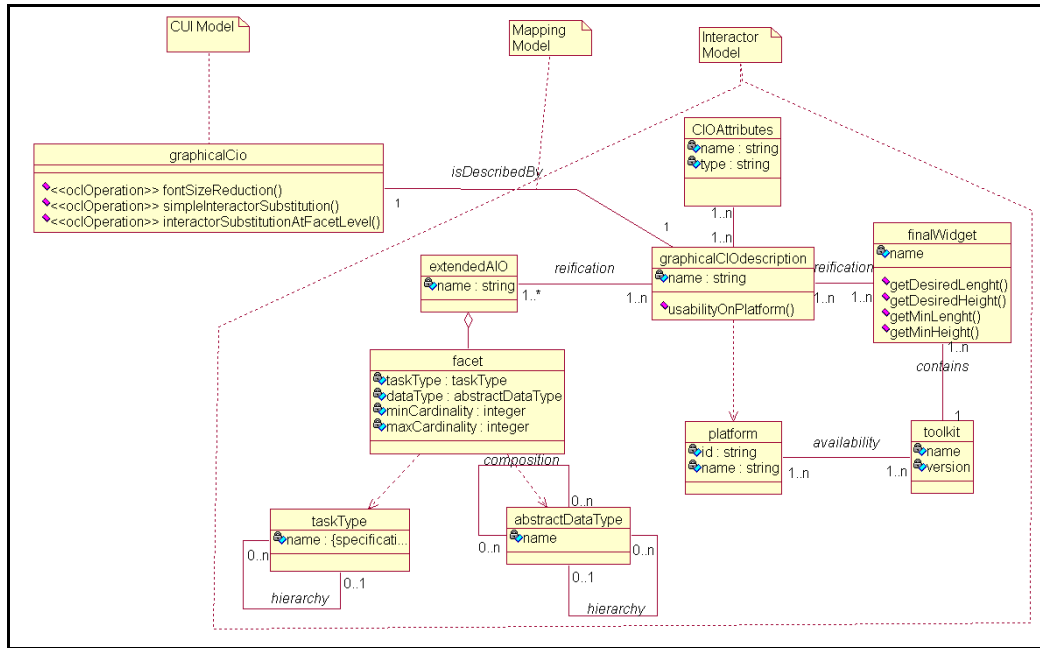
- Several classes from the Interactor model.
- Mappings between `graphicalCIO`'s and their description in the Interactor model.

The simple interactor substitution consists in replacing a `graphicalCIO` with another `graphicalCIO` reifying the same `extendedAIO`. For example, a `ListBox` could be replaced by a `Combobox`, as both reify the `MultipleTextChooser` `extendedAIO`. This is translated into an OCL post-condition that checks whether the two `graphicalCIO`'s (before and after the substitution) are different and are reifications of the same `extendedAIO`.

If the interactor on the source platform is an “ambiguous” graphical CIO, which may correspond to several `extendedAIO`'s, the simple interactor substitution rule can not be applied directly (hence the rule's precondition). For example, a `Combobox` can be

4. Effective knowledge for Graceful Degradation

abstracted into a SimpleTextChooser, but also into a MultipleTextChooser. Do we have to select a target interactor reifying SimpleTextChooser or MultipleTextChooser? This question can not be answered without examining the structure of the extendedAIO (its facets). On the contrary, the rule applies to graphicalCIO's whose abstraction(s) do not possess facets (e.g. Separators).



rule name	SimpleInteractorSubstitution
context	graphicalCIO::simpleInteractorSubstitution()
pre	graphicalCIO.graphicalCIOdescription.extendedAIO -> size = 1
post	graphicalCIO.graphicalCIOdescription.extendedAIO -> includesAll(graphicalCIO.graphicalCIOdescription.extendedAIO@pre)
	and
	graphicalCIO <> graphicalCIO@pre

Example 3: formalization of the InteractorSubstitutionAtFacetLevel

When simple interactor substitution can not be applied, or do not produce satisfying results, interaction substitution has to be performed at the facet level i.e. substitution by a CIO linked to another Extended AIO

- that supports a supertype of the original task type (first term of the post-condition),
- and/or a supertype of the original abstract data type (second term of the post-condition).

4. Effective knowledge for Graceful Degradation

Subtyping is evaluated using the `oclIsKindOf` OCL operation. The rule only applies to interactors possessing at least one central facet (precondition).

rule name	InteractorSubstitutionAtFacetLevel
context	graphicalCIO::InteractorSubstitutionAtFacetLevel()
pre	graphicalCIO. graphicalCIOdescription.extendedAIO.facet -> exists (isCentral=true)
post	(graphicalCIO@pre. graphicalCIOdescription.extendedAIO.facet.taskType -> select (tt_source: taskType isCentral=true)) -> forAll (graphicalCIO. graphicalCIOdescription.extendedAIO.facet.taskType -> exists((tt_target: taskType tt_source.oclIsKindOf(tt_target)) and (graphicalCIO@pre. graphicalCIOdescription.extendedAIO.facet. abstractDataType -> select (adt_source: abstractDataType isCentral=true) -> forAll (graphicalCIO. graphicalCIOdescription.extendedAIO.facet.taskType -> exists(adt_target: abstractDataType adt_source.oclIsKindOf(adt_target)) and graphicalCIO <> graphicalCIO@pre

The largest part of the rules catalogue remains unformalized yet, but these few examples permit demonstrating how formalization could be achieved. Nevertheless, the exercise is tedious, and the majority of the rules could have been expressed more quickly and effectively by using pseudo-code.

4.3 Discussion and conclusion

The above discussion of GD rules remained at the rather comfortable level of individual rules. However, rules are not independent: they are used in combination with other rules, and the application of one rule may have an effect on other rules.

Several types of relationships between rules could be listed:

- *Priority/ordering*: Which rule should be considered firstly when designing a new UI? Which sequence of rules should be applied afterwards? These questions will not be answered in this thesis, where we have considered that the responsibility for selecting and ordering rules was left to the human designer. A possible solution for improving this situation should be to analyse the practices of designers experimenting with the tools described in Chapter 6 and to describe the observed sequences of applied GD rules in terms of “frequently used transformation scenario’s”.
- *Entailment*: The application of some rules may trigger the need to apply other rules. For example, deleting an element of the UI is likely to be followed by the application

of a moving rule in order to avoid unnecessary blank spaces or to preserve the alignment of graphical components.

- *Exclusion:* Conversely, the application of some rules can make other rules inapplicable or useless. For example, if several CIO's of the source platform are merged into a single CIO on the target platform, it is not possible anymore to split between the original CIO's. Or, more generally, if a given transformation has generated an acceptable target UI, the application of additional GD rules may be less useful.

Another important issue for GD rules is their impact on the usability of the UI. GD rules are not supposed to “degrade” the ergonomics of the source user interface; instead, they are expected to solve some usability problems that would have occurred on the target UI if no adaptation had been performed. For example, repositioning content into one single column may avoid horizontal scrolling, while applying a splitting rule may be a solution to an excessive vertical scrolling depth. Applying a rule has also side effects: for example, splitting may cause a loss of contextual information, resizing fonts has an impact on the text's legibility,...

Foreseeing and describing the relationships between rules and their impact on ergonomics is difficult, if not impossible, due to the number of factors influencing the usability of a user interface and to the potential combinatorial explosion when a large number of rules are considered. Such information is not an absolute requirement for a human-controlled application of the rules, at design time, as envisioned in this thesis: the knowledge base of GD rules described in section 6.1 contains some directions on the links between rules and on the advantages and disadvantages of these rules, but the tool supporting the semi-automatic application of the rules (section 6.2) does not. Rather than trying to describe completely and precisely the effects of a rule, we could imagine enhancing the last tool with an automatic usability evaluation component, that would be called iteratively to analyse the target UI's ergonomics and make recommendations on the next rule or set of rules to apply.

Most of the rules presented above were taken from the HCI literature, in order to gather a maximum of transformations that could be automated, even if non-automatable rules were also collected. There is no pretension to completeness in this description, but a mere attempt to clarify the notion of GD rule by exploring which components of a user interface can be subject to modification when a design is adapted to a more constrained platform. Likewise, the classification of the rules using the CAMELEON framework was merely theoretical: we found that, in most cases, this framework helped us structuring our descriptions and understanding the transformation processes. The decision of classifying a rule into one abstraction level or another might have an impact on the implementation, if we chose to perform graceful degradation as a composition of translation (“horizontal” transformation at the level where the rule has been classified) and reification. Actually, as will be seen in section 6.2, we opted in favour of an implementation at the CUI level for all rules, while taking advantage of the information of models at higher abstraction levels, when these models are available. This solution

permits circumventing the additional difficulties of reifying specifications starting from the AUI or Task& Concepts level, instead of starting from the CUI level, and of maintaining the consistency between the various layers of the target UI specification (the only model produced for the target platform is a CUI).

We also observed that most of the rules can not be easily specified declaratively. As no assumption was made on the better way to achieve model transformations, having only a procedural description of most of the transformation rules is not a problem. However, the difficulty of specifying rules declaratively and even, in some cases, of giving a procedural specification, may indicate that the approach is not suitable for complex problems such as complex layout adaptation.

Chapter 5 Multilevel application of rules: example of the splitting rule

This chapter is dedicated to a detailed presentation of the splitting rule, which permits paginating content [Flor06]. Pagination is perhaps the most difficult and significant step of the whole graceful degradation process. Splitting generates important changes into the very structure of the UI, has an important influence on the quality of the final results, and is appreciated by users that consider it as one of the most useful GD rules [Henr04].

Automatic pagination is a complex problem that has partially been addressed in earlier work that will be described in section 5.1. Splitting will be examined at two levels of abstraction: Concrete UI (section 5.2) and Abstract UI (section 5.3).

5.1 State of the art of pagination techniques

Pagination has been applied in different contexts, notably for redistribution of Web content, especially Web forms, among several pages and as a part of UIDL-based systems.

5.1.1 Pagination of Web pages

5.1.1.a Covigo

Covigo's library of special tags for HTML [Mand02] implements pagination of Web pages at run-time, using simple heuristics such as breaking every fifth `<tr>` or breaking by size. The size can be retrieved from the CC/PP profile of the connecting device. The content outside the paginated body is repeated for each page (i.e. headers, footers, etc).

5.1.1.b RIML

Similarly to the previous approach, the mark-up language RIML [Spri03], which relies on XHTML and XFORMS for content definition, has defined separate, additional mark-up for specifying layout and pagination capabilities. The new mark-up delimits sections, which are the interface's building blocks, and associated containers. Each container can be specified as a paginating container. After pagination, the sections that belong to a paginating container can be distributed over different pages, while the content of non-paginating containers will be repeated on each resulting page. A special section contains the navigational elements to include in every paginated page when pagination occurs.

5.1.1.c Watters and Zhang

Unlike the two first approaches, Watters and Zhang [Watt03] can process any pre-existing HTML form, and not only newly created page specifications. Their algorithm segments forms into a

sequence of smaller forms, using partition indicators such as horizontal lines, nested lists and tables. Of course, grouping directives induced from the “partition indicators” within the code are less accurate than they would have been in an explicit specification. Complex layout relationships (e.g. use of tables for layout purpose) will probably constitute a bottleneck for such approaches.

5.1.1.d Chen et al.

Splitting pre-existing Web pages is also the concern of Chen & al. [Chen05]. Their technique consists of three steps. First, the high-level content blocks typical in current Web site designs (header, footer, sidebars, and body) are identified by analysing the position and dimension of the nodes in the HTML DOM tree. Afterwards, each block can be further partitioned by detecting “explicit separators” i.e., tags such as <HR>, <TABLE> or <DIV> (similarly to 5.1.1.c). The last step consists of finding “implicit separators” i.e., blank spaces. Once the page is split into fragments, an index page linking to each subpage is produced by generating a thumbnail image of the original Web page, with the appropriate hyperlinks. The technique can be deployed on the server side, proxy side or client side.

5.1.2 Pagination of content expressed in a user interface description language

The second group of approaches relies on a generic description of the user interface in a higher level language, instead of HTML.

5.1.2.a Pagination with DDL

Göbel et al. [Göbe01] use an XML-based Dialog Description Language (DDL), specially conceived for the device-independent description of web-based dialogs. A dialog in DLL is composed of containers and other elements (controls, images...). Containers whose elements must appear together are declared atomic. Elements are assigned a weight indicating resource requirements in terms of memory and screen size. The fragmentation generates fragments of balanced weights, while respecting the integrity of atomic containers. Navigation elements are added in order to permit navigation between dialog fragments. No indication is given on how to assign weights to leaf elements, which is a difficult task, especially when considering multiplatform rendering.

5.1.2.b Pagination with XUL

Ye and Herbert [Ye04] apply similar heuristics on an abstract UI description in XUL. Their algorithm exploits the hierarchy of widgets and containers, while respecting the value of a “breakable” attribute attached to each component, which has to be explicitly provided by the designer.

5.1.2.c PIMA

The PIMA system [Bana04] also relies on a unique high level description, which is then converted into multiple device-specific representations. This conversion includes a splitting process. As in other approaches, PIMA's algorithm uses grouping constraints defined at the generic description level as well as size constraints. An interesting feature of PIMA is to take into account the navigation and the possibility to apply distinct navigation policies between screens resulting from a splitting process: creation of a fully connected navigation (e.g., tabbed windows), of a linear navigation (e.g., forward and back buttons), etc.

5.1.2.d ROAM

While the fragmentation methods enumerated so far were mostly working on a hierarchy of interface components (i.e. on elements related to the presentation of the user interface), the splitting algorithm of the Roam system [Chu04] takes as input a structure combining a task model and a layout structure. Roam's splitting algorithm is only a small part of a general transformation algorithm which adapts a presentation specified for the device with the largest display device for use on smaller devices, while taking into account layout specifications. No scrolling is allowed in the generated pages. Again, the splitting algorithm works on a tree structure whose nodes can be annotated as splittable or not. It does not intend to find the best place to split but merely places the extra widgets that can not fit in a page on a new page. Navigation between the new pages is also generated. The navigation policy is determined by the target device: on a Pocket PC, a "page selection" menu item is added to the menu bar and on a cell phone, a page containing the selection menu is added, and all other pages provide a link to this menu page.

In comparison, our own splitting rules will be:

- *Generic*. GD rules rely on UsiXML and are not tied to a given technology at the Final UI level, so that we will not have to write a separate set of algorithms for HTML pages (like [Mand02], [Spri03] or [Watt03]) and another one for AWT/Swing windows, for example. The use of UsiXML also avoids the need to introduce another mark-up language specially designed for supporting pagination, or additional language constructs, unlike RIML.
- *Fully controlled by the developer*. With the exception of RIML, which allows some customization of the size limit applied to generated pages, all the approaches described above are fully automatic; no human control is foreseen.
- *Task driven*. Information on "breakable" or "splittable" fragments ([Göbe01], [Spri03], [Ye04], [Bana04], [Chu04]) is useful, but not very rich semantically. When higher level specifications, especially tasks models, are available (like in ROAM), these specifications must be used to refine the splitting process. In particular, the temporal relationships between tasks must be used.
- *Able to adapt the dialog* (i.e.; the transitions between containers) in a flexible, customizable way. When they do not merely evade the problem of the dialog between the newly created fragments ([Ye04]), most of the splitting approaches only consider one possible kind of transition (the basic next-previous navigation, or the indexed navigation). A notable exception is PIMA which, nevertheless, does not seem to exploit the whole range of navigation possibilities.

5.2 Splitting at the Concrete UI level

Not all the layers of UsiXML are mandatory in a UI specification. In the simplest case, we suppose that the designer has just produced a description of the Concrete User Interface (CUI).

Different constructs in the CUI model of UsiXML can be used for pagination purposes:

- The layout of each graphical container (window, dialog box...) is specified using embedded boxes. Those boxes are declared as splittable or not, which is the basic ingredient for pagination.

5. Multilevel application of rules: example of the splitting rule

- Each container and each component of the CUI is marked as pageable or not. Pageable components can be distributed between the graphical containers created during the splitting process, while non pageable components have to be present in each fragment. For example, a menu bar or a widget permitting to logout from the system could constitute non pageable components, because their presence in each container is useful.
- Transitions can be specified between each pair of containers.

Implementing splitting rules starting from such a model is quite a straightforward process: the splittable attribute indicates where to split, and the pageable attribute indicates which elements will be duplicated. When several boxes are splittable, the outermost box is chosen.

Each execution of our splitting rules is fully controllable and configurable by the designer. The parameters taken into account by the algorithm are:

- The number of graphical containers (windows...) at output. By default, we take the number of boxes directly embedded into the main container (level 1–boxes).
- The content of the n graphical containers at output. By default, we take the content of each level 1–box, but the designer is allowed to select content by drag-and-drop.
- The names assigned to each graphical containers at output, which will be used as windows titles and for widgets pointing to these interactive spaces. By default, names are automatically generated by suffixing the original name.

One last parameter, namely the type of transitions generated between the new graphical containers, deserves a little more explanation.

As the dialog model of UsiXML does not possess a graphical representation yet, we will represent the behavioural aspects as statecharts. We have considered four types of transitions between the graphical containers generated by the splitting algorithm (hereafter target containers):

- *Linear navigation* (Figure 5-1) establishes transitions between one target container and another container considered as its successor. It is typically realized with “next-previous” links or buttons. This type of navigation offers the most guidance to the user. Linear navigation is unidirectional (for example, “next” links only) or bidirectional (going backward is allowed).
- *Indexed navigation* (Figure 5-2) establishes transitions between a newly created container, the index, and each target container. Unidirectional indexed navigation provides only transitions from the index to the other containers; while bidirectional indexed navigation offers transitions in both directions.
- *Mixed navigation* (Figure 5-3) is a combination of linear and indexed navigation.
- *Fully-connected navigation* (Figure 5-4) links each pair of target containers. This type of navigation is the least restricting for the user. It is typically rendered as a tabbed panel.



Figure 5-1 Unidirectional/bidirectional linear navigation

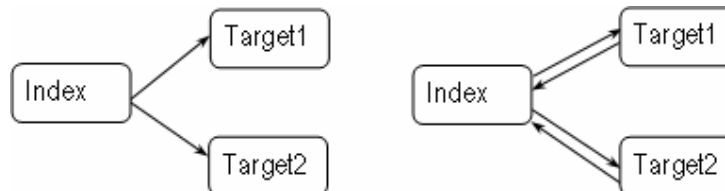


Figure 5-2 Unidirectional/bidirectional indexed navigation

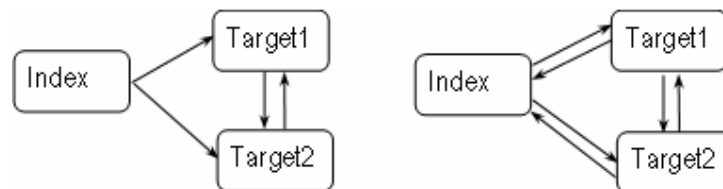


Figure 5-3 Unidirectional/bidirectional mixed navigation

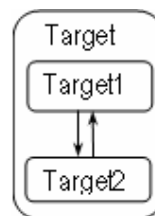


Figure 5-4 Fully-connected navigation

The splitting algorithm has been integrated in a GD plug-in for the GrafiXML environment (section 6.2). In the GD plug-in, the application of the splitting rule is under the control of the designer. He/she can parameterize the rule, apply it, preview the results, and compare alternatives. He/she is given guidance thanks to the knowledge base of rules linked with the tool (section 6.1).

5.3 Splitting at the Abstract UI level

Until now, we have supposed that the designer has just produced a description of the Concrete User interface. Let's now consider the scenario where a task model and an Abstract user interface have been produced. In that case, we can use the high level information from the task model to refine our algorithm.

5.3.1 Preliminaries

The CT^T-based task model of UsiXML described in 3.2 is thus a hierarchy of tasks, where each task can be decomposed into two or more subtasks. A task T can be declared as optional ($[T]$) or iterative (T^*). Sibling tasks, appearing at the same level in the task hierarchy, are connected by temporal/logical operators:

- Choice $T1 \parallel T2$: exclusive choice between $T1$ and $T2$.
- Order independency $T1 \mid = \mid T2$: $T1$ and $T2$ can be performed in any order.
- Independent concurrency $T1 \mid \mid \mid T2$ and Concurrency with information exchange $T1 \mid \parallel \mid T2$: $T1$ and $T2$ can be performed in any order. We shall call these operators “concurrent operators”.
- Disabling $T1 \mid > T2$ and Suspend-resume $T1 \mid > T2$: $T2$ disables/interrupts $T1$.
- Enabling ($T1 \mid >> T2$) and Enabling with information passing ($T1 \mid \parallel >> T2$): $T1$ and $T2$ are executed in sequence. We shall call those operators “sequential operators”.

There is a priority ordering between the temporal operators. Figure 5-5 shows the list of operators sorted by decreasing order of priority. The order is as follows:

- (1) Unary operators: iteration (T^*), finite iteration ($T(n)$) and optional ($[T]$).
- (2) Deterministic ($T1 \parallel T2$) and non-deterministic choice ($T1 \pi T2$).
- (3) Order independency ($T1 \mid = \mid T2$).
- (4) Independent concurrency ($T1 \mid \mid \mid T2$) and concurrency with information exchange ($T1 \mid \parallel \mid T2$).
- (5) Disabling ($T1 \mid > T2$) and suspend-resume ($T1 \mid > T2$).
- (6) Enabling ($T1 \mid >> T2$) and enabling with information passing ($T1 \mid \parallel >> T2$).

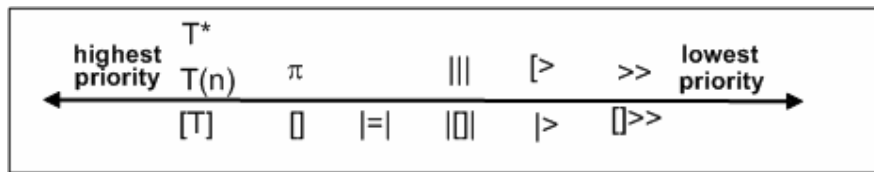


Figure 5-5 Priority ordering between the temporal operators in the task model

An easy way to cope with the priorities among these temporal operators is the priority tree technique used in [Luyt03b]. A priority tree is a view on a task model (originally a CT^T task model), with the same semantics as the original task model, but where all the temporal relations at Figure 5-6 shows, at the left hand side, a CT^T/UsiXML task model in an arbitrary form and at the right hand side, the priority tree generated from the first representation.

5. Multilevel application of rules: example of the splitting rule

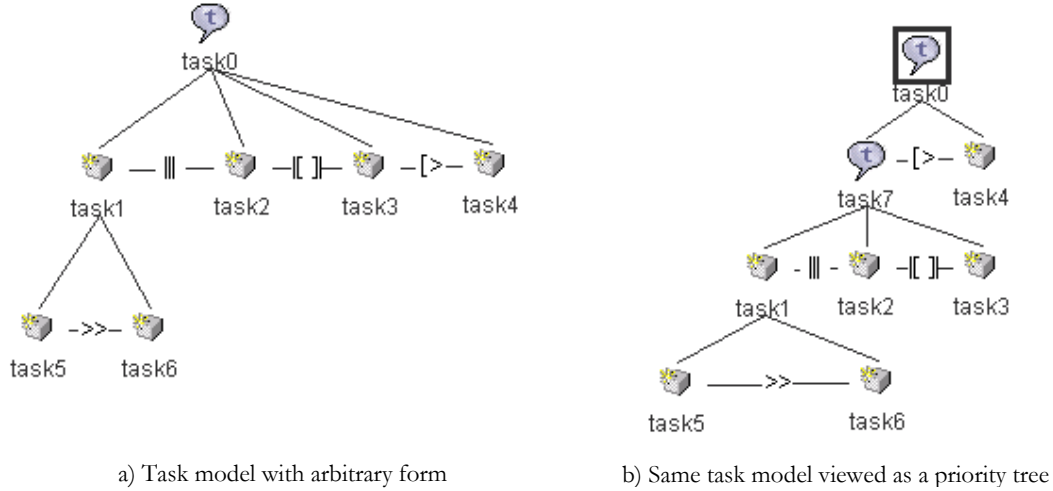


Figure 5-6 A task model and its priority tree representation

As explained above (3.4), an Abstract User Interface describes how the tasks contained in the task model will be grouped together within a presentation (window, panel of a tabbed panel, dialog box). Task groupings, although expressed at a modality and interactor independent level, are highly dependent on the available display area, and thus on the computing platform: intuitively, we understand that a set of tasks presented together in the same window on a desktop computer will not always be kept together on the PDA version of the user interface.

In UsiXML, an AUI consists of Abstract Containers, or interaction spaces, whose components are mapped to tasks of the task model. In this chapter, we will rely on the shortcut representation of interaction spaces which consists of drawing contours directly on the task model, instead of presenting a task model, an AUI model and a mapping model separately.

We consider that all tasks mapped to components of an interaction space are leaf tasks, because only leaf tasks will be directly reified in the concrete presentation.

Which groupings of tasks form a correct interaction space? There is probably no good answer to that question since the constitution of the interaction space is often a human activity that depends on the designer's experience and practices. Research has shown that it is possible to generate correct presentation units from a task model and a domain model. For example, the CTTE case tool generates "Enable Tasks Sets" (ETS) i.e. sets of tasks that are enabled in the same slice of time. A recent version of this algorithm can be found in [Luyt03b]. [Pate00] also describes heuristics to group ETS. However, the ETS algorithm and the grouping heuristics are definitely unable to produce all the correct interaction spaces. As our objective is translation, we will assume that every interaction space on the source platform is well-formed because it was produced by a human designer.

Translating at the abstract interface level may imply modifications in the distribution of tasks among interaction spaces. Not all transformations are correct and the correctness of a transformation is linked with the temporal operators between the tasks within the task model. We have built our transformation algorithm inductively, starting from small examples of tasks models

5. Multilevel application of rules: example of the splitting rule

together with the interaction space that could be acceptable for these tasks models. As we do not define what a well-formed interaction space is, we are also unable to prove theoretically the correctness of our translation rules, but we have tested our algorithm on a battery of examples. A better evaluation could have been provided by comparing the results of the algorithms with the designs produced by human designers asked to manually split a source interface.

At this level, we will adopt two simplifying hypothesis:

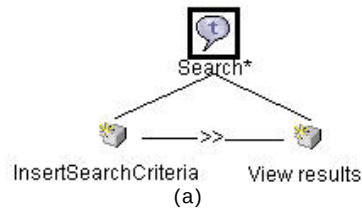
- We will work with constant tasks sets: we consider that all the tasks will be present both on the source platform and on the target platform
- We do not consider duplicated tasks. Duplicated tasks are tasks with the same identifier appearing at different places in the task model. They can be used to express recursion, if one task is inserted in a subtree originated by a task with the same identifier. Although allowed by the CTT environment, those situations are not very frequent.

5.3.2 Principles

At the AUI level, our splitting algorithm relies on a few principles, detailed hereafter.

► Principle 1: An interaction space can be split at the level of a sequential operator.

“Splitting at the level of operator Op” means that all the tasks (and their descendants) to the left of Op which belonged to the source interaction space will be assigned to target interaction space 1 (IS-target1) and that all the tasks (and their descendants) to the right of Op which belonged to the source interaction space will be assigned to target interaction space 2 (IS-target2). Unlike generative approaches such as Paterno’s ETS algorithm [Pate00], we do not consider that sequential tasks are automatically assigned to different interaction spaces. This decision should remain the responsibility of the designer to preserve human control and flexibility. Presenting sequential tasks in the same interaction space can make sense, especially when those sequential tasks decompose a higher level task which has to be accomplished iteratively.



5. Multilevel application of rules: example of the splitting rule



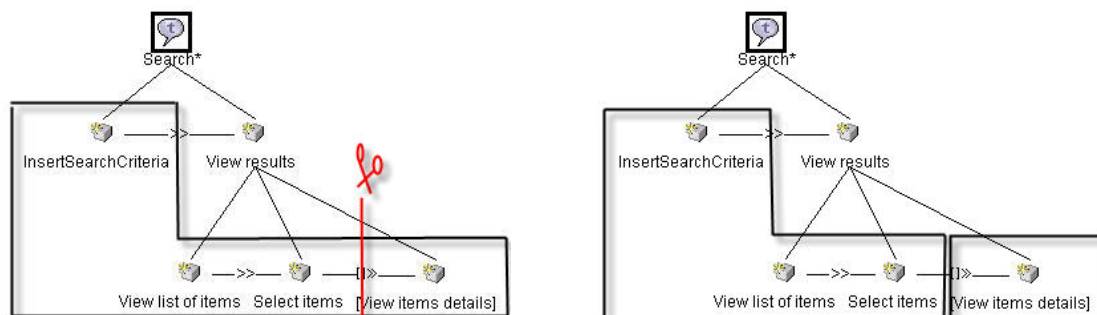
Figure 5-7 Task model for a simple IR system, with different distribution of tasks among interaction spaces

Let's consider again the example introduced above and reproduced on Figure 5-7a.

In this simple information retrieval system, a designer could choose to place the two sequential tasks **Insert search criteria** and **View results** into the same interaction space, if the screen space is unconstrained (Figure 5-7). If a new version of the user interface has to be conceived for a platform with less display capabilities, pagination could be operated at the level of the sequential operator, creating the two interaction spaces in Figure 5-7c.

► Principle 2: When an interaction space includes several sequential tasks, split before the first optional task in the sequence.

Where the optional task is not actually carried on, the user will not even have to navigate to the second interaction space. For example, let us consider the extension of the previous task model reproduced in Fig. 9. The **View results** task now consists of three subtasks: the first subtask displays the full list of results generated by the request to the information system; the second subtask is a selection task that focuses on one of the displayed results; and the last subtask is an optional task which displays a complete description of the selected item. Again, in a first, unconstrained version of the specification, the designer could choose presenting all tasks in the same interaction space (Figure 5-8a). If we now determine the best place to operate pagination, we will notice that here, obviously, the best design solution is to split the source interaction space before the optional task **View items details**, generating the target interaction spaces of Figure 5-8b. By the way, splitting between two tasks linked with the same component of the AUI is not allowed: it would not make sense to split between **View list of items** and **Select items**.



5. Multilevel application of rules: example of the splitting rule

(a)

(b)

Figure 5-8 Splitting an interaction space containing a sequence of tasks, one of them being an optional task.

► Principle 3: When it is not possible to split an interaction space at the level of a sequential operator, split at the level of a concurrent, order independency or choice operator (|||, |□|, |=|, □)

The temporal operators with a lower priority are considered first. Splitting at the level of an interrupting or disabling task is not allowed: splitting at the level of one of the four operators above introduces constraints that were not present in the task model, and splitting at the level of a sequential operator should always be preferred when possible. Figure 5-9 shows the example of a (very) small task model which consists of the higher level task **Insert personal data** and its two concurrent subtasks **Insert identity** and **Insert address**. The initial interaction space contains both subtasks (Figure 5-9a). Splitting separates those two subtasks (Figure 5-9b).



Figure 5-9 Splitting an interaction space at the level of a concurrent operator

► Principle 4: When splitting rules can be applied at distinct levels in the task hierarchy, split at the highest level.

The rationale behind this principle is that tasks at a lower level in the task tree will be more closely semantically linked than tasks at a higher level. For example, let us consider the task model in Figure 5-10, a more complete version of the previous example. On the first, less constrained platform, the designer could place all the tasks together (Figure 5-10a). If less space is available, the best place to operate pagination, obviously, is to split the source interaction space at the highest level in the hierarchy, generating the target interaction spaces shown on Figure 5-10b. This transformation preserves the integrity of the **Insert identity** and **Insert address** tasks: their subtasks, which were considered by the designer as tied enough to form concepts, are maintained together.

5. Multilevel application of rules: example of the splitting rule

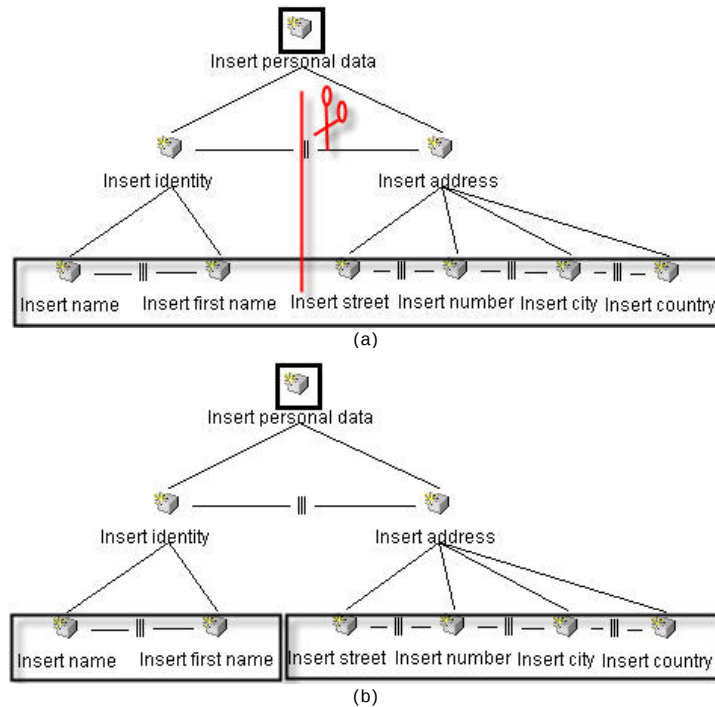


Figure 5-10 Splitting an interaction space containing concurrent tasks at different level in the hierarchy

► Principle 5: When splitting in the scope of an operator with a higher level of priority, a distribution of tasks amongst target interaction spaces has to be operated.

Let us consider the small extension to the previous example as reproduced in Figure 5-11. If we naively split at the level of the sequential operator as described above, we will obtain a first interaction space $IS1 = (\text{Insert name}, \text{Insert first name})$ and a second interaction space $IS2 = (\text{Insert street}, \text{Insert number}, \text{Insert city}, \text{Insert country}, \text{Cancel})$. Such a transformation introduces a discrepancy between source and target platforms since it is not possible anymore to access the disabling **Cancel** task when performing the **Insert identity** task on the target platform: the user has to realize entirely the **Insert identity** task, and then he/she should access the second interaction space where the **Cancel** task is available. This kind of design introduces a usability defect and can be frustrating for the user, especially when long tasks have to be achieved entirely without any possibility of interruption. A better transformation should distribute the task to the right of the disable operator to each target interaction space (Figure 5-11b), giving a solution with $IS1 = (\text{Insert name}, \text{Insert first name}, \text{Cancel})$ and $IS2 = (\text{Insert street}, \text{Insert number}, \text{Insert city}, \text{Insert country}, \text{Cancel})$. This distribution principle is now defined in the next subsection (5.3.3), along with the complete description of the algorithm based on this principle.

5. Multilevel application of rules: example of the splitting rule

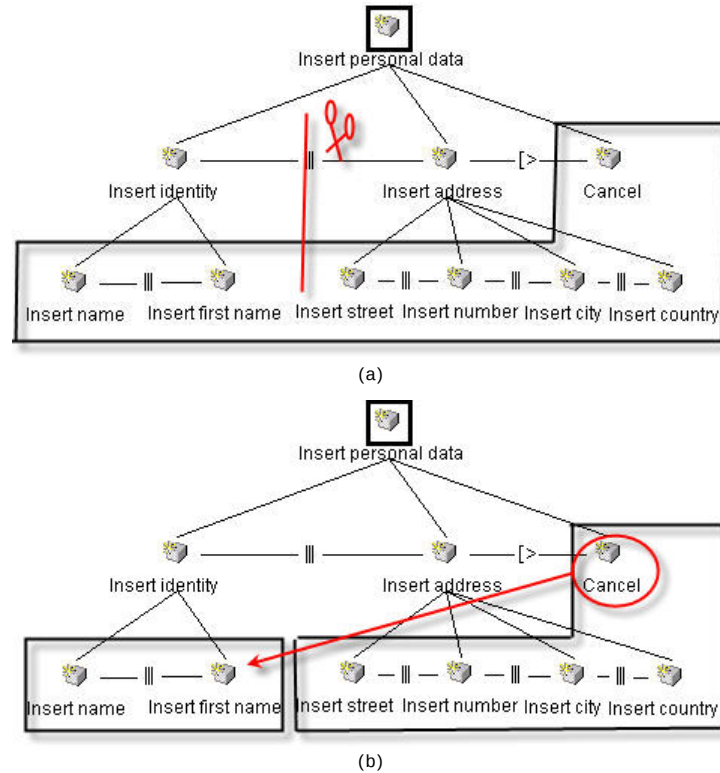


Figure 5-11 An example of distribution of a disabling task

5.3.3 Description of the algorithm

Our algorithm requires at input a subset of tasks of the task model, viewed as a priority tree. This subset of tasks, or Source interaction space (hereafter: ISSource) contains the leaf tasks that are mapped to the components of an abstract container that the designer has decided to split. ISSource is a list $(T_1, \dots, T_n)$ where

- $T_1, \dots, T_n$ are leaf tasks in the task model.
- $(T_1, \dots, T_n)$ is a subsequence of the list $(T_i, \dots, T_j)$ formed by the leaf nodes in the task model arranged as an ordered tree.

Until the interaction space is split and unless there are no more operators to go through:

1. We try to split at the level of a sequential operator

1.1. If there is an optional task in the sequence, we split before this task.

1.2. Else

1.2.1. We look for the first suitable sequential operator where to split. "First" means "at the highest level in the task hierarchy" and "suitable" means that splitting at that place would generate non empty target interaction spaces, well balanced in terms of number of tasks. We search the task tree applying a breadth-first strategy, starting from the task that is the lowest common ancestor of the tasks forming ISSource.

5. Multilevel application of rules: example of the splitting rule

1.2.2. If such a sequential operator is found, the source interaction space is then split into two temporary target interaction spaces IStarget1 and IStarget2. Let T1 and T2 be the two tasks in the task model linked by the operator where we have decided to split. IStarget1 will contain the first part of ISsource, delimited by T1 if T1 is a leaf task, its right-most descendant otherwise. IStarget2 will contain the remainder of ISsource.

1.3. We then apply distribution rules.

2. When it was not possible to operate sequential splitting, we then try to split at the level of another operator.

2.1. We look for the first suitable operator where to split.

2.2. If such an operator is found, the source interaction space is then split into two temporary target interaction spaces.

2.3. We then apply distribution rules.

Distribution rules are applied when splitting occurs between two tasks T1 and T2 that remain in the scope of a temporal operator with higher priority (following the priority ordering of Figure 5-5). By construction, splitting always occurs between sister tasks in the priority tree. Let T1 and T2 be two sister tasks, linked by temporal operator Op1. T1 and T2 are in the scope of temporal operator Op2 iff an ancestor of T1 and T2 is linked by a temporal operator Op2 to a given task T3. If Op2 has a higher priority level than Op1 and T3 has descendants in ISsource, then distribution must be applied.

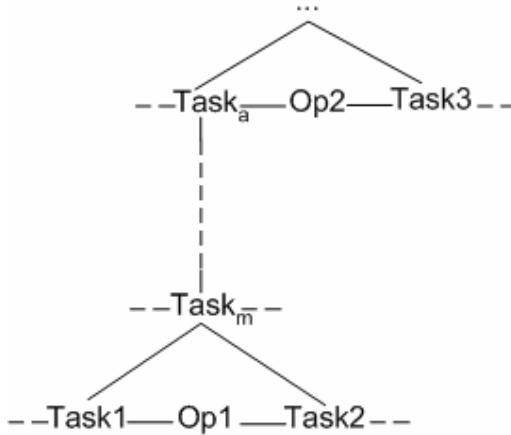


Figure 5-12 Scope of a temporal operator: illustration

Distribution consists in appending to the right of IStarget1 the descendants of T3 that belong to ISsource and appending to the left of IStarget2 the descendants of T3 that belong to ISsource.

The algorithm starts by distributing at the level of the mother of T1 and T2, and then the upper levels are successively considered, until reaching the level of the lowest common ancestor of all tasks in ISsource.

5.4 Conclusion

When applied at the CUI level, the algorithm proposed is quite classical. Nevertheless, it goes further than state-of-the art approaches listed in section 5.1.

Our approach is original in that it invokes the UI description at several levels of abstraction taking them into account when available. As far as we know, no similar attempt exists today that exploits information from the AUI and task levels to improve the splitting process.

In the future, improvements could be made to the algorithm. Information from the domain model could be used, in order to check which tasks manipulate the same concepts or have some concepts in common: these tasks should be preferably grouped together. Other criteria could be used, such as the balance in terms of cognitive load of each target interaction space (the cognitive load of each target interaction space should be similar). The cognitive load of a task could be approximated by the number of concepts manipulated by that task. The number of different objects, classes and relationships from the class diagram manipulated by a task could also have an influence on its cognitive load. Also the balance in terms of display area between the target interaction space or the ratio between display size and available screen size on the target platform seem obvious criteria to take into account. The exact display area required by a task can not be determined from the UI specification only, but it could be approximated by analysing the components of the CUI mapped to the task.

Chapter 6 Tool support

GD rules are meant to be applied either manually, by a human designer seeking guidance on how to adapt a UI to a more constrained platform, either automatically. In the first scenario, the designer needs to have an easy access to a structured set of transformation rules. For this reason, we have gathered a collection of rules in a knowledge base described in section 6.1. In the second scenario, the GD rules should be applied to a UsiXML specification. A prototype for supporting the automatic application of the rules is presented in section 6.2.

6.1 A knowledge base of GD rules

In Chapter 4, we presented a classification of GD rules. The rules introduced there were very general and some of them were not directly usable. For example, at the Concrete User Interface level, we identified a “moving rule”. However, this kind of rule is neither useful to a human designer, neither directly automatable. In order to identify more precise, working rules, a second exploratory study has been carried out, notably by two students during their master thesis, and the catalogue of rules has been extended until becoming a paper document of 200 pages [Henr04] where, for example, the “moving rule” has been broken up into 10 rules with a more precise definition, as shown on Table 6-1.

Vertical repositioning	
R1.	Vertical repositioning of elements in one or more columns
R2.	Vertical repositioning with a column length constraint
R3.	Justified vertical repositioning
R4.	Centred vertical repositioning
R5.	Vertical repositioning with indentation
R6.	Balanced vertical repositioning
Horizontal repositioning	
R7.	Horizontal repositioning with a row length constraint
R8.	Justified horizontal repositioning
R9.	Centred horizontal repositioning
R10.	Balanced horizontal repositioning

Table 6-1 Decomposition of the moving rule

This second stage of the gathering of rules raised two questions:

- What makes up a GD rule and what does not? We need to identify discriminating criteria that will permit us to decide whether a modification between observed UI

on multiplatform systems would be included as a rule in the knowledge base or not.

- The increasing number of rules has pointed out the interest to gather them in a knowledge base. Indeed, potential use of the set of rules in traditional design requires the rules to be given a good organization, in order to be easily retrieved and used by a human designer. What should be the best organization for this knowledge base? This question has not been researched until now: tools such as SIERRA [Vand95], Sherlock [Gram00], GUIDE [Henn00] or MetroWeb [Mari05] manage knowledge bases of usability guidelines, but no tool permits searching information and structuring knowledge about adaptation rules.

6.1.1 Selection criteria

In order to decide what will be considered as a GD rule and what will not, we will use several criteria:

- *Reusability*: only generic transformations reusable beyond a specific software or domain will be kept. We opted for that solution in order to provide a basic set of rules meant to apply in the broadest range of contexts, but of course this should not prevent designers from developing their own domain specific knowledge bases. On the other side, rules specific to a given pair of platforms or to a given target platform will be kept (e.g. for all transformations to an interactive kiosk, resize all widgets with a minimum of 9.12 mm or 38 x 38 pixels¹²).
- *Good level of granularity*: rules are to be precise enough. For example “move widgets” is an indication that is neither useful to a human designer, neither automatable, while “put all widgets behind their associated label” is a rule that can be applied by a tool and that is more useful to a designer.
- *Complexity*: some transformations are both useful and reusable, but are not inserted in the database because they introduce much more programming complexity at the target platform side than in the original user interface. Transformations based on complex visualisation techniques such as zooming, fish-eye or rifling are thus deliberately excluded.

6.1.2 Structure of the knowledge base

6.1.2.a Introduction

The structure of the knowledge base will depend on the answer to two questions:

- What kind of information about a GD rule do we want to represent?
- To what kind of requests do we want to answer?

¹² SDK Documentation for Windows Mobile-Based Pocket PCs, Pocket PC User Interface Guidelines, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/ui_guide_ppc/htm/Ergonomics.asp

In answer to these questions, a list of requirements was established:

- The knowledge base should include a description of each rule in terms of input and output, a formal description (4.2) when available, an indication on its level in the CAMELEON framework, examples and references, comments on the advantages and disadvantages of using it, situations when the rule should not be applied, information on its importance, on whether it can be automated or not, on its relationship with other rules.
- The rules database must satisfy requests such as “Which rules are available when migrating from a classical Web-based user interface to a Web-based user interface on a PDA? Or, more precisely, to a Pocket PC?”, “Which rules can I use when migrating to a platform with a smaller screen size? Or with a lower-resolution pointing device? Or with only a limited version of the graphical toolkit?”

We found that the best way to achieve the last requirement was to consider that a GD rule is composed of a condition (left-hand side of the rule) and a reaction (right-hand side of the rule). This structure has the additional advantage to be compatible with the meta-model of adaptation rules for plastic environments recently developed in the framework of the SIMILAR network [Gann05].

6.1.2.b Class diagram of the database

Starting from the basic structure selected above, we have established a class diagram of the concepts to be involved in our database (Figure 6-1).

Let us now review the content of the class diagram.

GDRule

A graceful degradation rule (GDRule) is a transformation rule which will perform adaptation of the presentation (and indirectly of the dialog) to the platform, when the target platform is more constrained than the source platform. A GDRule is an aggregation of a condition (left-hand side of the rule) and a reaction (right-hand side of the rule). It is characterized by:

- An identifier (*idRule*).
- A *centrality*: a rule is central if its application is mandatory, non central otherwise. An example of central rule is the interactor substitution rule, when the interactor is no longer available on the target platform.
- An evaluation in terms of the *advantages* / *disadvantages* the application of the rule will have on the target user interface and on the multiplatform system in general.
- Possible *exceptions* to the application of the rule. For example, resizing images containing text or figures is likely to make them illegible and should be avoided.

6. Tool support

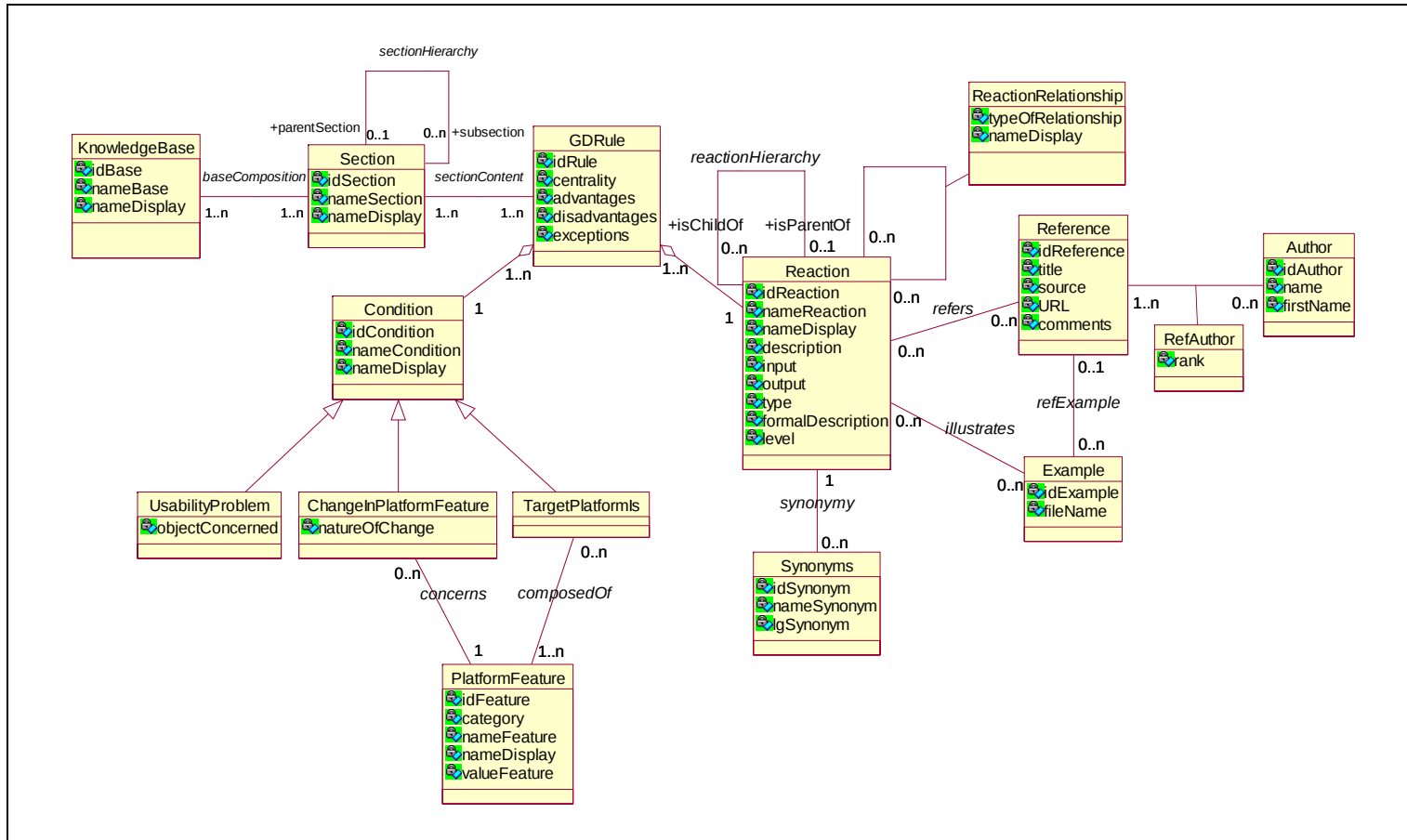


Figure 6-1 Class diagram of the concepts of the knowledge base

Condition

A condition constitutes the left-hand side of a graceful degradation rule. A condition is described by the following attributes:

- An identifier (*idCondition*).
- A internal name (*nameCondition*) and a name in a format that will be displayed to the user (*nameDisplay*).

As we distinguish between three kinds of launching conditions, this class has three subclasses: UsabilityProblem, ChangeInPlatformFeature and TargetPlatformIs.

UsabilityProblem (Subclass of Condition)

Conditions of this class are expressed as a usability problem: for example “horizontal scrolling on the target UI” or “font unreadable”. They contain the additional attribute *objectConcerned*, which specifies the name of the object concerned by the usability problem (e.g. fonts, images ... or all objects).

ChangeInPlatformFeature (Subclass of Condition)

Conditions of this class are linked to a change in a platform feature: for example “pointing device has changed to stylus” or “number of screen colours has decreased”. They contain the additional attribute *natureOfChange*, which describes the type of modification undergone by the linked platform feature (e.g. decreases, disappears...). An instance of ChangeInPlatformFeature is linked to 1 and only one instance of PlatformFeature.

TargetPlatformIs (Subclass of Condition)

This class includes conditions related to one or several characteristics of the target platform: for example “on a Compaq iPaq Pocket PC” or “on a platform with a browser that does not support frames”. The instances of this class are linked to 0 to N instances of the class PlatformFeature through the composedOf relationship.

PlatformFeature

A platform feature is a characteristic of a platform corresponding to an attribute in the platform model. This class has five attributes:

- *idFeature*: identifier of the platform feature.
- *category*: category of the feature in the platform model (hardwarePlatform, softwarePlatform, networkCharacteristics, browserUA).
- *nameFeature*: name of the feature in the platform model, for example screenSize, pointingDevice, keyboard... For a complete list, we invite the reader to refer to section 3.6.
- *nameDisplay*: name of the feature in a format that will be displayed to the user.

6. Tool support

- *valueFeature*: value of the feature, for example (screenSize)=800x600, (pointingDevice)=finger, (keyboard)=phoneKeypad

An instance of PlatformFeature is linked to 0 to N instances of ChangeInPlatformFeature and to 0 to N instances of TargetPlatformIs.

Of course, the three types of conditions mentioned above are intimately linked: for example, if the target platform is a PDA, then some platform features such as the screen size will change, and usability problems such as the presence of horizontal scrolling will occur. However, the distinction makes the consultation of the knowledge base easier for its users.

Reaction

A reaction constitutes the right-hand side of a graceful degradation rule. It can be described with the following features:

- An identifier (idReaction).
- An internal name and a name in a format that will be displayed to the user (displayName).
- A description, in natural language.
- A description of the source user interface, in natural language (input).
- A description of the target user interface, in natural language (output).
- Its type: executable or not.
- Possibly a formal description, when the rule is formalized (see 4.2).
- The level of the Unified reference framework where it can be applied: Tasks&Concepts, Abstract User Interface, Concrete User Interface or Final User Interface. When a reaction may be applied at different abstraction levels (e.g. splitting rule), the highest abstraction level is held.

Reactions can be related by different types of relationships: some reactions are more general than others (e.g.; “reduce the length of the objects of the UI” is more general than “apply a combination of shrinking and cropping to all images that cause horizontal scrolling”), some reactions entail other reactions (e.g. “delete an image” entails “delete the related caption”, or “reduce the font size of the text presented in a graphical CIO” entails “reduce the size of the related graphical CIO”), ...

A reaction can be documented thanks to four additional classes: Synonyms, Reference, Author and Example.

Synonyms

A reaction is characterized by a name, but it can own synonyms or equivalents in another language. Objects of this class possess the following attributes: an identifier (*idSynonym*), a name (*nameSynonym*) and a language (*lgSynonym*).

Reference

This class contains documents (papers, books, Web page...) referencing the reaction. A reference is described by the following attributes:

- *idReference*: identifier of the reference.
- *title*: title of the reference.
- *source*: complete description of the source (for example, publisher, publication year, ... depending on the category of the reference and the availability of the information), if any.
- *URL*: URL of the Web page where the reference was found, if any.
- *comments*: optional comments on the reference.

A reference is linked to 0 or several instances of the Author class. For a given instance of this relationship, an author has a *rank* with respect to the reference.

Author

Objects of this class are authors of documents inserted in the Reference class. An author has a *name* and an (optional) *first name*.

Example

This class contains graphical examples illustrating a reaction. An example can be related to a reference (refExample relationship) or not. An example is described by an identifier (*idExample*) and the name of the file where the example is stored (*fileName*).

In order to facilitate the consultation and management of large-size or multilingual corpora of rules, the rules were regrouped into sections, each section belonging to one or several knowledge bases.

Section

Sections correspond to categories discussed in section 4.1, for example LayoutModificationRules, ResizingRules, MovingRules, SplittingRules, TaskModificationRules... A section is described by an identifier (*idSection*), an internal *name* and a name in a format that will be displayed to the user (*nameDisplay*). Sections are

organized into a hierarchy (for example, `resizingRules` are a subset of `LayoutModificationRules`).

KnowledgeBase

A knowledge base is meant to regroup several sections regrouping GDRules written in the same language. A knowledge base is described by an identifier (*idBase*), an internal *name* and a name in a format that will be displayed to the user (*nameDisplay*).

6.1.2.c Implementation

The class diagram described above was then translated into a relational schema of 21 tables (Figure 6-2), which has been implemented using the relational database management system MySQL 5.0. The database server is accessed using the scripting language PHP 5.0, and a simple Web-based interface¹³ using HTML and CSS permits the insertion, the display and the management of rules, examples and references. Figure 6-3 shows the functionalities offered by the Web-based interface. Figure 6-4 is a screenshot showing the results of a query to the database.

¹³ <http://www.isys.ucl.ac.be/bchi/research/salamandreTrav/GDbase/homeConsultation.php>

6. Tool support

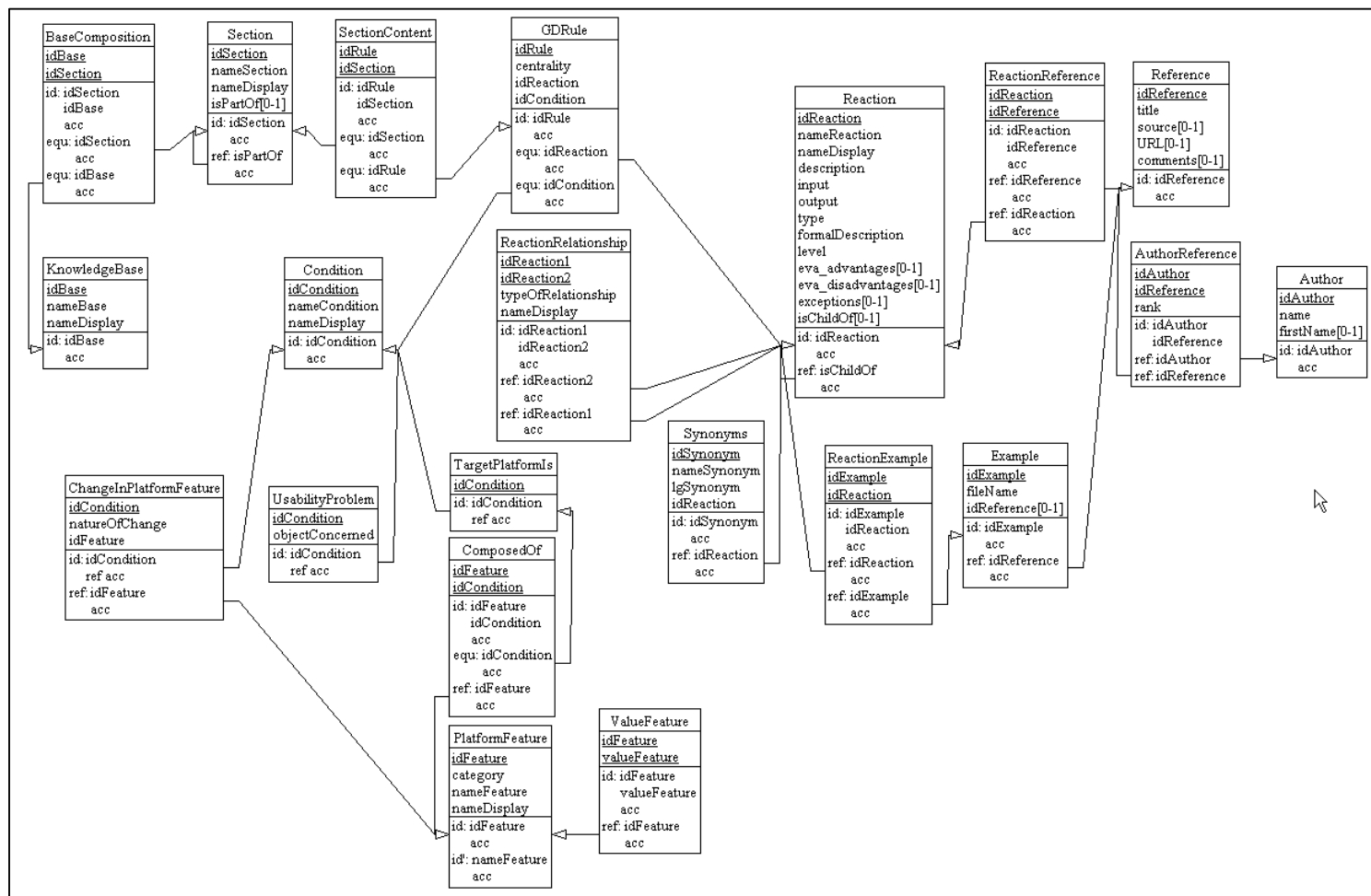
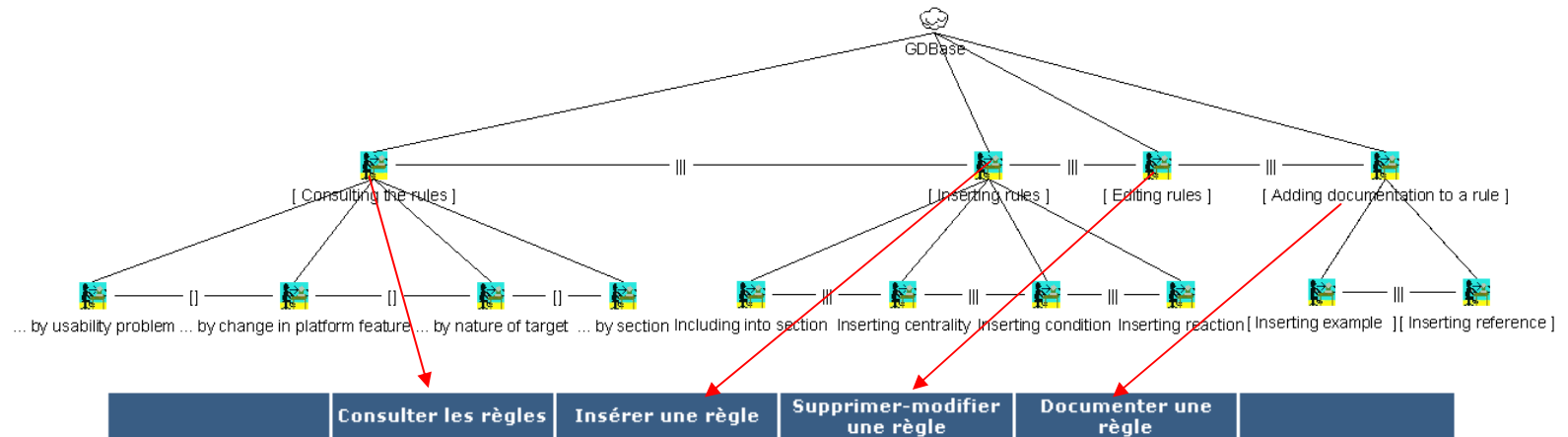


Figure 6-2 Relational model of the database

6. Tool support



Consultation du catalogue de règles

Sélectionnez une condition de déclenchement:

Problème d'utilisabilité:

Interacteur indisponible

Rechercher

OU Modification d'une caractéristique de la plate-forme:

Augmentation de la résolution de l'écran

Rechercher

OU Changement de plate-forme:

La plate-forme cible est un communicateur

Rechercher

Figure 6-3 Functionalities offered by the Web-based interface to the knowledge base

6. Tool support

Modification des images	
Nom de la règle	Combiner redimensionnement et rognage
Description	Réduire l'image à son sujet central en éliminant les pourtours et la redimensionner au maximum
Entrées	Une image
Sorties	Cette image réduite à son sujet central et réduite en taille
Avantages	1. Gain d'espace à l'écran 2. Diminue l'espace mémoire requis 3. Information principale conservée
Inconvénients	Pas toujours applicable aisément
Exemples	Exemple
Nom de la règle	Remplacement des cartes interactives par des liste de liens
Description	Remplacement d'une carte interactive par une liste de liens permettant une navigation équivalente.
Entrées	Une carte interactive
Sorties	Une liste de liens équivalente du point de vue de la navigation
Avantages	1- gain de place important à l'écran 2- diminue l'espace mémoire requis 3- information principale conservée
Inconvénients	1- perte de qualité visuelle et de vue d'ensemble 2- perte de l'information relative à la localisation
Nom de la règle	Modification du format d'un élément graphique
Description	Remplacement d'un élément graphique dans un format donné (ex : BMP, JPG, ...) par une version de cet élément dans un format qui requiert moins d'espace de stockage (ex : BMP -> JPG, JPG -> GIF, ...) ou dans une moindre résolution. On peut recourir soit à une conversion simple (automatique), soit à la production indépendante d'une deuxième version plus légère (ex : dessin, icône, ...)
Entrées	Une image dans un format donné
Sorties	Une image dans un format comprimé
Avantages	1- gain de place important à l'écran 2- diminue l'espace mémoire requis 3- information principale conservée
Inconvénients	- perte de qualité visuelle
Nom de la règle	Remplacer l'image par un texte alternatif
Description	Remplacer l'image par un texte alternatif (dans le cas d'un site Web, par le contenu de la balise ALT)
Entrées	Une image
Sorties	Son texte alternatif
Avantages	1. Gain de place important à l'écran 2. Diminue l'espace mémoire requis 3. Convient aussi aux plate-formes ne supportant pas les images
Inconvénients	1. Perte d'information 2. Perte en termes d'agrément de la présentation
Exemples	Exemple

Figure 6-4 The consultation of rules in the interface to the database.

6.1.3 Support of the adaptation process in the knowledge base

According to Dieterich et al. [Diet94], the adaptation process can be broken down into four stages, each of them being controlled either by the user, either by the system:

1. The *initiative* stage is the decision of one of the agents (user or system) to suggest an adaptation.
2. During the *proposal* stage, alternatives for adaptation are proposed.
3. At *decision* stage, one of these alternatives is chosen.
4. Finally (*execution* stage), the selected alternative is executed.

Dieterich's framework includes other classification criteria such as the goals of adaptation and the strategies of adaptation (when the changes are made).

In the approach supported by the knowledge base, the goal of the adaptation is to obtain a usable UI on the target device, while taking into account the cross-platform consistency among the different versions of the UI. The rules are applied at design time. The human designer controls the majority of the adaptation stages:

1. The initiative stage is under responsibility of the human designer, who detects a launching condition (classes Condition, UsabilityProblem, ChangeInPlatformFeature, TargetplatformIs of the class diagram).
2. The proposal stage is covered by the system, which is able to link a given launching condition to one or several possible reactions (aggregation relationships linking the Condition and Reaction classes to the compound class GDRule).
3. The decision is taken by the human designer. He or she can rely on information from the database in order to perform the best choice (information of the GDRule class on the centrality of the rule, its advantages and disadvantages and possible exceptions).
4. The execution is manual. Information on how to adapt may be found in the Reaction class (description of the reaction, of its input and output, formal description) and in the ReactionRelationship association class (permits to identify other reactions linked to the first, basic reaction).

Identifying the entities supporting each adaptation stage among the main classes and relationships of our class diagram of GDRule leads to the partition shown on Figure 6-5. We observe furthermore that each main concept identified (GDRule, Condition, Reaction, aggregation relationships) belongs to a separate stage in Dieterich's framework, which is a strong argument in support of the division of concepts in our meta-model.

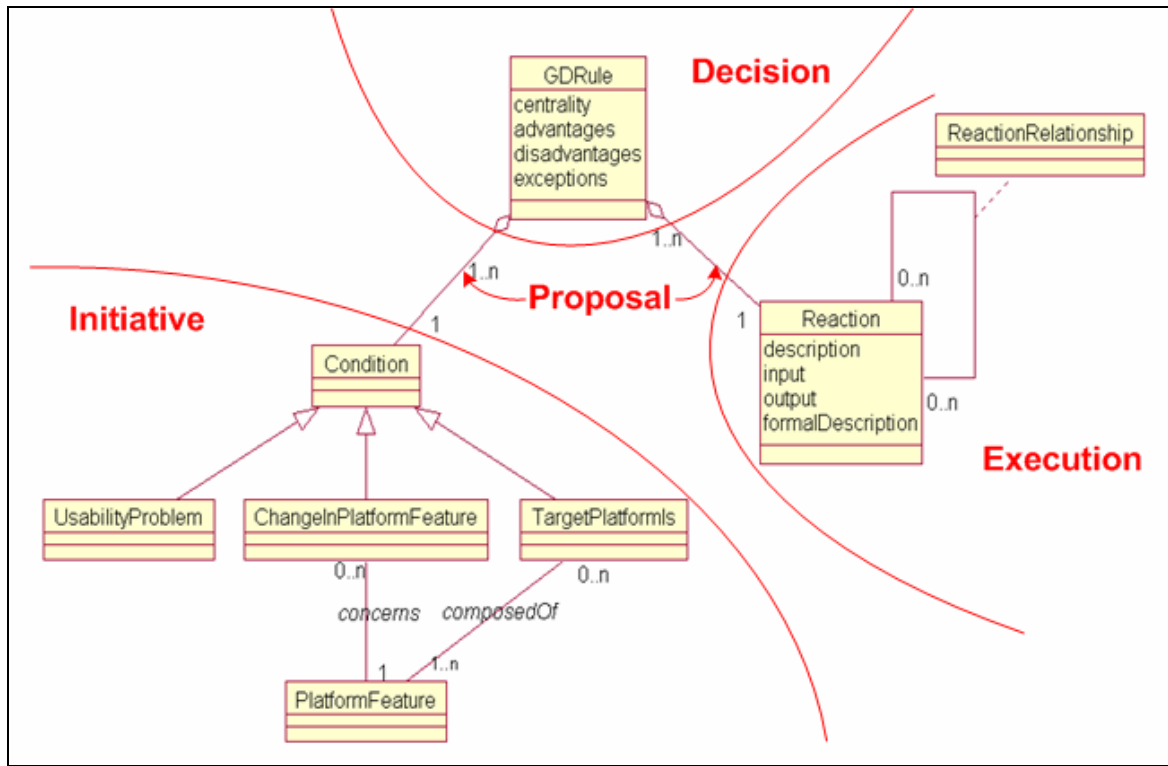


Figure 6-5 Support of Dieterich's four adaptation stages in the knowledge base

6.2 A tool support for GD rules

6.2.1 A plug-in to the GrafiXML editor

GD rules can be applied not only manually, as heuristics for a human designer wishing to adapt a UI to a more constrained platform, but also automatically, with the help of a MB-UIDE. A collection of GD rules has thus been implemented in a plug-in to the GrafiXML editor, which permits building graphically user interfaces that are saved in UsiXML.

The rules have been implemented as transformations on UI descriptions in the GrafiXML's editor specific format, and further propagated to descriptions in UsiXML. We have chosen this option because it provided a quick way to visualize the results of a GD rule, relying on a pre-existing tool which represented several months of development. The main drawback of this choice is that the current implementation of the rules is totally dependent of GrafiXML's specific format (i.e. they work on a collection of Java classes representing UsiXML's graphical objects) and can not be exported to other tools. However, the small implementation attached to GrafiXML permits demonstrating the feasibility of the approach, to evaluate the quality of the UT's that can be produced using GD rules semi-automatically, and to collect users' impressions on the approach and the tool.

6.2.2 Functionalities

The rules to be supported have been selected on a criterion of frequency and utility, based on the experience acquired during the case studies.

The rules are regrouped into sections, corresponding to the sections identified above (6.1.2.b). Currently, five sections are proposed, as shown on Figure 6-6: resizing rules, moving rules, interactor transformations, image transformations and splitting rules. The rules included in each section are detailed in Table 6-2.

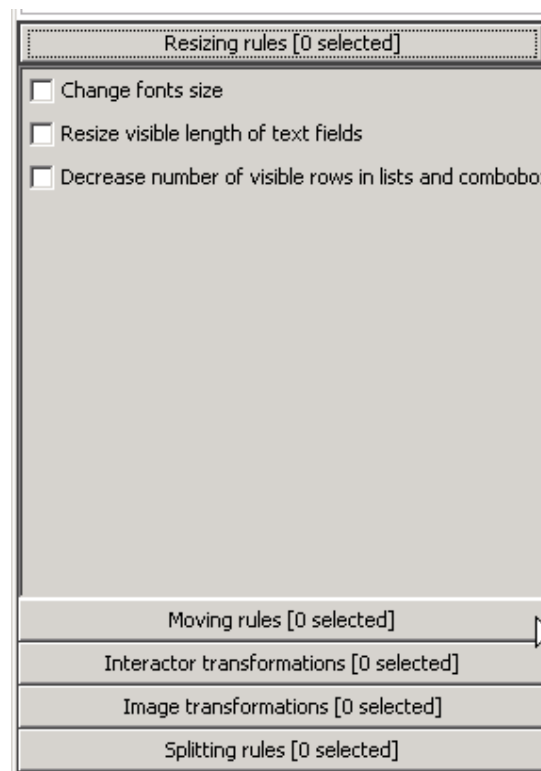


Figure 6-6 The five groups of rules in the GD plug-in

6. Tool support

Resizing rules
Font size reduction (reduce font size to a given minimum)
Input field shrinkage (reduce the visible length of text fields, without reduction of the maximal length)
Number of visible list items decrease (reduce the number of items in list boxes and combo boxes that are visible without scrolling.)
Moving rules
Vertical repositioning in columns (vertical repositioning of the boxes structuring the source UI into one or several columns. The difference between the column's sizes must be minimized.)
Vertical alignment of group box content (vertical repositioning of all the elements inside the selected boxes, without modification of the relative position of the boxes)
Interactor transformations
Interactor substitution (substitution of an interactor by another interactor supporting the same data type and the same functionalities)
Image transformations
Replace image by Alt (replace images by a textual description)
Scale and crop (reduce images to their core subject by truncating their edges and minimize their size as much as possible)
Splitting rules
Interaction space splitting (with different navigation types)

Table 6-2 Detail of the plug-in's five sections

Each selected rule can be given parameter values (Figure 6-7). Default values are given for each parameter. A small textual description is associated with each rule, and a hyperlink provides access to the complete description of the rule in the knowledge base (Figure 6-8).

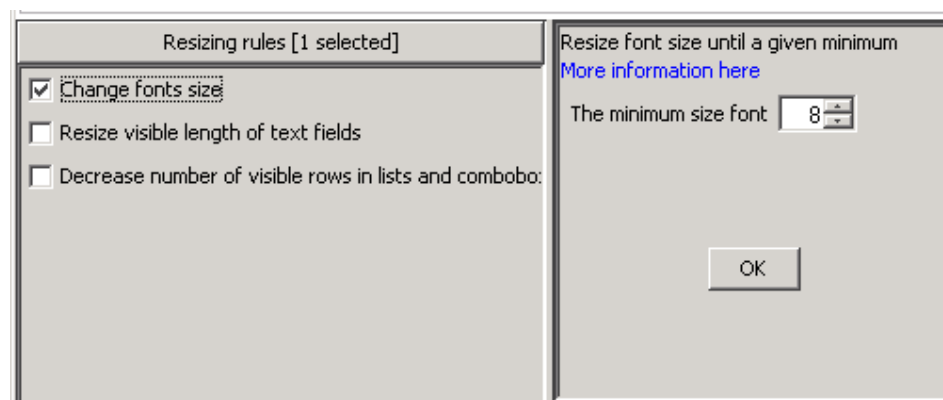


Figure 6-7 Details panel linked to a selected rule

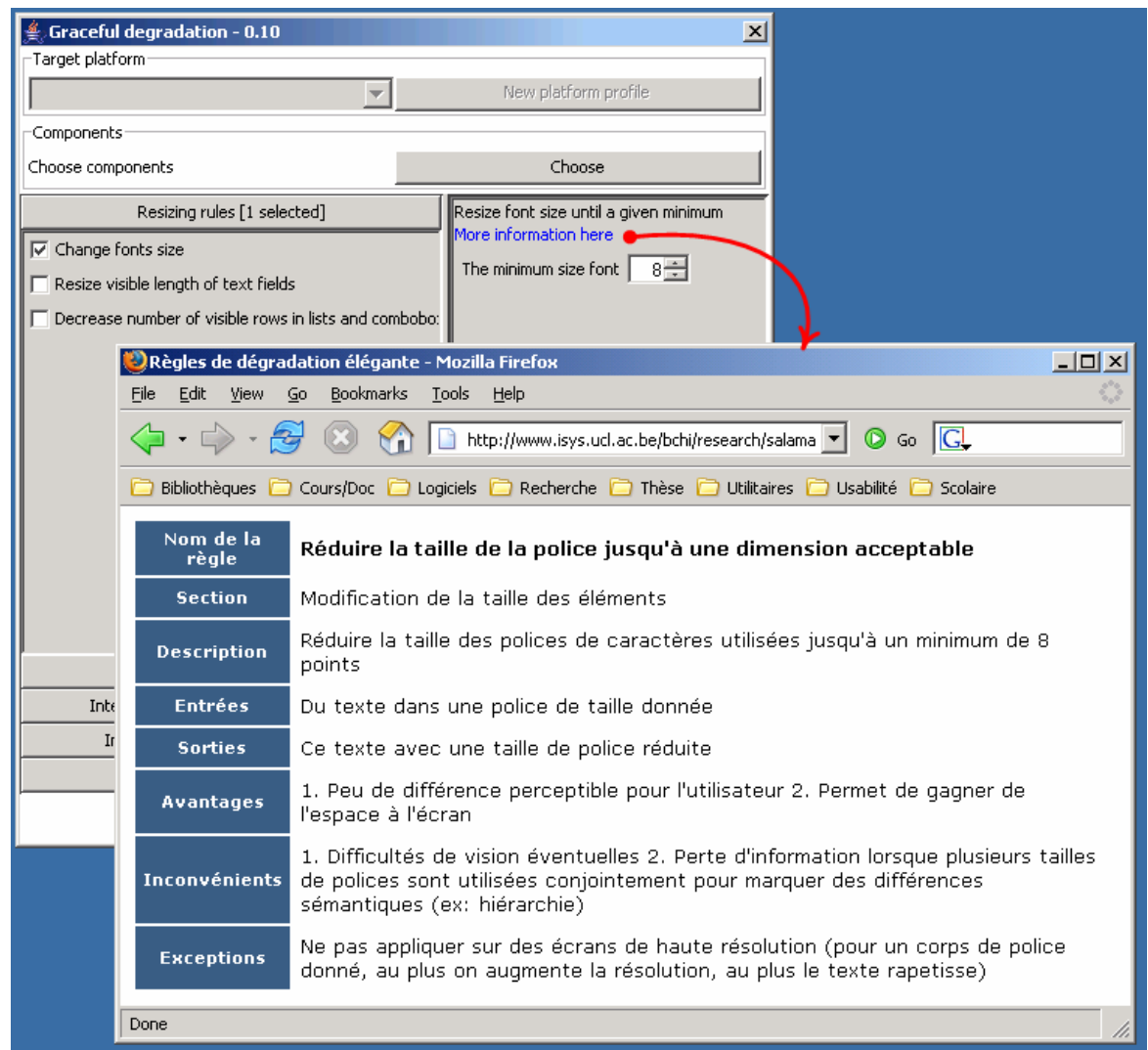


Figure 6-8 Access to the knowledge base from the GD plug-in

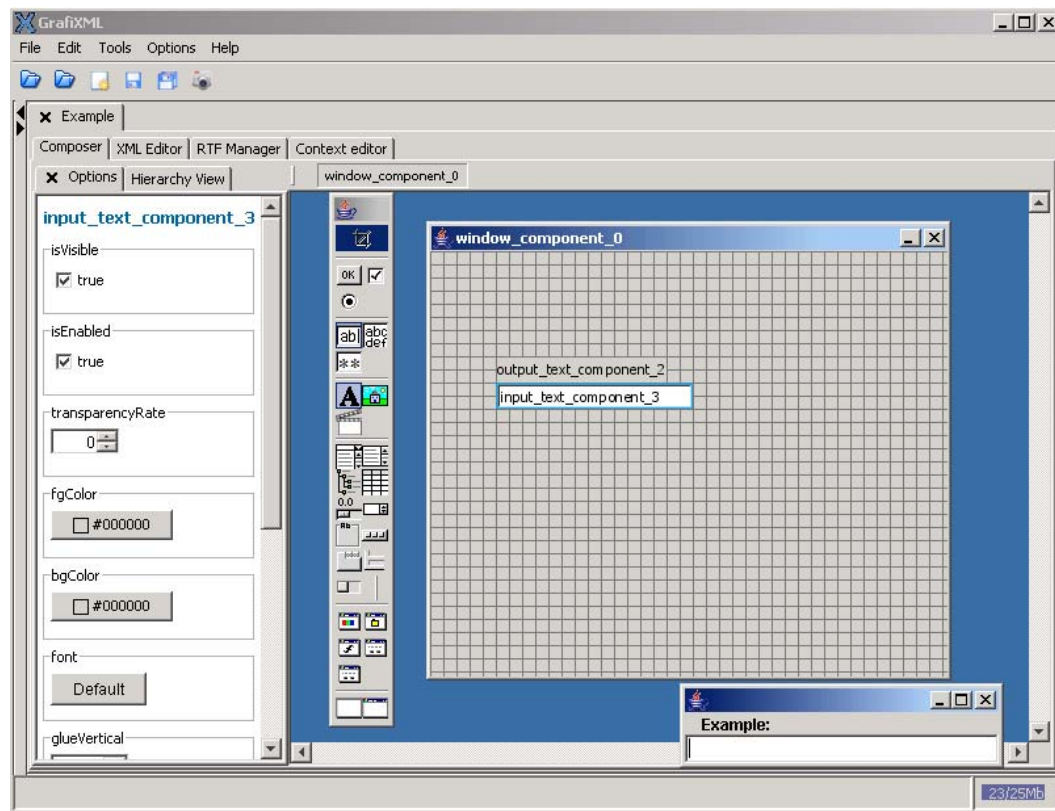
The complete list of rules to be implemented in the plug-in and their precise description can be found in Annex E. The implementation of these rules is still ongoing work performed by our team's developer: at the time of writing these lines, a few rules have not been implemented yet.

6.2.3 Scenario supported by the tool

The GD tool was designed to support the following scenario:

- (1) The user produces a source UI in UsiXML. This specification may have been built by hand, using any XML editor or text editor, or with the graphical editor GrafiXML, or recovered from existing code using reverse engineering tools such as Vaquita [Boui04]. The source UI is then opened within GrafiXML (Figure 6-9).

6. Tool support



```
<cuiModel id="Example-cui_11" name="Example-cui">
  <window id="window_component_0" name="window_component_0"
    width="400" height="350">
    <box id="box_1" name="box_1" type="vertical">
      <outputText id="output_text_component_2"
        name="output_text_component_2"
        content="/uiModel/resourceModel/cioRef[@cioId='output_text_component_2']/resource/@content"
        defaultContent="Example:" isVisible="true"
        isEnabled="true" isBold="true" textColor="#000000"/>
      <inputText id="input_text_component_3"
        name="input_text_component_3" isVisible="true"
        isEnabled="true" textColor="#000000" maxLength="50"
        numberOfColumns="15" isEditable="true"/>
    </box>
  </window>
</cuiModel>
<contextModel id="Example-contextModel_11" name="Example-contextModel">
  <context id="Example-context-fr_FR_11" name="Example-context-fr_FR">
    <userStereotype id="Example-stfr_FR_11" language="fr_FR" stereotypeName="Example-stfr_FR"/>
    <platform id="Example-platform_11" name="Example-platform"/>
    <environment id="Example-env_11" name="Example-env"/>
  </context>
</contextModel>
<resourceModel id="Example-res_11" name="Example-res">
  <cioRef cioId="output_text_component_2">
    <resource content="Example:" contextId="Example-context-fr_FR_11"/>
  </cioRef>
</resourceModel>
```

Figure 6-9 A very simple UsiXML specification in textual format and in the GrafiXML composer

6. Tool support

- (2) The user opens the GD plug-in from the “Tools” item menu of GrafiXML.
- (3) He or she selects a platform profile (firstly, the default platform profile will be Web-based UI on PDA’s).
- (4) He or she can select a set of GD rules among the different panels of the plug-in. For each rule:
 - A small description and an illustration are given, and access to the knowledge base is possible (Figure 6-10).
 - Parameters values may be specified (Figure 6-11).
 - The list of components to which the rule applies may be specified (Figure 6-12).

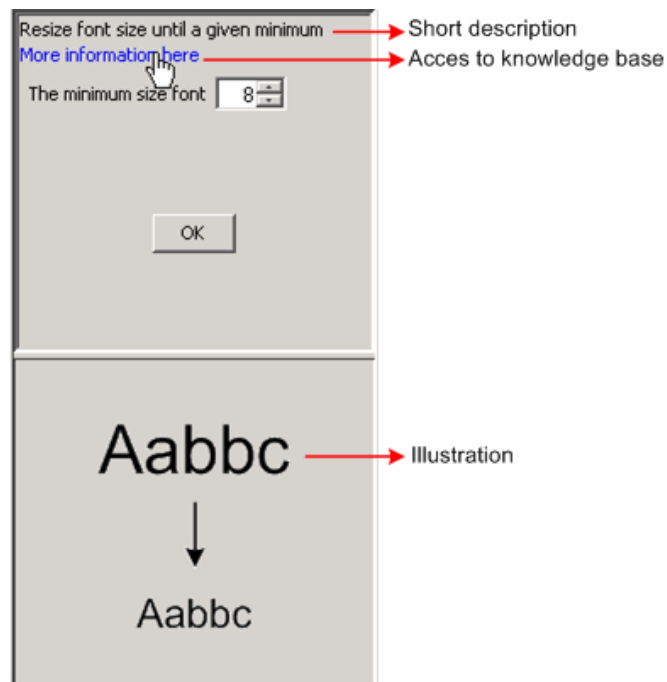


Figure 6-10 Description of a rule in the GD tool

6. Tool support

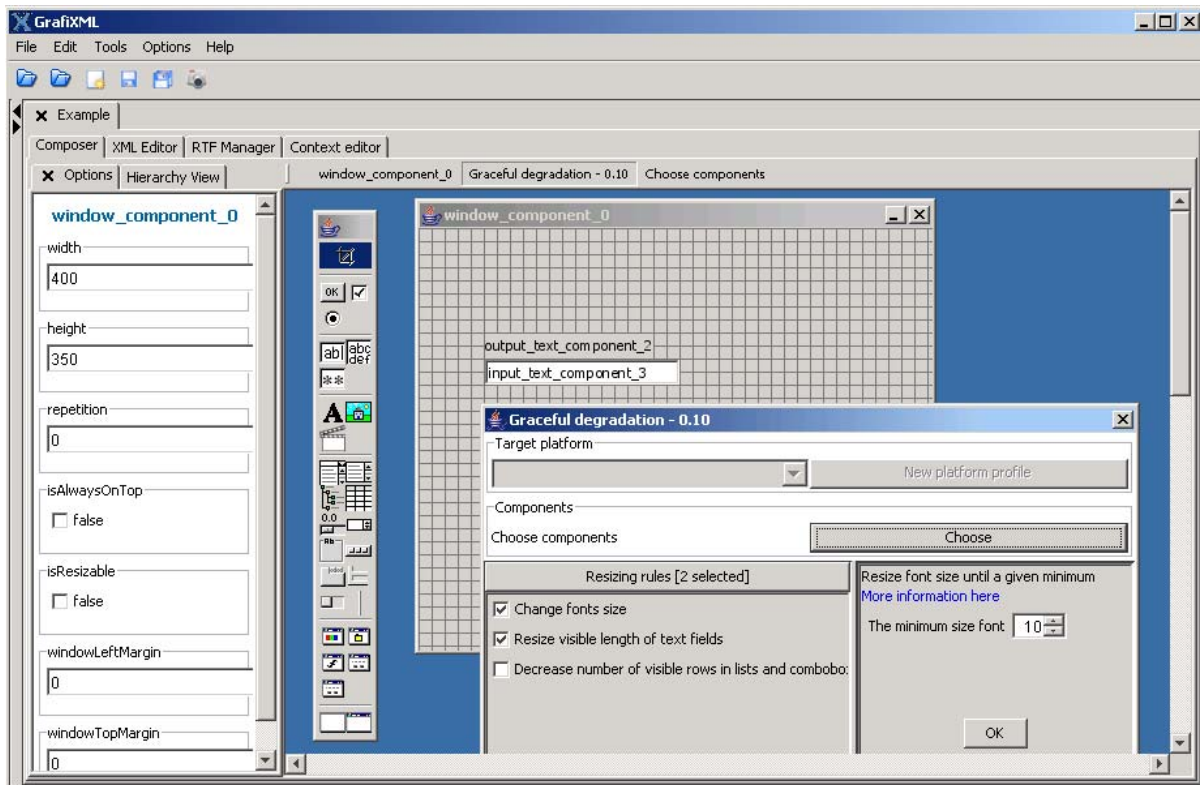


Figure 6-11 Selection of rules and parameters in the GD tool

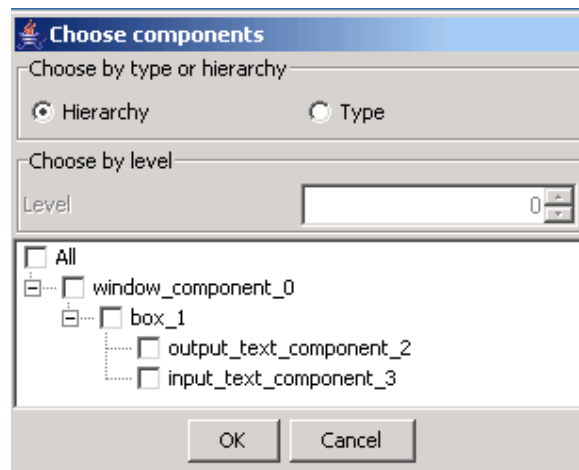


Figure 6-12 Selection of the components to which the rule applies

- (5) He or she can preview the result of the application of the selected rules
- (6) ... or save the resulting interface, which now appear as a distinct project in a new tab of the tabbed panel.

6.2.4 Support of the adaptation process in the GD plug-in

The semi-automatic approach supported by the GD plug-in also pursues the goal of obtaining a usable UI on the target device, while minimizing the gap between the different versions of the UI. Again, the rules are applied at design time. Although the human designer still controls most of the process, more adaptation stages are covered by the system:

1. The initiative stage is still under responsibility of a human designer, wishing to adapt a UI to a given target platform.
2. The proposal stage is covered by the system, which proposes a list of suitable adaptation rules.
3. The decision is taken by the human designer, who selects a rule (or a set of rules) and specifies the value of its parameters. Each rule implemented in the GD plug-in is linked to the corresponding rule in the knowledge base, so that the descriptions, recommendations and examples of the database are also accessible from the automatic tool.
4. The execution is automatic.

Stages 3 (decision) and 4 (execution) may be performed iteratively : after previewing the result of the application of a given set of rules, the designer may wish to apply a second set of rules to the same source interface, or to test additional rules on the first result.

6.2.5 User testing

In order to obtain feed-back on the usability and utility of the GD plug-in, we conducted a small, informal user study during the development of the tool.

6.2.5.a Experiment

Seven software industry professionals (6 men, 1 woman) participated in the experiment. The age of the participants ranged from 23 to 31. Of these participants, 3 were identified as “experienced developers”, given that they had at least 6 years experience in designing and developing information systems and 3 years experience in developing multiplatform information systems, and 4 were categorized as “novice developers”, as they had at most 1 year experience in the field.

A development version of the plug-in was presented to the participants, with GrafiXML opened on a pre-existing project, in order to avoid them the need to specify a source user interface. No specific task was assigned, because the prototype was far from bug-free, but subjects were asked to freely experiment with the tool, with the following question in mind: “One of my clients wants this graphical interface to be ported to a small platform, might this tool help me doing my work?”

After a few minutes, participants were asked to fill a standard usability questionnaire (the IBM Computer Usability Satisfaction Questionnaire CSUQ [Lewi95], shown on Table 6-3). Free comments were also encouraged.

6. Tool support

Question statement
1. Overall, I am satisfied with how easy it is to use this system
2. It was simple to use this system
3. I can effectively complete my work using this system
4. I am able to complete my work quickly using this system
5. I am able to efficiently complete my work using this system
6. I feel comfortable using this system
7. It was easy to learn to use this system
8. I believe I became productive quickly using this system
9. The system gives error messages that clearly tell me how to fix problems
10. Whenever I make a mistake using the system, I recover easily and quickly
11. The information (such as online help, on-screen messages, and other documentation) provided with this system is clear
12. It is easy to find the information I needed
13. The information provided for the system is easy to understand
14. The information is effective in helping me complete the tasks and scenarios
15. The organization of information on the system screens is clear
16. The interface of this system is pleasant
17. I like using the interface of this system
18. This system has all the functions and capabilities I expect it to have
19. Overall, I am satisfied with this system

Table 6-3 The CSUQ questionnaire

6.2.5.b Results

The analysis of the CSUQ questionnaire showed that the overall average response of the designers to the plug-in was slightly above average. It is not possible to draw many conclusions from this result, for different reasons:

- The experiment did not reflect any normal usage of the tool, since the users were not actually performing the task.
- The interviewer was known by the users, so that the satisfaction ratings obtained may have been distorted.
- The number of respondents was small, and all of them belong to the same team into the same organization.
- The questionnaire does not permit any evaluation of the utility of the tool, since it is focused on usability aspects.

Nevertheless, the users offered helpful comments towards further simplifying and enhancing the tool, which has been and will be used in upgrading the system. The identified weak points of the system mainly concerned the limited documentation and on-line help facilities provided (Q11) and the quality of error management (Q9, Q10). This was a known shortcoming of the prototype plug-in, attributed to restricted resources at development time. Another identified weakness was the lack of an undo facility. Furthermore, some of the respondents had specific requests for additional functionalities they would have liked to see supported in future versions of the system. In particular, one participant expressed the wish that the plug-in should be equipped with some kind of templates providing predefined layouts well-adapted to the target platform. Neither the combination of templates with GD rules, nor the possibility to enhance UsiXML with a template

mechanism has been explored for the moment, but this could be an interesting direction for future work. Such a use of platform-specific templates for prototyping cross-platform user interfaces is not unknown in the literature [Lin03] [Nich04]. However, existing approaches are targeted to specific domains: they provide, for example, templates for a shopping card or for a UI controlling a given appliance device, such as a tape recorder.

In general, the evaluation offered valuable insight into the functional and the interaction characteristics of the system. Participants' comments showed that the plug-in could be used in different scenarios typical in their work environment:

1. A pre-existing system must be ported as quickly as possible to a small platform. In this case, the plug-in can be used as a rapid prototyping tool.
2. A pre-existing system must be ported to a small platform, and the customer comes with sketches of the envisioned design. In this case, the plug-in in its current version is less useful, but additional functionalities such as automatic recognition of usability problems or realistic, platform specific preview, should be very helpful.
3. A pre-existing system must be ported to a small platform, and the customer asks for design propositions. In this scenario, the plug-in can produce rapidly different design proposals to discuss with the client.

Chapter 7 Case studies

Our case studies cover the two types of scenarios envisioned for the use of GD rules:

- Manual adaptation of an existing user interface to a more constrained platform by a human designer: scenario applied on the ARTHUR case study.
- Semi-automatic adaptation using the GD plug-in described in section 6.2.

7.1 ARTHUR

7.1.1 Introduction

ARTHUR (Architecture de Télécommunications Hospitalières pour les Services d'Urgences) is an information system developed for emergency departments and other related units (intensive care unit, biology test lab, radiology department ...) in Belgian hospitals [Amou05]. ARTHUR provides computerized support for medical and nursing tasks as well as for administrative tasks in the emergency department. ARTHUR is multi-device and runs on workstations, Pocket PCs and a wall display.

We have collaborated in the design of the first version of the ARTHUR system, especially the patient records management system. The technologies chosen for the user interfaces of this version were the Web standards HTML, XSLT and XML, with slight differences between the language versions on the different devices: for instance, the Pocket Internet Explorer on the PDA only supported JavaScript 1.1 and HTML 3.2, with minor exceptions for some tags, and CSS was not available.

The differences in web standards, screen size and resolution imply big differences in design and implementation between the user interfaces on the different devices. Figure 7-1 illustrates this problem on a screenshot of the ARTHUR prototype. From this screen, members of the medical team (doctors and nurses) must consult and modify nursing records. Figure 7-1 shows the desktop version of ARTHUR and highlights some elements that are likely to cause design changes between the ARTHUR desktop and PDA versions.

7. Case studies

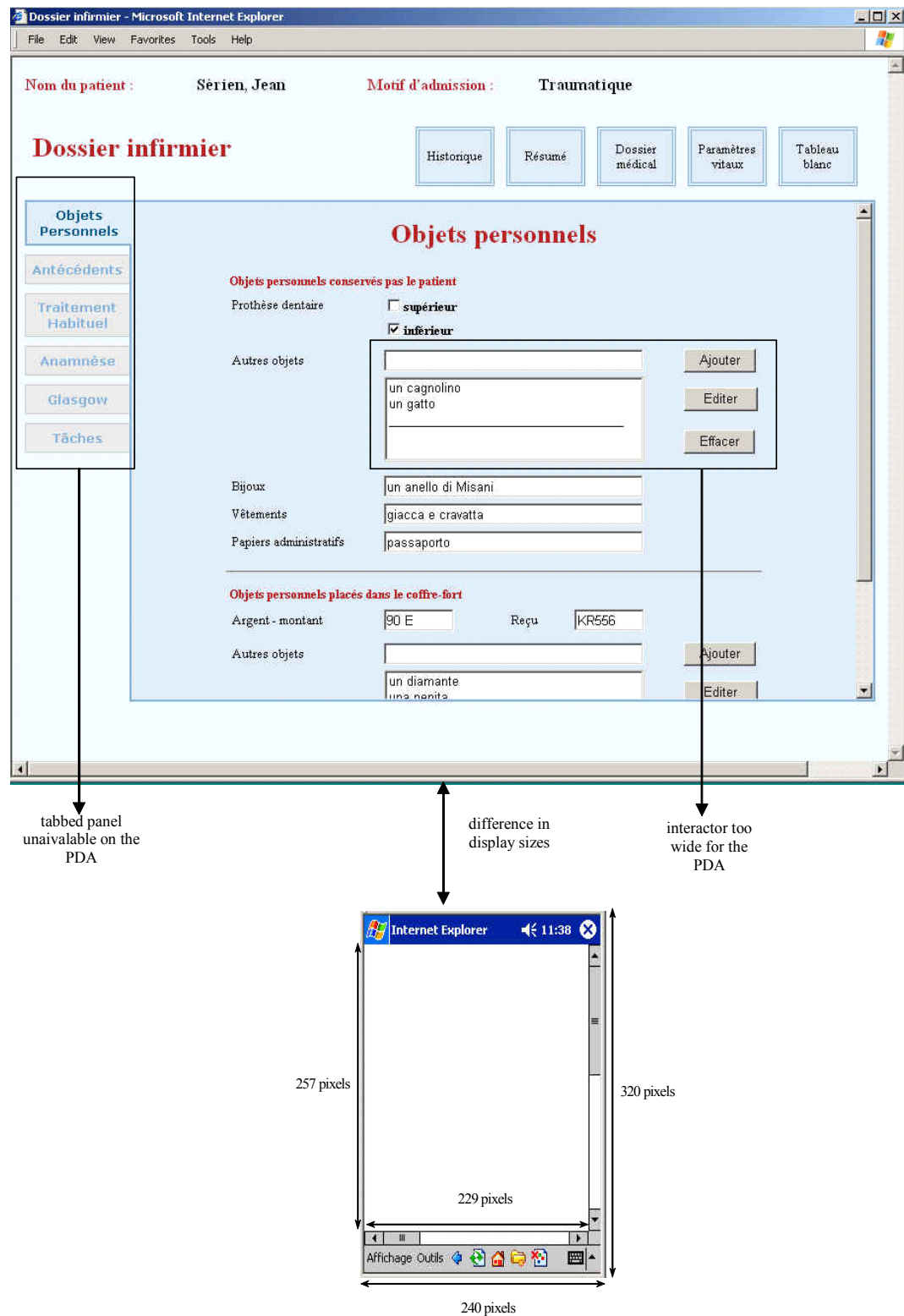


Figure 7-1 Problems raised by ARTHUR's user interfaces

The user interfaces of ARTHUR have been designed using an iterative and task-based approach. Task models were built after observing members of the medical staff performing their tasks. Afterwards, mock-ups of the UI were drawn and modified according to the comments of the staff until reaching a satisfying version. The first prototype of the ARTHUR information system supported the patients' health records management, as illustrated in Figure 7-2.

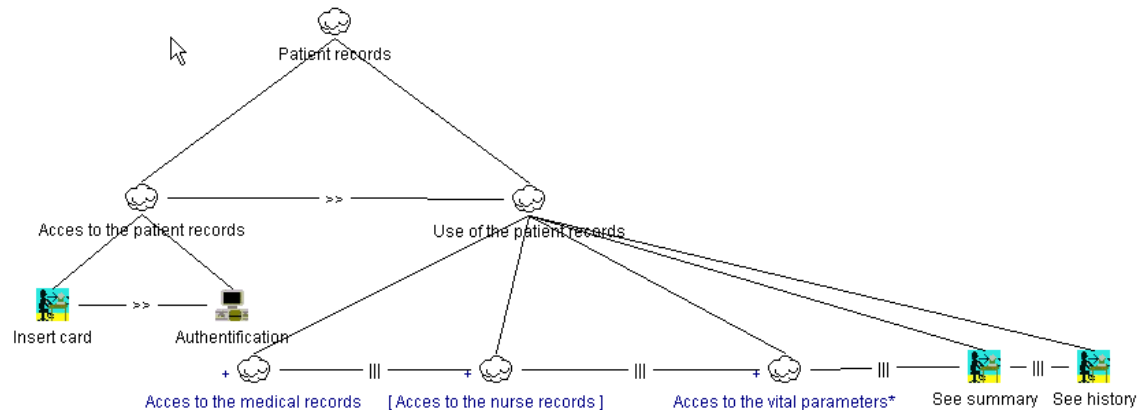


Figure 7-2 Task model of the ARTHUR prototype

After authentication (this task does not require any display), the user is allowed to use the patients' health records. This general task is subdivided in five subtasks: access to the medical records, access to the nursing records, access to the vital parameters, view of the summary and view of the patient's history.

The first subtask investigated was the management of the nursing records. The six subtasks implied in the nursing records management are the consultation and modification of data related to the patient's personal effects, past records, regular treatments, symptoms, Glasgow test and health tasks to be carried (Figure 7-3). These tasks have similar substructure: their first subtask is the information visualization that enables the optional modification of one or more fields (add/delete/edit an item). The modifications can then be validated or cancelled (see example on Figure 7-4).

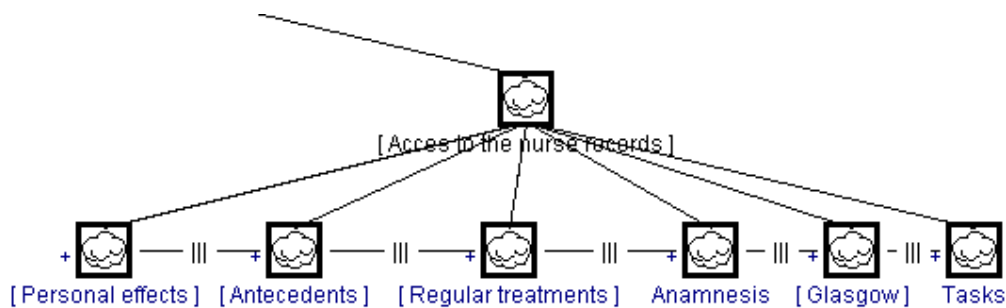


Figure 7-3 Subtasks for the nursing records management

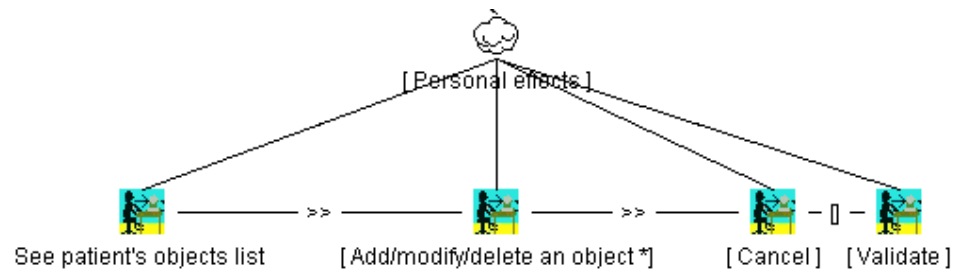


Figure 7-4 Example of a subtask detail

7.1.2 Application of GD Rules

7.1.2.a Source interface

The first ARTHUR interfaces were designed for workstations. Starting from that source interfaces, we have investigated some design options for PDA interfaces (iPAQ Pocket PC), applying our transformation rules. The mock-ups for the different options have been submitted to future users of the ARTHUR system for comments and test.

As an illustration of that design work, we will describe the transformation rules that have been applied to the screen related to the subtask described above, namely the consultation and modification of information related to the patient's personal effects, subtask of the management of the nursing records (see Figure 7-5).

Dossier infirmier - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Nom du patient : Sérien, Jean Motif d'admission : Traumatique

Dossier infirmier

Historique Résumé Dossier médical Paramètres visuels Tableau blanc

Objets Personnels

Antécédents Traitement Habituel Anamnèse Glasgow Tâches

Objets personnels

Objets personnels conservés par le patient

Prothèse dentaire ☐ supérieur ☒ inférieur

Autres objets

Ajouter Editer Effacer

Bijoux

Vêtements

Papiers administratifs

Objets personnels placés dans le coffre-fort

Argent - montant Regu

Autres objets

Ajouter Editer

Figure 7-5 Desktop version of the ARTHUR prototype: source interface

7.1.2.b Application of GD Rules at the Tasks and Concepts level

An option for the PDA interface is to delete all edition tasks and to keep only the consultation tasks, due to the difficulty of text entry with the virtual keyboard or the character-recognition system (Figure 7-6).

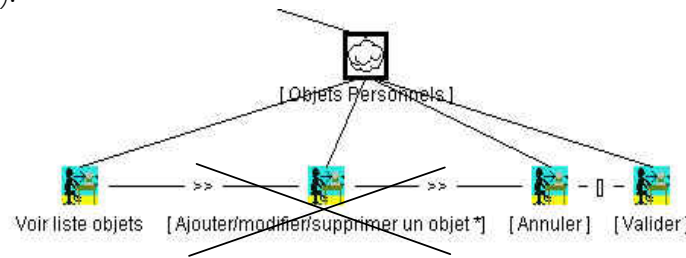


Figure 7-6 An alternative for the PDA version of ARTHUR: deleting edition tasks

7.1.2.c Application of the GD Rules at the Abstract User Interface Level

As the source interface already uses a tabbed panel, we do not wish to split the presentation unit in order to avoid scrolling: too many levels of navigation objects would cause the user interface to become unusable.

7.1.2.d Application of the GD Rules at the Concrete User Interface Level

A lot of transformation rules can be considered at this level:

7.1.2.d.1 Substitution rules

Substitution rules are mandatory when an interactor on the source platform is no longer available on the target platform. It is the case of the tabbed panel on the desktop, which could not be programmed on the PDA, due to the lack of CSS support on Pocket Internet Explorer at that time. Therefore, we had to choose a substitute for the tabbed panel.

Two options have been considered:

- (1) Replacing the tabbed panel by a frameset and the tabs by hyperlinks.
- (2) Replacing the tabbed panel by a frameset and the tabs by image links.

Another substitution has been proposed: the accumulator in the PC interface should be replaced by a reduced version, in order to save screen space, as illustrated on Figure 7-7. (To be precise: this interactor substitution belongs to the final user interface level: the description of these widgets in terms of the interactor model defined above does not permit to distinguish them at the concrete user interface level, they are both Accumulators).

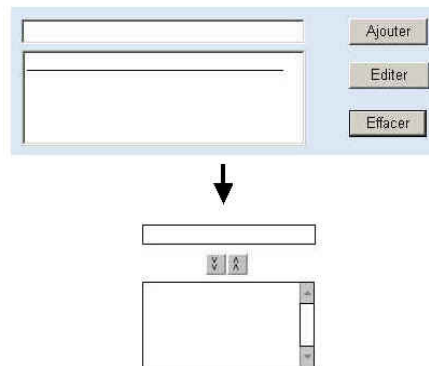


Figure 7-7 Interactor substitution in ARTHUR

7.1.2.d.2 Removability rules

Some interfaces elements do not need to be explicitly labelled in order to be understood: we suggest removing the titles « Nom du patient » and « Motif d'admission ».

7.1.2.d.3 Modification rules

Some labels can be summarized: « Objets personnels chez le patient » and « Objets personnels dans le coffre » should be replaced by a common « Objets personnels » title and two shorter subtitles « Chez le patient » and « Dans le coffre ».

Emphasis could be represented by bold characters instead of red colour (better visibility).

7.1.2.d.4 Moving rules

On the Pocket PC, ergonomic rules advise to place all menus and tabs at the lower edge of the screen in order to allow the user to use them without obscuring the data on the screen with his or her hand. However, it was not possible to move the ARTHUR « menus » to the bottom: the virtual keyboard would mask them when displayed.

However, other moving rules have been applied in order to avoid horizontal scrolling:

- (1) The title and upper buttons could be displayed on the same line on the PC interface but they are better displayed on distinct lines on the PDA.
- (2) In the main frame of the PDA interface, labels are better put above the controls, while there were placed to their left on the source interface.

7.1.2.d.5 Resizing rules

All controls (buttons, edit fields...) have to be resized in order to fit into the screen width.

7.1.2.e Target Interfaces

Two alternative designs for the PDA interface, resulting from the application of the transformation rules described above, are presented on Figure 7-8.

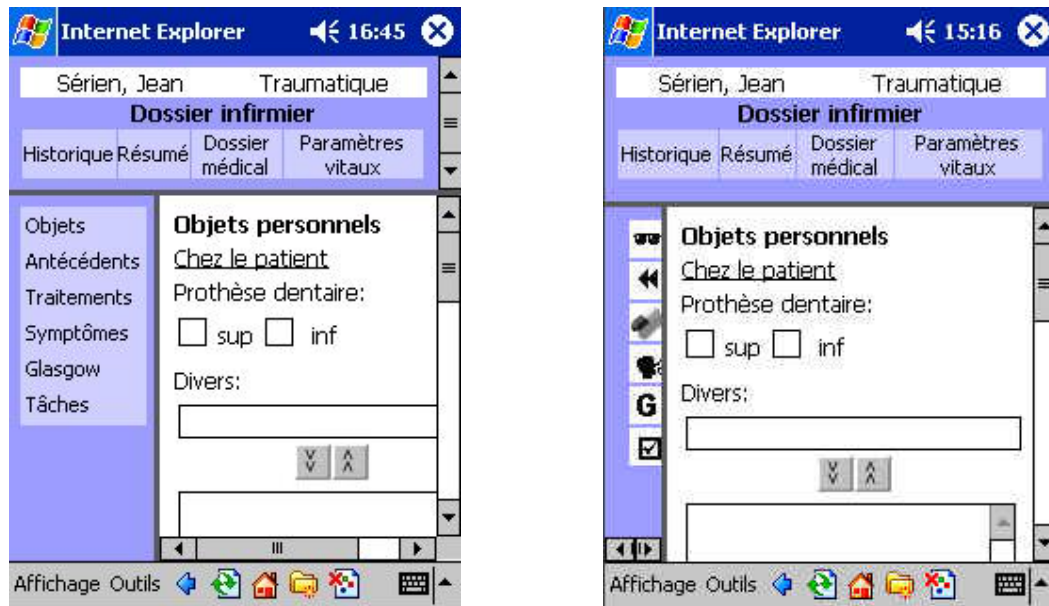


Figure 7-8 Design alternatives for ARTHUR's PDA user interfaces (mock-ups in HTML)

7.1.3 Conclusion

This case study has demonstrated:

- (1) The feasibility of the GD methodology when applied manually: we were able to make design proposals for the PDA version of the ARTHUR system, resulting from an application of the graceful degradation rules to the ARTHUR desktop version.
- (2) The validity of the CAMELEON framework for classifying graceful degradation rules, since each rule used in the design of the ARTHUR PDA interfaces can be situated in that framework.

7.2 Semi-automatic adaptation of the rules: a hotel booking system

The first case study showed how GD rules could be applied manually by a designer wishing to adapt a pre-existing user interface to a more constrained device. This second case study shows how the tool described above (6.2) can support a semi-automatic application of GD rules.

7.2.1 The hotel booking system

A Webmaster is asked to design the user interface of an on-line hotel booking system. This user interface must run both on a traditional desktop and on limited size devices (PDA). The system must permit specifying:

- The hotel location.
- The arrival and departure dates.

7. Case studies

- The number of nights and rooms required.
- The number of guests, adults and children.
- The category of the hotel and the price range.
- The accommodation required.

Using the GrafiXML editor, the designer produces a first, unconstrained version of the user interface (Figure 7-9).

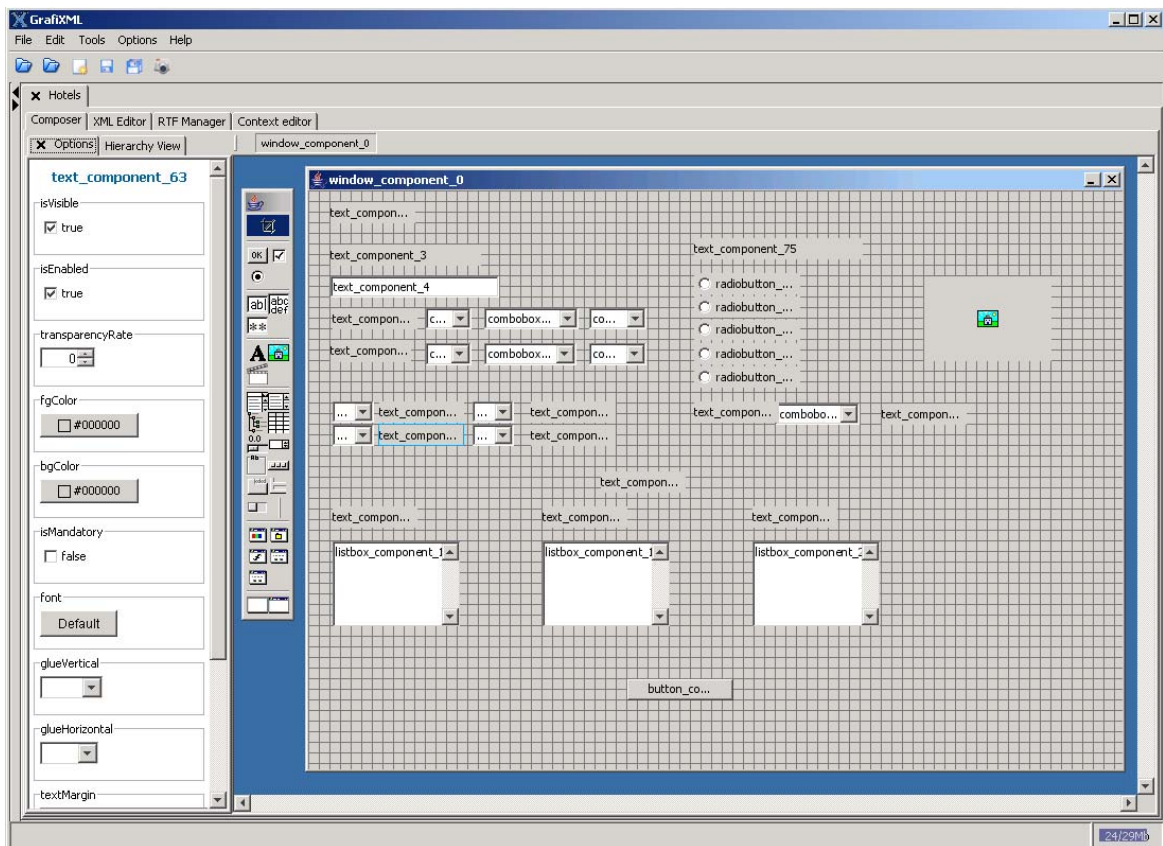


Figure 7-9 The source UI of the hotel booking system in the GrafiXML editor

Different views are available in the editor: design view (Figure 7-9), code (Figure 7-10) and preview (Figure 7-11).

7. Case studies

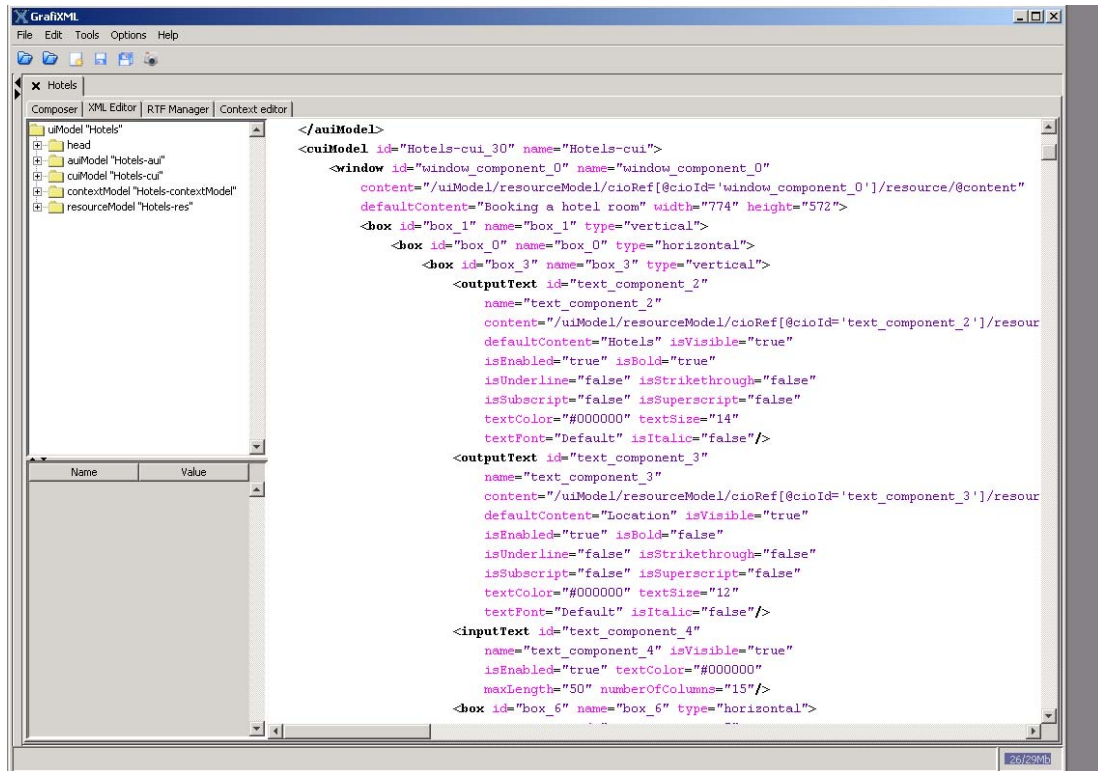


Figure 7-10 UsiXML code of the source UI

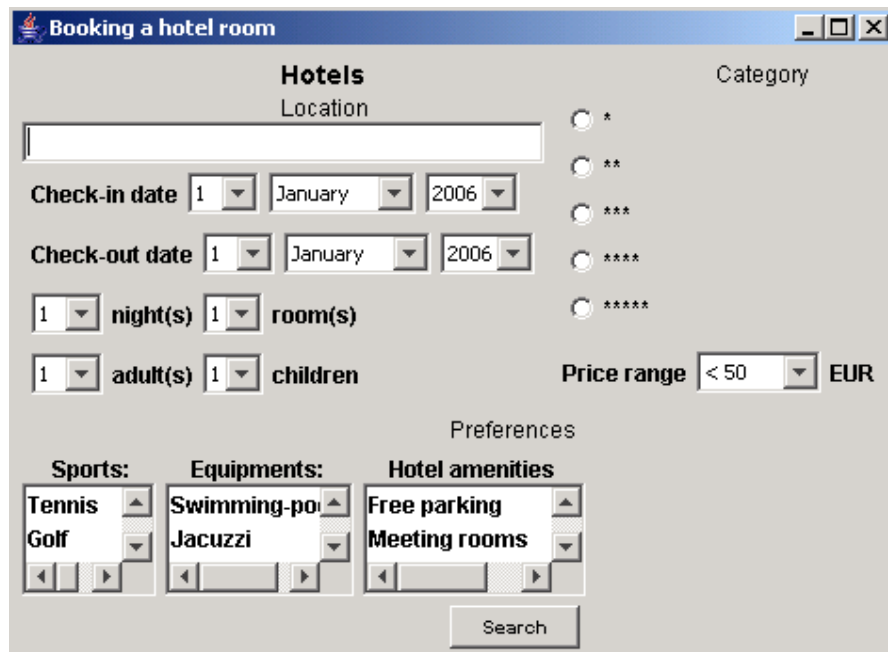


Figure 7-11 Preview of the source UI

7.2.2 Production of the target UIs

The second stage of the design task consists in selecting rules to apply to the source UI. A strategy frequently observed among the few users who tested the system consists in applying first a “global” rule, such as vertical alignment or splitting, and then refining the result by exploring the possibilities offered in the other panes (resizing elements, interactor substitutions...)

Combining the few rules implemented in the plug-in already permits to produces several convincing alternative UIs for a smaller target platform.

Figure 7-12a shows a preview in GrafiXML of a user interface generated by the plug-in when vertical alignment and font resizing are applied. Figure 7-12b shows a slightly different version with the list of radio buttons replaced by a combobox. Figure 7-13 is the result of applying a splitting rule with sequential navigation on the source user interface.

7.2.3 Conclusion

This case study has demonstrated the feasibility of the GD methodology when applied semi-automatically. Even if the set of rules implemented in the prototype authoring tool is limited, we have been able to show how designers can experiment with GD rules and produce several design alternatives in parallel or iteratively without much effort.

Booking a hotel ...

Hotels Location

Check-in date

1

January

2006

Check-out date

1

January

2006

1

night(s)

1

room(s)

1

night(s)

1

children

Category

☐ *

☐ **

☐ ***

☐ ****

☐ *****

Price range

< 50

EUR

Preferences Sports:

Tennis

Golf

Football

Equipments:

Swimming-pool

Jacuzzi

Hammam

Hotel amenities

Free parking

Meeting rooms

Wheelchair accessible

Search

(a)

Booking a hotel ...

Hotels Location

Check-in date

1

January

2006

Check-out date

1

January

2006

1

night(s)

1

room(s)

1

night(s)

1

children

Category

*

Price range

< 50

EUR

Preferences Sports:

Tennis

Golf

Football

Equipments:

Swimming-pool

Jacuzzi

Hammam

Hotel amenities

Free parking

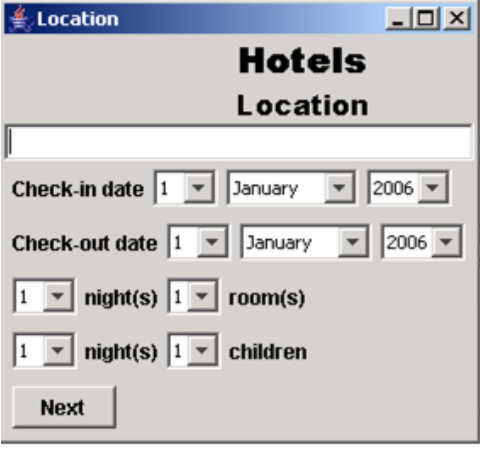
Meeting rooms

Wheelchair accessible

Search

(b)

Figure 7-12 Design alternatives for a small target device (preview in GrafiXML) - 1



Hotels Location

Check-in date 1 January 2006

Check-out date 1 January 2006

1 night(s) 1 room(s)

1 night(s) 1 children

Next



Category

☐ *

☐ **

☐ ***

☐ ****

☐ *****

Price range < 50 EUR

Next



Preferences

Sports:	Equipments:	Hotel amenities
Tennis	Swimming-pool	Free parking
Golf	Jacuzzi	Meeting rooms

Search

Figure 7-13 Design alternative for a small target device (preview in GrafiXML) - 2

Chapter 8 Validation

8.1 Theoretical validation

In Chapter 2, we have identified and described five categories of approaches to the development of UIs for multiple platforms. These approaches were then compared using a set of criteria:

- The production costs.
- The completeness.
- The level of control.
- The usability of the UI produced.
- The cross-platform consistency.
- The guidance.

The theoretical validation of the graceful degradation approach will consist in discussing the characteristics of our methodology against the same set of criteria.

8.1.1 Production costs

In section 2.4.1, we considered that production costs include three facets:

- The cost of developing a first version of the user interface for N platforms;
- The maintenance cost when functions are modified or added;
- The maintenance cost when formats change.

8.1.1.a Cost of developing user interfaces for N platforms

In the GD approach, as in multireification approaches like UIML, the design effort is directly proportional to the number of platform families (i.e. platforms with similar capabilities), but only the source UI has to be designed from scratch, for all the UIs built subsequently, the primary design options are largely reused.

Like in other model-based approaches, no coding is required. Also the specification effort is limited: only one specification language has to be used, and only the source UI has to be entirely specified, the target UIs being produced semi-automatically.

Table 8-1 compares the evaluation of graceful degradation relatively to the five other categories of approaches.

8. Validation

Low	Medium	High
Virtual toolkits Generic clients Transcoding Multireification approach (high level spec) Abstraction-reification	Multireification approach (low level spec) Graceful degradation	Traditional development

Table 8-1 Compared costs of developing UIs for multiple platforms

8.1.1.b Cost of modifying or adding functionalities

Modifying a functionality using graceful degradation requires, at least, one modification at the level of the specification of the source UI. In that case, the modification cost can be very low.

However, the designer could choose, instead of reusing the transformation rules applied on the first source user interface, to select new sets of transformation rules, more adapted to the new source UI. In that case, the modification cost is slightly higher.

In both cases however, no coding is required, so graceful degradation has a modification cost comparable to other model-based approaches, as illustrated in Table 8-2.

Low	Medium	High
Virtual toolkits Generic clients Transcoding Multireification approach (high level spec) Abstraction-reification Graceful degradation	Multireification approach (low level spec)	Traditional development

Table 8-2 Compared costs of modifying/adding a functionality on multiple platforms

8.1.1.c Cost of modifying or adding formats

Again, the performance of the graceful degradation for this criterion is similar to other model-based approaches (see Table 8-3): in all model-based approaches, modifying a format means modifying the generation tool, without any change in the UI specifications. In contrast, the modification of a format (programming language, toolkit...) using non model-based approaches requires modifying all the UIs that rely on this format. The difference is especially important when a large number of systems have been built.

8. Validation

Low	Medium	High
Multireification approach (high level spec)	Virtual toolkits	Traditional development
Multireification approach (low level spec)	Generic clients	
Abstraction-reification	Transcoding	
Graceful degradation		

Table 8-3 Compared costs of modifying/adding a format on UIs deployed on multiple platforms

8.1.2 Completeness

Until now, the model-based community has not demonstrated that automatic generation of UIs starting from high level specifications (Tasks&Concepts level) was able to produce UIs of any kind, as good as those that could be created with conventional techniques. When attempting to generate multiple platform-specific UIs starting from high level specifications, the “low ceiling” problem [Myer00] becomes still more acute.

Other model-based approaches, such as multireification from low-level specifications, appear to have a “higher ceiling”, as a big part of the design process does not need to be automated: the choice of interactors, layouts... is specified by the human designer instead of being deduced from tasks models or domain models (which remains intrinsically difficult). Graceful degradation, as shown on Table 8-4, benefits from the same characteristics as low-level multireification for this criterion.

Low	Medium	High
Multireification approach (high level spec)	Multireification approach (low level spec)	Traditional development
	Abstraction-reification	Virtual toolkits
	Graceful degradation	Generic clients
		Transcoding

Table 8-4 Compared completeness of the development approaches

8.1.3 Level of control

Another criticism addressed to model-based development is that those techniques generate unpredictable results: the connection between specification and final user interface is difficult to understand and control [Myer00].

On the other side, graceful degradation, as shown on Table 8-5, offers a better level of control. The approach relies on an explicit set of rules, fully documented and accessible. It offers to the designer a full control on the selection of those rules. The results of the application of a rule may be previewed.

8. Validation

Low	Medium	High
Multireification approach (high level spec)	Transcoding Multireification approach (low level spec) Abstraction-reification Graceful degradation	Traditional development Virtual toolkits Generic clients

Table 8-5 Compared level of control offered by the development approaches

8.1.4 Usability

Obviously, the usability of a user interface does not depend only on the chosen development technique. Nevertheless, as shown on Table 8-6, the approaches relying on transcoding or reverse-engineering, generic clients or virtual toolkits are incapable of producing user interfaces adapted to different platforms at the same time, especially if the platforms are very different.

In contrast, the core component of graceful degradation is adaptation to the target platform, in order to preserve the UI's usability.

These theoretical assumptions were also tested empirically (section 8.2.7.a).

Low	Medium	High
Virtual toolkits Generic clients Transcoding Abstraction-reification		Traditional development Multireification approach (high level spec) Multireification approach (low level spec) Graceful degradation

Table 8-6 Compared usability of UIs targeted to very distinct platforms

8.1.5 Cross-platform consistency

In section 2.4.5, cross-platform consistency was defined as the capability to provide similar functionalities, similar operation procedures, similar data representations and the same data sets in each platform-specific version of the UI.

All model-based forward-engineering approaches ensure some form of consistency between the early phases of the development cycle (requirements analysis, specification) and the final product. In a multiplatform context, these approaches also guarantee consistency between the UI generated for different target platforms, with the exception of low-level generation, which requires a separate specification for each platform family and does not differ much from traditional techniques on that point, as shown on Table 8-7.

8. Validation

Consistency was one of our major concerns, and one important point that differentiates graceful degradation from low-level multireification, so we have tried to confirm this intuition on experimental data (section 8.2.7.b).

Low	Medium	High
Traditional development Multireification approach (low level spec)		Virtual toolkits Generic clients Transcoding Abstraction-reification Multireification approach (high level spec) Graceful degradation

Table 8-7 Compared cross-platform consistency between the UIs produced

8.1.6 Guidance

Guidance is a core component of graceful degradation: in contrast to other methods, where each new version of the user interface has to be redesigned from scratch; GD guides the transformation process between source and target UI by providing explicit transformation rules. For any change of platform feature or usability problem, the designer is oriented to possible actions. The consequences of each transformation rule are well documented, and additional information (advantages/disadvantages, exceptions, examples) is available on demand.

8.1.7 Conclusion

Let us now consider the position of graceful degradation in the summary table below.

Criteria	Development costs	Modification of functionalities	Modification of formats	Completeness	Level of control	Usability	Cross-platform consistency	Guidance
Approach								
Traditional development								
Virtual toolkits								-
Generic clients								-
Transcoding								-
Multireification approach (high level spec)								-
Multireification approach (lower level spec)								
Abstraction-reification								-
Graceful degradation								

Legend

= good for that criterion

= medium for that criterion

= bad for that criterion

-

 = irrelevant

Table 8-8 Global comparison of graceful degradation and other approaches on all criteria

Graceful degradation appears as a trade-off approach to the development of user interfaces for multiple, different platforms.

In terms of development and maintenance costs, our methodology is a middle way between the approaches that require only one single code or specification for all platform-specific versions of the UI and the approaches that require one code per target platform (traditional development) or one specification per platform family (low-level multireification).

In terms of completeness, although we do not expect to be able to produce any kind of complex user interfaces using GD rules only, graceful degradation performs better than other model-based approaches, because the source CUI is entirely specified by a human designer, instead of being built automatically from higher level descriptions, and because most of the aesthetical and practical design options taken by the designer when building the source UI can be preserved by the transformation process.

For the same reason, the usability of the user interfaces produced using GD rules is acceptable. On that criterion again, graceful degradation is a good trade-off between solutions relying on a unique code or specification and ad-hoc solutions such as traditional development, that permit to achieve the highest level of usability.

Graceful degradation also guarantees a reasonable degree of cross-platform consistency: the core components of the source UI are kept on all target UIs, even if the application of the transformation rules have an impact on the similarity between versions.

Guidance, which is a dimension largely neglected by the other development approaches, is also one strong point of our methodology.

8.2 Empirical validation

Empirical validation of our method and rules is realized by

- (1) Demonstrating the possibility of adapting an existing user interface to a more constrained target using the graceful degradation approach, by applying the methodology on case studies, both in the case of manual and semi-automatic development (Chapter 7).
- (2) Analysing the user's appreciation of the user interfaces produced by graceful degradation, in contrast to other methods (ad-hoc development and direct migration).

The last point is the subject of this section. The experiment described hereafter was carried out by two students during their master thesis [Henr04], conducted under our supervision.

8.2.1 Goals of the experiment

The study had two main goals:

- (1) To measure the usability of the user interfaces produced with the graceful degradation approach.
- (2) To investigate the cross-platform consistency of these user interfaces.

8.2.2 Experimental UIs

The GD approach consists in transforming a source UI into a target UI adapted to a more constrained platform. Therefore, the experiment has been conducted on UIs conceived for a desktop, and later adapted to a PDA.

We have selected two different source user interfaces from two Web sites:

- (1) An informative Web site, comprising only text and images: Iacchos¹⁴, a Web site for wine lovers (Figure 8-1).
- (2) An interactive Web site, comprising form controls by which the user interacts with the system: Maporama¹⁵, an on-line map service providing free maps and driving directions (Figure 8-2).



Figure 8-1 Screenshot of the Iacchos Web site: first source user interface of the experiment

¹⁴ <http://www.iacchos.com> (May 2004)

¹⁵ <http://www.maporama.com> (May 2004)

The screenshot shows the Maporama website interface. At the top is the logo 'maporama The World My Way' and a navigation bar with links: Société, Solutions, Produits, Services, Technologie, Clients, Communiqués. Below this is a language selector set to 'Français' and a 'GO' button. A 'Ajouter aux favoris' button is also present. The main content area is divided into several sections: 'Partner Hub 2003' with links 'Découvrir le Produit' and 'Connexion client'; 'Cartes' with a search form for 'Adresse' (Pays: France, Adresse, gare, monument, Code postal, Ville, Type d'affichage: Normal); 'Itinéraires' with a search form for 'Départ' (Pays: France, Adresse, gare, monument, Code postal, Ville: (obligatoire)) and 'Arrivée' (Pays: France, Adresse, gare, monument, Code postal, Ville: (obligatoire)); 'My Maporama' with a login form (Email, Mot de passe) and a 'GO' button; and 'Maporama Mobile' with links to 'Testez gratuitement Maporama Mobile avec AvantGo, FAQ' and 'Download Maporama Palm Application'. Each search form has a 'GO' button.

Figure 8-2 Screenshot of the Maporama Web site: second source user interface of the experiment

For both source user interfaces, 4 distinct target user interfaces have been considered:

- (1) A first version without modification (direct migration: the HTML code of the source interface was directly interpreted by the PDA's browser): version A.
- (2) A version resulting from the manual application of GD rules belonging to the Concrete User Interface level (layout modification rules): version B.
- (3) A version resulting from the manual application of a splitting rule (Abstract User Interface level): version C. On the Iacchos Web site, the navigation created between the interface's fragments was sequential (Figure 8-3) while, on the Maporama Web site, we provided a fully-connected navigation, thanks to a tabbed panel (Figure 8-4).
- (4) The last version (version D) was an "independent" version, i.e. an ad-hoc version, where the user interface has been totally redesigned in order to fit the target platform.

Versions B and C were produced by the two students together. Versions D were downloaded from the AvantGo Company Web site¹⁶ and are thus real-life examples of UIs manually adapted for use on handheld devices. Both the particular characteristics of the designers of the PDA UIs and the fact that B/C versions were not produced by the same designers as D versions may constitute a bias in our experiment (even if predicting the potential effects of these biases seems difficult).

During the experiment, the interviewers referred to these versions by the single letter identifier A, B, C and D, so that the interviewees did not have other information on these versions (they did not

¹⁶ <http://www.avantgo.com>, May 2006.

8. Validation

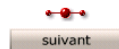
know, for example, that one version had been designed by a commercial company while two other versions had been produced by the students who conducted the experiment).

2^{èmes} Masters iacchos du Vin



Vous êtes un amateur passionné par les grands vins de France et d'ailleurs ? En formant une équipe de 4 amateurs entre amis ou entre collègues de votre club de dégustation, participez aux « 2^{èmes} Masters iacchos du Vin », le concours amateur international de dégustation et vivez un week-end passionnant, sympathique et convivial dans l'univers des grands vins !

Rendez-vous le **20 mars 2004** en région bordelaise !



• Qui organise ?

Ce concours est entièrement conçu, organisé et animé par des amateurs passionnés :

- **Vincent Mercier**, membre actif du Club de dégustation « Tire-Bouchon Attitudes » (Association Loi 1901 - Bordeaux), créateur du concept et grand animateur de cette manifestation,
- le **site Internet** iacchos.com des amateurs de grands vins, qui apporte tout son soutien et la promotion de la manifestation auprès de la communauté des œnophiles.

Il s'agit de la deuxième édition. Ce concours amateur a pour but de rassembler des passionnés dans un esprit « sportif ». Il n'a aucun but lucratif. Tous les vins sont achetés par les organisateurs.

Pour en savoir plus sur la première édition des Masters iacchos de mars 2002 (les vins, le QCM, les photos et les résultats...), [cliquez ici](#) !

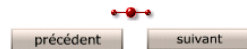


Figure 8-3 “Splitted” version of the Iacchos Web site, with sequential navigation

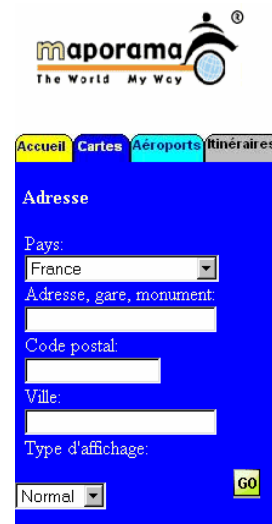


Figure 8-4 “Splitted” version of the Maporama Web site, with fully-connected navigation

8. Validation

8.2.3 Participants

The participants were selected by quota sampling: 12 respondents were chosen, each belonging to a category determined by the values of three variables: the computer literacy, the sex and the age range, as shown on Table 8-9.

Due to the non-random nature of the sample and to its limited size (forced by time constraints: the experiment required more than one hour per participant), we did not expect to obtain statistically significant results, but to identify trends and to get a global evaluation of the approach and of the rules.

Variable 1 : Computer literacy	Variable 2 : Age	Variable 3 : Sex	Nr subject
Good	Less than 26	Woman	Subject 1
		Man	Subject 2
	Between 26 and 55	Woman	Subject 3
		Man	Subject 4
	More than 55	Woman	Subject 5
		Man	Subject 6
Low	Less than 26	Woman	Subject 7
		Man	Subject 8
	Between 26 and 55	Woman	Subject 9
		Man	Subject 10
	More than 55	Woman	Subject 11
		Man	Subject 12

Table 8-9 The 12 categories of subjects recruited for the experience

8.2.4 Tasks

One task per Web site was devised:

- On the informative site (Iacchos), users were asked to find two pieces of information: a price and a person's name (price and name had been modified on each version in order to make the task less repetitive).
- On the interactive site (Maporama), the task consisted in navigating the Web pages, finding a form and filling it.

We chose these simple tasks in order to be able to carry out the experiment within a reasonable time. Increasing the complexity of the tasks (for example, by proposing a comparison task) would

probably have revealed more differences between the versions. This effect of the task complexity on experiments with small screen devices has been observed in other contexts [Chae04].

8.2.5 Questionnaires

Two main series of questionnaires were developed. The first series consisted of 8 questionnaires administered after each test on a distinct PDA version. These questionnaires were designed to assess, for each UI version:

- The user's satisfaction.
- The perceived ease of use.
- The perceived speed of use.
- The perceived clarity of information presentation.

Five response options, ranging from "5-very high" until "1-very low" were possible for each item. Furthermore, the participants were given the opportunity to provide free text comments on the positive / negative aspects of each version.

The second series included two comparison questionnaires administered after completing the entire set of tests linked to a given Web site, where users were asked to rank the 4 PDA versions. One ranking had to be provided for each of the following criteria: user's preference, aesthetics, perceived similarity with the desktop version in terms of functionalities and perceived similarity with the desktop version in terms of presentation.

8.2.6 Experimental procedure

The session began with a brief explanation of the purpose of the study, and a general introduction to the manipulation of the PDA (use of key buttons, of the virtual keyboard, of the stylus...) by the student playing the role of the instructor. Indeed, none of the participants was an actual PDA user, which is a limitation of our study. In order to avoid an unconscious manipulation of the tests by the instructor (a possible tendency to show her own versions in a more favourable light), all the interventions had been drawn up beforehand and were merely read during the experiment.

The participant was then given a task to perform on the first Web site (Iacchos). He/she began with the desktop version, and then successively carried out the same task on each PDA version. The target versions were presented in random order, distinct for each participant, in order to minimize the bias linked to the learning effect. A questionnaire of the first series was administered at the end of each task. Each task completion time was recorded by the second student, who acted as an observer and also took note of any event of interest that may occur (user comment, problem encountered...) A comparison questionnaire had to be filled after the tests related to the first Web site.

The same modus operandi was observed for the tests related to the second Web site (Maporama) and a global summary questionnaire was then proposed. The order between the two Web sites was kept constant (Iacchos was always before Maporama); because the participants were all PDA beginners and that the task to be carried out on the Maporama Web site was slightly more difficult.

8. Validation

8.2.7 Results

8.2.7.a Evaluation of the usability

The ISO 9241 standard defines usability 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” [ISO98]. Effectiveness is measured by evaluating the quality of the results that the user generates with the system; efficiency is assessed by quantitative measurements, such as the duration or the error count and satisfaction is evaluated through questionnaires.

The tasks the users had to complete during our experiment were not complex enough to measure differences between user interface versions in terms of effectiveness. Efficiency was evaluated through the measure of task completion time and user satisfaction was assessed through the subjective ranking of the approaches along several quality criteria (satisfaction, ease of use, perceived speed of use and clarity).

Table 8-10 and Table 6-1 summarize the answers to the first series of questions (on a 5-point scale, 1 being the lowest score and 5 the highest), while the charts on Figure 8-5 present the global ranking of each version, drawn from the two comparison questionnaires.

		<u>lacchos A</u>	<u>lacchos B</u>	<u>lacchos C</u>	<u>lacchos D</u>
		Direct migration	GD: layout modification rule	GD: splitting with sequential navigation	Ad-hoc development
satisfaction	mean	1.8	3.8	3.2	4
	median	1	4	3	4
ease of use	mean	1.8	4	3.2	4.3
	median	1	4	3	4
speed of use	mean	2.1	4.3	3	4.1
	median	2	4	3	4
clarity	mean	1.8	3.8	3.4	4.4
	median	1	4	3,5	4
global appreciation		1.83	3.96	3.19	4.19
time	mean	1:52	1:17	2:08	0:41
	median	1:42	0:59	1:36	0:35

Table 8-10 Evaluation of the four PDA versions of the Iacchos Web site

8. Validation

		Maporama A	Maporama B	Maporama C	Maporama D
		Direct migration	GD: layout modification rule	GD: splitting with tabbed panel	Ad-hoc development
satisfaction	mean	1.5	3.5	3.9	3.7
	median	1	3	4	4
ease of use	mean	1.8	3.4	3.8	3.8
	median	2	3.5	4	4
speed of use	mean	1.5	3.3	3.8	3.8
	median	1	3	4	4
clarity	mean	1.5	3.3	4.2	3.8
	median	1	3.5	4	4
global appreciation		1.56	3.37	3.94	3.75
time	mean	3:28	2:10	2:04	2:29
	median	2:53	1:48	1:47	1:46

Table 8-11 Evaluation of the four PDA versions of the Maporama Web site

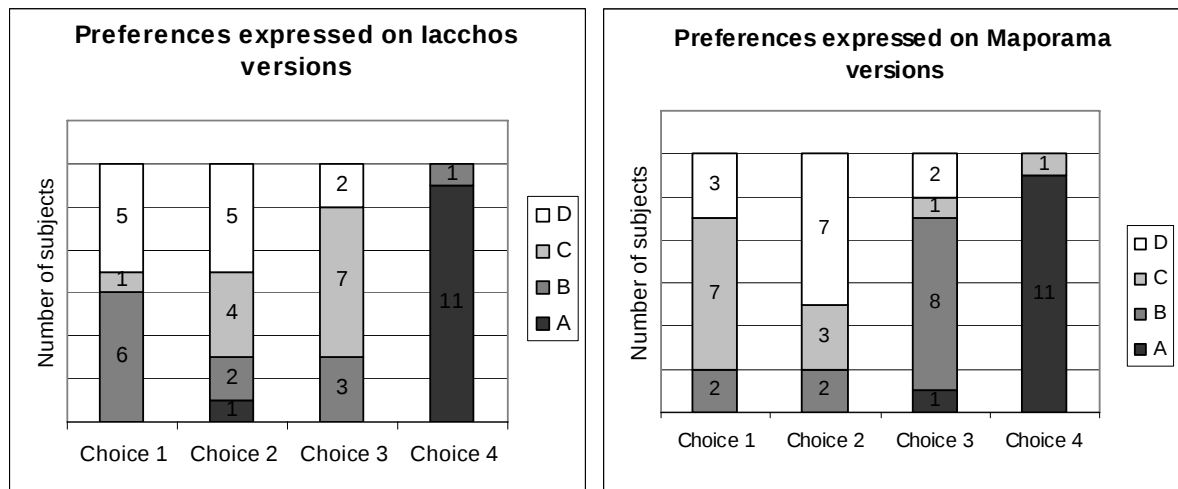


Figure 8-5 Preferences expressed on Iacchos / Maporama versions

The primary goal of the experiment was to compare the usability of the user interfaces produced by graceful degradation (versions B and C), in contrast to ad-hoc development (versions D) and direct migration (versions A).

Our hypothesis was a ranking $D > C > B > A$, the D versions being the most appreciated and the most usable, and the A versions the least appreciated and usable, with a middle position for the interfaces produced with the GD approach.

8. Validation

We analysed the global appreciation and task completion time of the four UI versions using the nonparametric Wilcoxon Signed-Ranked test, in order to check whether the apparent differences presented in Table 8-10 and Table 8-11 were significant or not. The results of these tests are presented in Table 8-12 and Table 8-13. We found significant differences (with p-values < .05) for all Iacchos versions. For the Maporama UIs, we were only able to demonstrate the difference between the A versions and the other versions (i.e. for this particular set of UIs, we were able to prove that the PDA UIs produced by GD were more appreciated and required less task completion time than the PDA UI produced by direct migration, but we were not able to found significant differences between the UIs produced by GD and the ad-hoc UI).

	Iacchos A / Iacchos B	Iacchos B / Iacchos C	Iacchos C / Iacchos D	
Global score	0.001	0.014	0.000	
	Maporama A / Maporama B	Maporama B / Maporama C	Maporama C / Maporama D	Maporama B / Maporama D
Global score	0.000	0.059	0.344	0.107

Table 8-12 p-values for Wilcoxon Signed-Ranked test (H0: Global score on perceived usability is equal)

	Iacchos A / Iacchos B	Iacchos B / Iacchos C	Iacchos C / Iacchos D	
Time	0.009	0.017	0.001	
	Maporama A / Maporama B	Maporama B / Maporama C	Maporama C / Maporama D	Maporama B / Maporama D
Time	0.002	0.606	0.027	0.209

Table 8-13 p-values for Wilcoxon Signed-Ranked test (H0 : Global score on task completion time is equal)

In summary, experimental results do not contradict our ranking hypothesis (D > C > B > A):

- The A versions (HTML code directly used on the PDA) were the least appreciated, on both Web sites, obtained the lowest score for all criteria and required the highest task completion times. The difference between the A versions and the other versions were found statistically significant, for both source UIs and for all criteria.
- The D versions (traditional, ad-hoc development) were generally highly appreciated, were classified in first or second position by a large majority of users, obtained a high global mean for the four quality criteria assessed and required the lowest task completion times (even if the differences could not be proved to be significant for the Maporama UI).
- The B versions (GD rules at the Concrete User Interface level) are effectively positioned between D and A versions on both Web sites in terms of task completion time, of global mean for the four quality criteria, and for nearly each individual criterion.
- The classification of the C versions seems to be linked with the navigation style created between the interaction spaces on the target platform:

- (1) When the navigation is sequential (Iacchos), the C version gives lower scores than the B version: lower global mean, lower score for all quality criteria, higher task completion time.
- (2) On the other side, when the navigation is fully-connected (Maporama), the C version performs much better: it is the first choice version for a majority of participants, the most appreciated for all quality criteria and one of the best in terms of efficiency. Most of the time, the C version of the Maporama Web site was even more appreciated than the D version (ad-hoc development).

User's perception of the different navigation types is confirmed by their comments: when questioned on sequential navigation, ten people out of twelve gave a negative appreciation. In contrast, navigation inside a tabbed panel received only one negative comment and one moderate.

Note also that efficiency and user appreciation are not always correlated. So, on Table 8-10, Iacchos C is the least efficient version (highest mean task completion time), but is not the least appreciated. Similar observations may be found in a study of the performance, precision and user appreciation of widgets for various tasks [John95], where the authors found that the most efficient widget was not necessarily the most appreciated.

The present data suggests that our theoretical estimation of the usability of the UIs produced with our methodology (section 8.1.4) was correct but, again, these experimental results should be confirmed by tests conducted at a bigger scale. In particular, the independent variables should be refined to take into account variations of programming techniques (for the A versions), precise type of GD rule used (versions B and C) and variations between design teams (versions D). The number of different source UIs should also be increased and more complex tasks should be proposed to the participants.

8.2.7.b Evaluation of the cross-platform consistency

One theoretical argument in favour of graceful degradation was that the approach provided consistency between the platform specific versions. This aspect is important, because the usability of a multiplatform system depends not only on the usability of each platform specific version but also on the transitions between these versions. While [Deni03] proposes to assess cross-platform consistency by analyzing each possible transition between versions, we will only consider the transitions from the source UI to the target UIs. This approach is less complete than the first one, but the focus is set on the most important transitions: those between the user's most familiar environment (the desktop) and the other environments (here: the PDA), where we would like the user to reuse his/her knowledge of the first system.

Two measures were meant to assess the cross-platform consistency:

- The rating of perceived similarity with the desktop version in terms of functionalities (results on Figure 8-6).
- The rating of perceived similarity with the desktop version in terms of presentation (results on Figure 8-7).

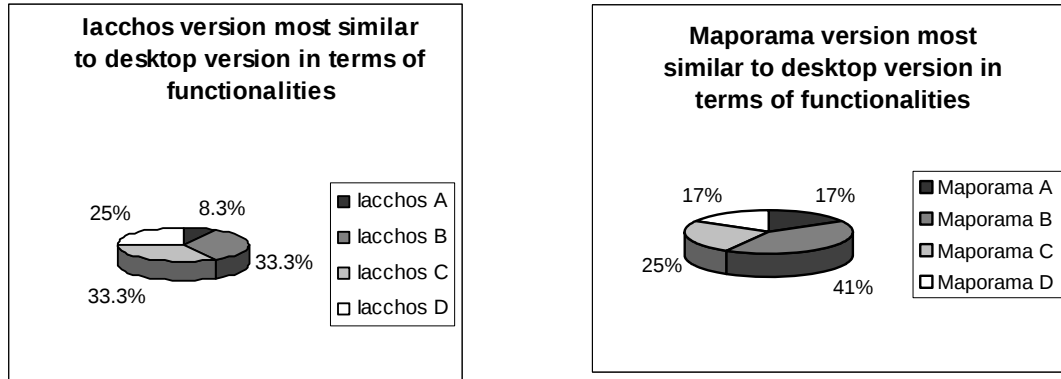


Figure 8-6 Perceived similarity of the PDA versions with the desktop version in terms of functionalities

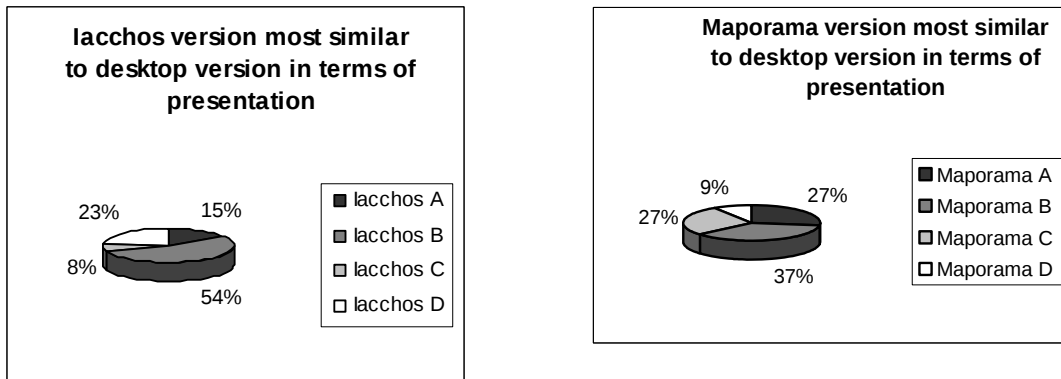


Figure 8-7 Perceived similarity of the PDA versions with the desktop version in terms of presentation

The order expected, from the most similar version until the most different was $A < B < C < D$. Surprisingly, few participants recognized that the A versions were identical to their source desktop version: A versions are considered as the least similar to the source interface in terms of functionalities both on Iacchos and Maporama, and they are only classified second and third in terms of similarity of presentation. This non-recognition was confirmed by oral comments made during the experiment and is probably due to the large amount of scrolling required or to some presentation features not satisfyingly rendered on the PDA.

The collected data was insufficient to be exploited statistically but, in average, the expected order between the other versions (i.e. $B < C < D$) is not contradictory with the experimental data because: (1) it is confirmed in three rankings out of four, even is the perceived similarity of the Iacchos D version is higher than the C version, (2) the average ranking for the functionalities similarity is $B < C < D < A$ and the average ranking for the presentation similarity is $B < A < C < D$.

These results seem to confirm that the GD approach leads to more cross-platform consistency than ad-hoc development, and that GD rules applied at the CUI level (B versions) have less impact on the cross-platform consistency than GD rules applied at the AUI level (C versions). Furthermore, they show that relying on generic clients for interpreting the same code does not

always provide user interfaces that are perceived as similar, which forces us to moderate the theoretical ranking presented in Table 8-7.

8.2.8 Conclusion

The experiment described above was meant to confront the subjective appreciation given on the impact of graceful degradation on usability (8.1.4) and cross-platform consistency (8.1.5) with factual data. The analysis of the results collected did not reveal evidence against our hypothesis that graceful degradation should be considered as a trade-off approach between direct migration and ad-hoc development, in terms of the two criteria studied. However, the study also suggests that parameters such as the exact type of GD rule (type of dialog between the splitted fragments of the UI for example), or the chosen source UI may have an impact on the users' appreciation of the UI. Our experimental set-up did not permit studying these potential effects. Another limitation of the experiment is that we did not meet the ideal condition of having all PDA UIs produced by independent designers.

Chapter 9 Conclusion

9.1 Summary of results

9.1.1 Theoretical and conceptual contributions

9.1.1.a Platform and interactor models

Chapter 3 proposed a platform model (3.6) and an interactor model (3.7) for the UsiXML user interface language. We observed a shortage of platform models in model-based user interface management systems. Therefore, we established a detailed list of hardware and software properties likely to influence the construction of user interfaces. An overview of this platform model can be found in [Limb04]. The set of widgets available on a given platform is also an important characteristic of this platform. For this reason, we developed an interactor model aimed to enhance the description of final interactors found in toolkits or mark-up languages with higher level information about their functionalities. Chapter 4 demonstrates how this interactor model supports interactor substitution rules, when a given widget is unsuitable or unavailable on a target platform (4.2.2).

9.1.1.b Catalogue of rules

In the first chapter of this thesis, we listed a series of problems faced in creating multiplatform user interfaces. The lack of knowledge and experience in the field was identified as an important limitation. Chapter 4 addressed this problem by presenting a structured collection of transformation rules that can be used both manually and semi-automatically by designers (4.1). This typology of rules has been published in [Flor04b]. Formalization and structuring of these rules have been addressed in Chapter 4 (4.2) and Chapter 6 (6.1), respectively.

9.1.1.c Splitting algorithm

Chapter 5 explored the splitting rule. Although the only input required to apply the splitting rule is a source interface at the CUI level, an algorithm was proposed that exploits the AUI and task models to guide the splitting process when these models are available. The novelty of this algorithm resides, on one hand, in the use of the temporal relationships between tasks in a splitting process and, on the other hand, in the scalability of the multi-level approach, which takes advantage of the possibility to build a UI specification using any combination of model components in UsiXML. A description of this algorithm has been published in [Flor06] and, more extensively, in [Flor06b].

9.1.2 Methodological contribution

The introduction of graceful degradation as a design approach represents a methodological contribution to the field of multiplatform user interface design, which is an answer to the lack of methods in the field of MUI identified at the begin of this thesis. Chapter 7 demonstrated the practicability of the method. Chapter 8 showed the advantages of GD compared to other approaches for developing of MUI in terms of costs, applicability domain, usability of the user interfaces produced, designer control, cross-platform consistency and guidance. GD thus also addresses concerns expressed on the high development and maintenance costs of MUI and on the lack of consistency and usability of multiplatform systems.

9.1.3 Tools developed

Two tools were built in support to the GD method. The first tool (6.1) offers an interface to a database that gathers, structures and organizes GD rules. To the best of our knowledge, this is the first tool aimed to manage knowledge bases of adaptation rules. The second tool (6.2) demonstrates the viability of GD when applied semi-automatically on a UI specification. In contrast to other model-based environments for building MUI, this tool mostly relies on “horizontal” transformations (translations in the CAMELEON terminology) instead of “vertical” transformations (i.e; reification and/or abstraction). The tool provides access to the description of the rules in the knowledge base (6.2.2). Industry professionals interviewed during a small user study (6.2.5) expressed their interest towards this kind of tool.

9.2 Future work in prospect

In the immediate future, possible research opportunities include:

- **Applying the method and rules to user interfaces in other formats.** Until now, GD rules have been described in terms of the CAMELEON framework and applied semi-automatically on source user interfaces in UsiXML. Applying GD concepts to user interfaces in formats such as UIML or XUL for example would demonstrate the transferability of the approach. The availability of stable and effective rendering engines for these languages would constitute an additional advantage.
- **Applying the method and rules to more complex applications.** In this thesis, we have selected two case studies to show the applicability of our approach, both in a scenario where the rules are used manually by a human developer and in a scenario where the application of the rules is supported by a tool. Investigating more complex and more diverse applications would be necessary to demonstrate the scalability of the approach.
- **Exploring new architectures and techniques for the GD tool.** The prototype built in the context on this thesis was only meant to prove the feasibility of designing user interfaces by applying GD rules semi-automatically. Further development of the tool should provide the designer with possibilities to add or modify GD rules, and to export these rules to other environments. The possibility to save a subset of rules and their parameters as a kind of “transformation scenario” that could be applied consistently on any interaction space from the

source user interface, or on another source interface, should also be considered: it would provide a real advantage in terms of reusability.

- **Exploring GD rules at the dialog level.** This thesis has been focused on presentational aspects. Future work should give more attention to aspects such as dialog adaptation, impact of existing GD rules on the dialog, representation of the dialog in UsiXML or other user interface description languages.
- **Extending the corpus of rules in other directions:**
 - **Extending the corpus of rules with upgradation rules.** Upgradation rules would permit to deal with the fact that a lesser restricted device, in comparison to a chosen source device, could occur during the lifetime of the system. A basic corpus of upgradation rules could be built by inverting the current GD rules.
 - **Extending the corpus of rules with platform-specific rules.** The current corpus is composed of rules that apply to every target platform. Gathering rules linked to characteristics and usability rules specific to a given platform would increase the utility of the tools, since designers often lack experience and knowledge about mobile platforms, or are specialized in a small number of platforms only.
- **Studying work practices of multiplatform designers.** This thesis has only tackled this aspect superficially (6.2.5.b), but a larger-scale investigation of current design practices would help us to understand which development processes must be supported by our tools.

Considering possible extensions at the long term, we could think about:

- **Applying the GD rules in different adaptation configurations:**
 - **System initiative:** automatic recognition of usability problems and/or detection of changes in the platform (e.g. smaller screen size). Existing works on automatic evaluation of usability should be a starting point, but additional difficulties should be taken into account: the evaluation should be carried on an abstract description of the user interface, and not at the code level, and the evaluation should take the target platform's characteristics into account.
 - **System proposal:** automatic proposition of GD rules when a problem is recognized. The structure of the GD rules as an aggregation of a condition and a related reaction is already a first step in this direction. More advanced metamodels of evolution rules (for example, [Gann05]) should also be considered.
 - **System decision:** automatic selection of a suitable GD rule among the proposals. In [Flor04b], we proposed a priority ordering of GD rules based on their impact on the continuity of the multiplatform system. This theoretical ranking should be refined by user studies such as the experiment described in section 8.2. Other selection criteria should also be taken into consideration, such as for example user preferences, or expected impact on

the usability of the target user interface, which would also requires investigation on the automatic evaluation of the quality of a user interface design, the definition of metrics, or the extension of the interactor model to include platform-specific information on the computational, physical and cognitive costs of each interactor.

- **System learning:** automatic acquisition of transformation scenarios based on the analysis of the interactions of human designers with the plug-in.

- **Applying the GD rules with different goals:**

- **UI personalization / customization** after deployment, either under direct user control, either driven by the computer and based on a user model (containing information on the user's role, experience, access to data, frequency of tasks carried on,...)
- **Improving the accessibility for users with disabilities:** for example, resizing rules, possibly coupled with splitting rules, could be useful for low-vision users...

9.3 Concluding remarks

We started this doctoral research with one research field: model-based development techniques for multiplatform user interfaces and one real-world case study: the ARTHUR system. At this time, most of the model-based solutions specifically adapted to the problem of multiplatform UIs were focused on automatically deriving UI code from abstract models (what we called “multireification”) and on defining the User Interface Description Languages necessary to specify those models.

Our experience with the ARTHUR prototype has led us to question the current model-based engineering methods. Classically, the development cycle of the ARTHUR software began with a stage of requirement elicitation and analysis, where several artefacts were produced: task models and UML use cases and class diagrams. Our first intention, which was to explore transformation rules able to produce platform specific user interfaces starting from these abstract models, quickly appeared unrealistic, due to the complexity of the envisioned system and the high expectations of emergency health professionals in terms of system usability.

This complexity, together with the difficulty to collect, understand and formalize user requirements, forced the ARTHUR development team to adopt a fast prototyping approach, where mock-ups were iteratively used to collect user feedback, which permitted obtaining more precise specifications and to build new mock-ups, more consistent with users' expectations.

Difficulty of transforming high level specifications in usable user interfaces fully automatically and confrontation with actual practices, where first versions of the user interfaces are produced before the end of the requirements elicitation stage were the initial motivations for the work described in this thesis. Our work has been neatly focused on the design stage with little attention to run-time and on presentational aspects rather than dialog, so the list of potential future work is still huge. We hope that further research on these fields may benefit from our attempt to clarify the problem space.

References

References

A

[Ali03]

Ali, M. F., A.Pérez-Quñones, M., Abrams, M., & Shell, E. (2004). Building Multi-Platform User Interfaces with UIML. In H. J. a. A. Seffah (Ed.), *Multiple User Interfaces: Engineering and Application Framework* (pp. 95-118). Chichester (GB): Wiley and Sons.

[Alle83]

Allen, J. F. (1983). Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11), 832-843.

[Amou05]

Ammouh, T., Gemo, M., Macq, B., Vanderdonckt, J., Gariani, A. W. E., Reynaert, M., et al. (2005). Versatile clinical information system design for emergency departments. *IEEE Transactions on Information Technology in Biomedicine*, 9(2), 174-183.

[Arta05]

Artail, H.A., & Raydan, M. (2005). Device-aware desktop web page transformation for rendering on handhelds. *Personal and Ubiquitous Computing*, 9(6), 368-380.

B

[Bana04]

Banavar, G., Bergman, L. D., Gaeremynck, Y., Soroker, D., & Sussman, J. (2004). Tooling and system support for authoring multi-device applications. *The Journal of Systems and Software*, 69(3), 227-242.

[Bart88]

Barthet, M.-F. (1988). Logiciels interactifs et ergonomie. Paris: Dunod Informatique.

[Bick97]

Bickmore, T. W., & Schilit, B. N. (1997). Digestor: Device-independent access to the World Wide Web. *Computer Networks and ISDN Systems*, 29(8-13), 1075-1082.

[Boda95]

Bodart, F., Hennebert, A. , Lheureux, J. , Provot, I. , Sacré, B. , & Vanderdonckt, J. (1995). *Towards a systematic building of software architecture: The TRIDENT methodological guide*. Proceedings of 1st Eurographics Workshop on Design, Specification, Verification of Interactive Systems DSV-IS'95 (7-9 June, Château de Bonas, France).

References

[Boui02]

Bouillon, L., Vanderdonckt, J., & Chow, K. C. (2004). *Flexible re-engineering of web sites*. Proceedings of the 9th International Conference on Intelligent User Interfaces IUI'04 (13-16 January, Funchal, Portugal).

[Boui04]

Bouillon, L., Vanderdonckt, J., & Eisenstein, J. (2002). *Model-Based Approaches to Reengineering Web Pages*. Proceedings of the 1st International Workshop on Task Models and Diagrams for User Interface Design TAMODIA '02 (18 -19 July, Bucharest).

[Brau04]

Braun, E., Hartl, A., Kangasharju, J. & Mühlhäuser, M. (2004). *Single Authoring for Multi-Device Interfaces*. Proceedings of the 8th ERCIM Workshop "User Interfaces For All" (28-29 June, Vienna). Retrieved March, 2006, from http://ui4all.ics.forth.gr/workshop2004/files/ui4all_proceedings/adjunct/techniques_devices_metaphors/45.pdf

[Butl01]

Butler, M. (2001). Current Technologies for Device Independence. *HP Technical Report HPL-2001-83*, Retrieved January, 2006, from <http://www.hpl.hp.com/techreports/2001/HPL-2001-83.html>

[Butl02]

Butler, M. (2002). Using Capability Profiles For Appliance Aggregation. *HP Technical Report HPL-2002-173R1* Retrieved January, 2006, from <http://www.hpl.hp.com/techreports/2002/HPL-2002-173R1.html>

[Buyu02]

Buyukkokten, O., Kaljuvee, O., Garcia-Molina, H., Paepcke, A., & Winograd, T. (2002). Efficient web browsing on handheld devices using page and form summarization. *ACM Transactions on Information Systems (TOIS)*, 20(1), 82-115.

C

[Caet02]

Caetano, A., Goulart, N., Fonseca, M., & Jorge, J. (2002). JavaSketchIt: Issues in Sketching the Look of User Interfaces. Proceedings of the 2002 AAAI Spring Symposium - Sketch Understanding (25-27 March, Palo Alto, United States).

[Calv03]

Calvary, G., Coutaz, J., Daassi, O., Balme, L., & Demeure, A. (2004). *Towards a new generation of widgets for supporting software plasticity: the 'comet'*. Proceedings of the 9th IFIP Working Conference on Engineering for Human-Computer Interaction, Jointly with The 11th International Workshop on Design,

References

- Specification and Verification of Interactive Systems EHCI-DSVIS'04 (11-13 July, Hamburg, Germany).
- [Calv04]
- Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., & Vanderdonckt, J. (2003). A unifying reference framework for multi-target user interfaces. *Interacting with Computers* 15(3), 289–308.
- [Calv05]
- Calvary, G., Daassi, O., Coutaz, J., & Demeure, A. (2005). Des widgets aux comets pour la Plasticité des Systèmes Interactifs. *Revue d'Interaction Homme-Machine*, 6(1), 33-53.
- [Ceri00]
- Ceri, S., Fraternali, P., & Bongio, A. (2000). *Web Modeling Language (WebML): a Modeling Language for Designing Web Sites*. Proceedings of the 9th World Wide Web Conference WWW9 (15-19 May, Amsterdam).
- [Chae04]
- Chae, MW., & Kim, JW. (2004). Do size and structure matter to mobile users? An empirical study of the effects of screen size, information structure, and task complexity on user activities with standard web phones. *Behaviour & Information Technology*, 23(3), 165-181.
- [Chen05]
- Chen, Y., Xie, X., Ma, W.-Y., & Zhang, H.-J.(2005). Adapting Web Pages for Small-Screen Devices. *IEEE Internet Computing*, 09(1), 50-56.
- [Ches04]
- Chesta, C., Paternò, F., & Santoro, C. (2004). Methods and Tools for Designing and Developing Usable Multi-Platform Interactive Applications. *PsychNology Journal*, 2, 123-139. Retrieved January 2006 from http://www.psychnology.org/File/PSYCHNOLOGY_JOURNAL_2_1_CHESTA.pdf
- [Chu04]
- Chu, H., Song, H., Wong, C., Kurakake, S., & Katagiri, M. (2004). Roam, a seamless application framework. *Journal of Systems and Software*, 69(3), 209-226.
- [Clar00]
- Clark, D. (2000). *From Abstract to Concrete: designing AUIML renderers*. IBM White Paper, May 2000.
- [Coye04]
- Coyette, A., Faulkner, S., Kolp, M., Limbourg, Q., & Vanderdonkt, J. (2004). *SKetchiXML: Towards a Multi-Agent Design Tool for Sketching User Interfaces Based on USIXML*. Proceedings of the 4th International Workshop on Task Models and Diagrams for User Interface Design TAMODIA'04 (15-16 November, Prague, Czech Republic).

References

[Crea00]

Crease, M., Gray, P., & Brewster, S. (2000). *A Toolkit of Mechanism and Context Independent Widgets*. Proceedings of the Design, Specification and Verification of Interactive Systems, Workshop 8, ICSE 2000 (5-6 June, Limerick, Ireland).

D

[Dees04]

Dees, W. (2004). *Handling device diversity through multi-level stylesheets*. Proceedings of the 8th International Conference on Intelligent User Interfaces IUI'2004 (13-16 January, Funchal, Portugal).

[Deli98]

Delis, V., & Papadias, D. (1998). *Querying Multimedia Documents By Spatiotemporal Structure*. Proceedings of the Third International Conference on Flexible Query Answering Systems FQAS '98 (13-15 May, Roskilde, Denmark).

[Deni03]

Denis, C., & Karsenty, L. (2003). Inter-usability of multi-device systems: A conceptual framework. In A. Seffah & H. Javahery (Eds.), *Multiple User Interfaces: Engineering and Application Framework* (pp. 373-385). Chichester (GB): Wiley and Sons.

[Diet94]

Dieterich, H., Malinowski, U., Kühme, T., & Schneider-Hufschmidt, M. (1994). State of the Art in Adaptive User Interfaces. In M. Schneider-Hufschmidt, T. Kühme & U. Malinowski (Eds.), *Adaptive User Interfaces: Principles and Practice* (pp. 13-48). Amsterdam: North-Holland.

[Ding06]

Ding, Y., & Litz, H. (2006). *Creating Multiplatform User Interfaces by Annotation and Adaptation*. Proceedings of the 10th International Conference on Intelligent User Interfaces IUI'2006 (29 January - 1 February, Sydney).

[Drag05]

Dragicevic, P., Chatty, S., Thevenin, D., & Vinot, J.-L. (2005). *Artistic resizing: a technique for rich scale-sensitive vector graphics*. Proceedings of the 18th annual ACM symposium on User interface software and technology UIST '05 (23-26 October, Seattle, United States).

E

[Ehri99]

Ehrig, H., Engels, G., Kreowski, H.-J., and Rozenberg, G. (1999). *Handbook of Graph Grammars and Computing by Graph Transformation, Application, Languages and Tools*, Vol. 2. Singapore: World Scientific.

References

[Eise01]

Eisenstein, J., Vanderdonckt, J., & Puerta, A. (2001). *Applying model-based techniques to the development of UIs for mobile computers*. Proceedings of the 6th International Conference on Intelligent User Interfaces IUI'01 (14 -17 January, Santa Fe, United States).

F

[Flor04]

Florins, M., Trevisan, D., & Vanderdonckt, J. (2004). *The Continuity Property in Mixed Reality and Multiplatform Systems: a Comparative Study*. Proceedings of the 4th International Workshop on Computer-Aided Design of User Interfaces CADUI'04 (13-16 January, Funchal, Portugal).

[Flor04b]

Florins, M., & Vanderdonckt, J. (2004). *Graceful Degradation of User Interfaces as a Design Method for Multiplatform Systems*. Proceedings of the 8th International Conference on Intelligent User Interfaces IUI'04 (13-16 January, Funchal, Portugal).

[Flor06]

Florins, M., Montero, F., Vanderdonckt, J., & Michotte, B. (2006). *Splitting rules for graceful degradation of user interfaces*. Proceedings of the 11th International Conference on Intelligent User Interfaces IUI'06 (29 January 29 - 01 February, Sydney, Australia).

[Flor06b]

Florins, M., Montero, F., Vanderdonckt, J., & Michotte, B. (2006). *Splitting rules for graceful degradation of user interfaces*. Proceedings of the Advanced Visual Interfaces International Working Conference AVI 2006 (23-26 May, Venice).

[Fox98]

Fox, A., Goldberg, I., Gribble, S. D., & Lee, D. C. (1998). *Experience With Top Gun Wingman: A Proxy-Based Graphical Web Browser for the 3Com PalmPilot*. Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing Middleware'98 (15-18 September, Lake District, UK).

G

[Gajo04]

Gajos, K., & Weld, D. S. (2004). SUPPLE: Automatically Generating User Interfaces. Proceedings of the 8th International Conference on Intelligent User Interfaces IUI'2004 (January 13-16, Funchal, Portugal).

References

[Gann05]

Ganneau, V. (2005). *Plasticité des Systèmes Interactifs: Modèle d'Evolution et Apprentissage*. Unpublished Master thesis, Université Joseph Fourier, Grenoble.

[Ghez01]

Ghezzi, C., Jazayeri, M., & Mandrioli, D. (1991). *Fundamentals of Software Engineering*. Englewood Cliffs (United States): Prentice Hall.

[Göbe01]

Göbel, S., Buchholz, S., Ziegert, T., & Schill, A. (2001). *Device independent representation of web-based dialogs and contents*. Proceedings of the IEEE Youth Forum in Computer Science and Engineering YUFORIC'01 (29-30 November, Valencia, Spain).

[Gold93]

Goldberg, D., & Richardson, C. (1993). Touch typing with a stylus. Proceedings of the Conference on Human Factors in Computing Systems INTERCHI'93 (24-29 April, Amsterdam).

[Gram00]

Grammenos, D., Akoumianakis, D., & Stephanidis, C. (2000). Integrated Support for Working With Guidelines: The Sherlock Guidelines Management System. *Interacting with Computers*, 12(3), 281-311.

[Grif01]

Griffiths, T., Barclay, P. J., Paton, N. W., McKirdy, J., Kennedy, J., Gray, P. D., et al. (2001). Teallach: A Model-Based User Interface Development Environment for Object Databases. *Interacting with Computers*, 14(1), 31-68.

[Grol02]

Grolaux, D., Van Roy, P., & Vanderdonckt, J. (2002). *FlexClock: A Plastic Clock Written in Oz with the QtK Toolkit*. Proceedings of the 1st International Workshop on Task Models and Diagrams for User Interface Design TAMODIA '02 (18-19 July, Bucharest).

[Grun03]

Grundy, J., & Zou, W. (2003). AUIT: Adaptable User Interface Technology, with Extended Java Server Pages. In A. Seffah & H. Javahery (Eds.), *Multiple User Interfaces: Engineering and Application Framework* (pp. 149-167). Chichester (UK): Wiley and Sons.

H

[Hain02]

Hainaut, J. (2002). Introduction to Database Reverse Engineering. Retrieved January, 2006, from <http://www.info.fundp.ac.be/~dbm/publication/2002/DBRE-2002.pdf>

References

[Hare96]

Harel, D., & Naamad, A. (1996). The STATEMATE Semantics of Statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4), 293 – 333.

[Henn00]

Henninger, S. (2000). A Methodology and Tools for Applying Context-Specific Usability Guidelines to Interface Design. *Interacting with Computers*, 12(3), 225-243.

[Henr04]

Henry, C., & Henry, K. (2004). *Recherche sur les préférences des utilisateurs en ce qui concerne la dégradation des interfaces en vue d'être visionnées sur des plates-formes à petit écran*. Unpublished Master thesis, Université Catholique de Louvain, Louvain-la-Neuve.

I

[ISO98]

ISO. (1998). International Standard ISO 9241-11: Ergonomic requirements for office work with visual display terminals (VDTs) - Part 11: Guidance on usability.

J

[Jaba03]

Jabarin, B., & Graham, T.C.N. (2003). *Architectures for Widget-Level Plasticity*. Proceedings of the 10th International Workshop on Interactive Systems Design, Specification, and Verification DSV-IS 2003 (11-13 June, Funchal, Portugal).

[John95]

Johnsgard, T.J., Page, S.R., Wilson, R.D., & Zeno, R.J. (1995). *A Comparison of Graphical User Interface Widgets for Various Tasks*. Proceedings of the Human Factors & Ergonomics Society 39th Annual Meeting HFS'95 (9-13 October, Santa Monica, United States).

[John92]

Johnson, J. (1992). *Selectors: going beyond user-interface widgets*. Proceedings of the SIGCHI conference on Human factors in computing systems CHI '92 (03-07 May, Monterey, United States).

[John95]

Johnson, P., Johnson, H., & Wilson, S. (1995). Rapid prototyping of user interfaces driven by task models In J. M. Carroll (Ed.), *Scenario-Based Design: Envisioning Work and Technology in System Development*. New-York: John Wiley & Sons.

References

K

[Kaas01]

Kaasinen, E., Kolari, J., & Laakko, T. (2001). *Mobile-Transparent Access to Web Services*. Proceedings of the 8th IFIP TC.13 Conference on Human-Computer Interaction Interact 2001 (9-13 July, Tokyo).

[Kera02]

Keränen, H., & Plomp, J. (2002). *Adaptive Runtime Layout of Hierarchical UI Components*. Proceedings of the 2d Nordic Conference on Human-Computer interaction NordiCHI '02 (19 - 23 October, Åarhus, Denmark).

[Kim93]

Kim, W., & Foley, J. (1993). Providing High-Level Control and Expert Assistance in User Interface Presentation Design. Proceedings of the Conference on Human Factors in Computing Systems INTERCHI'93 (24-29 April, Amsterdam).

L

[Land95]

Landay, J. A., & Myers, B. A. (1995). *Interactive Sketching for the Early Stages of User Interface Design*. Proceedings of the Conference on Human Factors in Computing Systems CHI'95 (7-11 May, Denver, United States).

[Laud06]

Laudon, K., & Laudon, J. (2006). *Management information systems: managing the digital firm*. Upper Saddle River (United States): Pearson Prentice Hall.

[Leun00]

Leung, K. R. P. H., Hui, L. C. K., Yiu, S. M., & Tang, R. W. M. (2000). Modeling Web Navigation by Statechart. *HKU CSIS Technical Report* Retrieved January, 2006, from <http://www.cs.hku.hk/research/techreps/document/TR-2000-01.pdf>

[Lewi95]

Lewis, J. R. (1995). IBM Computer Usability Satisfaction Questionnaires: Psychometric Evaluation and Instructions for Use. *International Journal of Human-Computer Interaction*, 7(1), 57-78.

[Lin03]

Lin, J. (2003). *Damask: A Tool for Early-Stage Design and Prototyping of Cross-Device User Interfaces*. Proceedings of the Conference Supplement of UIST 2003: ACM Symposium on User Interface Software and Technology (2-5 November, Vancouver, Canada).

References

[Limb03]

Limbourg, Q., & Vanderdonckt, J. (2003). Comparing Task Models for User Interface Design. In D. Diaper & N. Stanton (Eds.), *The Handbook of Task Analysis for Human-Computer Interaction* (pp. 135-154). Mahwah (United States): Lawrence Erlbaum Associates.

[Limb04]

Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Florins, M., & Trevisan, D. (2004). *USIXML: A User Interface Description Language for Context-Sensitive User Interfaces*. Proceedings of 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 EHCI-DSVIS'04 (11-13 July, Hamburg, Germany).

[Limb04b]

Limbourg, Q. (2004). *Multi-Path Development of User Interfaces*. Unpublished PHD thesis, Université catholique de Louvain, Louvain-la-Neuve.

[Lin02]

Lin, J., Thomsen, M., & Landay, J. A. (2002). *A Visual Language for Sketching Large and Complex Interactive Designs*. Proceedings of the Conference on Human Factors in Computing Systems CHI'02 (20-25 April, Minneapolis, United States).

[Lok01]

Lok, S., & Feiner, S. (2001). *A Survey of Automated Layout Techniques for Information Presentations*. In Proceedings of the 1st International Symposium on SmartGraphics (21-23 March, Hawthorne, United States).

[Lonc96]

Lonczewski, F., & Schreiber, S. (1996). *The FUSE-System: an Integrated User Interface Design Environment*. Proceedings of the 2nd International Workshop on Computer-Aided Design of User Interfaces CADUI'96 (5-7 June, Namur).

[Luo93]

Luo, P., Szekely, P., & Neches, R. (1993). *Management Of Interface Design In Humanoid*. Proceedings of the Conference on Human Factors in Computing Systems INTERCHI'93 (24-29 April, Amsterdam).

[Luyt03]

Luyten, K., Van Laerhoven, T., Coninx, K., & Van Reeth, F. (2003). Runtime transformations for modal independent user interface migration. *Interacting with Computers*, 15(3).

[Luyt03b]

Luyten K., Clerckx T., Coninx K. and Vanderdonckt J., *Derivation of a Dialog Model from a Task Model by Activity Chain Extraction*, In Preliminary Proceedings of DSV-IS'2003 (Funchal, Madeira Island, Portugal, 4-6 June 2003).

References

[Luyt04]

Luyten, K. (2004). Dynamic User Interface Generation for Mobile and Embedded Systems with Model-Based User Interface Development. Unpublished PHD thesis, transnationale Universiteit Limburg, Diepenbeek, Belgium.

M

[Mand02]

Mandyam, S., Vedati, K., Kuo, C., & Wang, W. (2002). *User Interface Adaptations: Indispensable for Single Authoring*. Proceedings of the W3C Workshop on Device Independent Authoring Techniques (25-26 September, St. Leon-Rot, Germany).

[Mari05]

Mariage, C. (2005). *MetroWeb: logiciel de support à l'évaluation de la qualité ergonomique des sites web*. Unpublished PHD thesis, Université catholique de Louvain, Louvain-la-Neuve.

[Mart90]

Martin, C. (1990). *A UIMS for knowledge based interface template generation and interaction*. Proceedings of the IFIP TC13 3rd International Conference on Human-Computer Interaction INTERACT'90 (27-31 August, Cambridge, UK).

[Menk02]

Menkhaus, G. (2002). A hybrid approach to adaptive user interface generation. *CIT - Journal of Computing and Information Technology*, 10(3), 171-179.

[Menk03]

Menkhaus, G., & Fischmeister, S. (2003). *Evaluation of User Interface Transcoding Systems*. Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics (27-30 July, Orlando, United States).

[Menk03b]

Menkhaus, G., & Fischmeister, S. (2003). *Dialog Model Clustering for User Interface Adaptation*. Proceedings of the 3rd International Conference on Web Engineering ICWE 03 (16-18 July, Oviedo, Spain).

[Meye97]

Meyers, B. (1997). Object-Oriented Software Construction (2d ed.). New York: Prentice Hall.

[Mont05]

Montero, F., López-Jaquero, V., Vanderdonckt, J., Gonzalez, P., Lozano, M. D., & Limbourg, Q. (2005). *Solving the Mapping Problem in User Interface Design by Seamless Integration in IdealXML*. Proceedings of the 12th International Workshop on Design, Specification and Verification of Interactive Systems DSVIS'05 (13-15 July, Newcastle upon Tyne, UK).

References

[Mori03]

Mori, G., Paternò, F., & Santoro, C. (2003). *Tool Support for Designing Nomadic Applications*. Proceedings of the 8th international conference on Intelligent user interfaces IUI'03 (12-15 January, Miami).

[Myer90]

Myers, B., Giuse, D., Dannenberg, R., Vander Zanden, B., Kosbie, D., Pervin, E., et al. (1990). Garnet: Comprehensive Support for Graphical, Highly-Interactive User Interfaces. *IEEE Computer*, 23(11), 71-85.

[Myer97]

Myers, B., McDaniel, R., Miller, R., Ferrenzy, A., Faulring, A., Kyle, B., et al. (1997). The Amulet Environment: New Models for Effective User Interface Software Development. *IEEE Transactions on Software Engineering*, 23(6), 347-365.

[Myer00]

Myers, B., Hudson, S., & Pausch, R. (2000). Past, present, future of user interface tools. *ACM Transactions on Computer-Human Interaction*, 7(1), 3-28.

[Myer03]

Myers, B. (2003). Graphical User Interface Programming. In A. B. Tucker (Ed.), *CRC Handbook of Computer Science and Engineering* (2d ed.). Boca Raton, United States: CRC Press.

N

[Nich02]

Nichols, J., Myers, B. A., Higgins, M., Hughes, J., Harris, T. K., Rosenfeld, R., et al. (2002). Generating Remote Control Interfaces for Complex Appliances. Proceedings of the 15th annual symposium on User Interface Software & Technology UIST'02 (27-30 October, Paris).

[Nich04]

Nichols, J., Myers, B. A., & Litwack, K. (2004). Improving Automatic Interface Generation with Smart Templates. Proceedings of the 8th International Conference on Intelligent User Interfaces IUI'2004 (13-16 January, Funchal, Portugal).

O

[Oliv01]

Oliveira, M. C. F. d., Turine, M. A. S., & Masiero, P. C. (2001). A statechart-based model for hypermedia applications. *ACM Transactions on Information Systems (TOIS)*, 19(1), 28-52.

References

[Olle88]

Olle, T. W., Hagelstein, J., Macdonald, I. G., Rolland, C., Sol, H. G., Van Assche, F., et al. (1998). *Information Systems Methodologies, a framework for understanding*. Reading (United States): Addison-Wesley.

[Olse98]

Olsen, D. R. (1998). *Developing User Interfaces*. San Francisco: Morgan Kaufmann Publishers.

[OMG05]

OMG. (2005). UML OCL 2.0 Specification Version 2.0. Retrieved January, 2006, from <http://www.omg.org/docs/ptc/05-06-06.pdf>

[Oust94]

Ousterhout, J. (1994). Tcl and TK Toolkit. Reading (United States): Addison Wesley.

P

[Pate00]

Paternò, F. (2000). *Model-Based Design and Evaluation of Interactive Applications*. Berlin: Springer-Verlag.

[Pate02]

Paternò, F., & Santoro, C. (2002). *One Model, Many Interfaces*. Proceedings of the 4th International Conference on Computer-Aided Design of User Interfaces CADUI'02 (15-17 May, Valenciennes, France).

[Paus92]

Pausch, R., Conway, M., & DeLin, R. (1992). Lesson Learned from SUIT, the Simple User Interface Toolkit. *ACM Transactions on Information Systems (TOIS)*, 10(4), 320-344.

[Phan00]

Phanouriou, C. (2000). *UIML: A Device-Independent User Interface Markup Language*. Unpublished PHD thesis, Virginia Polytechnic Institute, Blacksburg.

[Pier04]

Jeffrey S. Pierce, J. S., & Mahaney, H. E. (2004). Opportunistic Annexing for Handheld Devices: Opportunities and Challenges. Human-Computer Interface Consortium 2004. Retrieved June, 2006, from <http://www-static.cc.gatech.edu/~jpierce/papers/OA-HCIC2004.pdf>

[Puer96]

Puerta, A. R. (1996). *The Mecano Project: Comprehensive and Integrated Support for Model-Based Interface Development*. Proceedings of the 2nd International Workshop on Computer-Aided Design of User Interfaces CADUI'96 (5-7 June, Namur).

[Puer97]

Puerta, A. R. (1997). A model-based interface development environment. *IEEE Software*, 14(4), 40-47.

References

[Puer98]

Puerta, A. R. (1998). *Supporting User-Centered Design of Adaptive User Interfaces Via Interface Models*. Proceedings of the 1st Annual Workshop On Real-Time Intelligent User Interfaces For Decision Support And Information Visualization (January, San Francisco).

[Puer99]

Puerta, A. R., & Eisenstein, J. (1999). *Towards a General Computational Framework for Model-Based Interface Development Systems*. Proceedings of the 4th International Conference on Intelligent User Interfaces IUI'99 (5-8 January, Los Angeles).

[Puer02]

Puerta, A. R., & Eisenstein, J. (2002). XML: a common representation for interaction data. Proceedings of the 7th International Conference on Intelligent User Interfaces IUI'02 (13-16 January, San Francisco).

R

[Rati97]

Rational Software (1997), *UML Semantics*, v.1.1. Retrieved January, 2006, from http://umlcenter.visual-paradigm.com/umlresources/sema_11.pdf

S

[Schl97]

Schlunbaum, E. (1997). *Individual user interfaces and model-based user interface software tools*. Proceedings of the 2nd International Conference on Intelligent User Interfaces IUI '97 (6-9 January, Orlando, United States).

[Seff04]

Seffah, A., & Javahery, H. (2004). Multiple user Interfaces: Cross-Platform Applications and Context-Aware Interfaces. In A. Seffah & H. Javahery (Eds.), *Multiple User Interfaces: Engineering and Application Framework* (pp. 11-26). Chichester (GB): Wiley and Sons.

[Simo05]

Simon, R., Wegscheider, F., & Tolar, K. (2005). *Tool-supported single authoring for device independence and multimodality*. Proceedings of the 7th international conference on Human computer interaction with mobile devices & services MobileHCI '05 (19 - 22 September, Salzburg, Austria).

[Somm92]

Sommerville, I. (1992). *Software engineering*. Wokingham (UK): Addison-Wesley.

References

[Souc02]

Souchon, N. (2002). *Towards a Computational Notation for Supporting Context-Sensitive User Interface Development*. Unpublished 3rd cycle master thesis, Université catholique de Louvain, Louvain-la-Neuve.

[Souc03]

Souchon, N., & Vanderdonckt, J. (2003). *A Review of XML-Compliant User Interface Description Languages*. Proceedings of the 10th International Conference on Design, Specification, and Verification of Interactive Systems DSV-IS'03 (4-6 June, Funchal, Portugal).

[Spri03]

Priestersbach, A., Ziegert, T., Grassel, G., Wasmund, M., & Dermier, G. (2003). *Flexible pagination and layouting for device independent authoring*. Proceedings of the WWW2003 Emerging Applications for Wireless and Mobile access Workshop (May 20th, Budapest).

[Szek92]

Szekely, P., Luo, P., & Neches, R. (2002). *Facilitating the Exploration of Interface Design Alternatives: The HUMANOID Model of Interface Design*. Proceedings of the Conference on Human Factors in Computing Systems CHI'92 (3-7 May, Monterey, United States).

[Szek93]

Szekely, P., Luo, P., & Neches, R. (1993). *Beyond Interface Builders: Model-Based Interface Tools*. Proceedings of the Conference on Human Factors in Computing Systems INTERCHI'93 (24-29 April, Amsterdam).

[Szek94]

Szekely, P. (1994). *User Interface Prototyping: Tools and Techniques*. Proceedings of the Software Engineering and Human-Computer Interaction ICSE '94 Workshop on SE-HCI: Joint Research Issues (16-17 May, Sorrento, Italy).

[Szek95]

Szekely, P., Sukaviriya, P., Castells, P., Muthukumarasamy, J., & Salcher, E. (1995). *Declarative interface models for user interface construction tools: the MASTERMIND approach*. Proceedings of the IFIP Working Conference on Engineering for Human-Computer Interaction EHCI'95 (14-18 August, Grand Targhee Resort, United States).

[Szek96]

Szekely, P. (1996). *Retrospective and Challenges for Model-Based Interface Development*. Proceedings of the 2nd International Workshop on Computer-Aided Design of User Interfaces CADUI '96 (5-7 June, Namur).

References

T

[Thev99]

Thevenin, D., & Coutaz, J. (1999). *Plasticity of user interfaces: Framework and research agenda*. Proceedings of the 7th International Conference on Human-Computer Interaction Interact'99 (30 August - 3 September, Edinburgh).

[Thev01]

Thevenin, D. (2001). *Adaptation en Interaction Homme-Machine: le cas de la Plasticité*. Unpublished PHD thesis, Université Joseph Fourier, Grenoble.

V

[Vand93]

Vanderdonckt, J., & Bodart, F. (1993). *Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection*. Proceedings of the Conference on Human Factors in Computing Systems InterCHI'93 (14-19 April, Amsterdam).

[Vand95]

Vanderdonckt, J. (1995). *Accessing Guidelines Information with SIERRA*. Proceedings of the International Conference on Human-Computer Interaction Interact'95 (27-29 June, Lillehammer, Norway).

[Vand97]

Vanderdonckt, J. (1997). *Conception Assistée de la Présentation D'une Interface Homme-Machine Ergonomique Pour Une Application de Gestion Hautement Interactive*. Unpublished PHD thesis, Facultés Universitaires Notre-Dame de la Paix, Namur.

[Vand98]

Vanderdonckt, J. (1998). *Une description orientée objet des objets interactifs abstraits utilisés dans les Interfaces Homme-Machine*. Namur: Facultés Universitaires Notre-Dame de la Paix.

W

[W3C03]

W3C. (2003). *XForms 1.0, W3C Recommendation*. Retrieved January, 2006, from <http://www.w3.org/TR/xforms/>

[W3C04]

W3C. (2004). *Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies 1.0, W3C Recommendation*. Retrieved January, 2006, from <http://www.w3.org/TR/CCPP-struct-vocab/>

References

[W3C04b]

W3C. (2004). Authoring Techniques for Device Independence, W3C Working Group Note 18 February 2004. Retrieved March, 2006, from <http://www.w3.org/TR/2004/NOTE-di-atdi-20040218/>

[Warm99]

Warmer, J., & Kleppe, A. (1999). *The object constraint language: precise modeling with UML*. Boston: Addison-Wesley.

[Watt03]

Watters, C., & Zhang, R. (2003). *PDA Access to Internet Content: Focus on Forms*. Proceedings of the 36th Annual Hawaii International Conference on System Sciences HICSS'03 (6-9 January, Big Island, United States).

[Weis02]

Weiss, S. (2002). *Handheld Usability*. Chichester (UK): John Wiley & Sons.

[Wiec89]

Wiecha, C., Bennett, W., Boies, S., & Gould, J. (1989). *Generating Highly Interactive User Interfaces*. Proceedings of the Conference on Human Factors in Computing Systems CHI'89 (30 April-4 May, Austin).

[Wils96]

Wilson, S., & Johnson, P. (1996). *Bridging the Generation Gap: From Work Tasks to User Interface Designs*. Proceedings of the 2nd International Workshop on Computer-Aided Design of User Interfaces CADUI'96, Namur).

[Wong02]

Wong, C., Chu, H., & Katagiri, M. (2002). A Single-Authoring Technique for Building Device-Independent Presentations. Proceedings of the W3C Workshop on Device Independent Authoring Techniques (25-26 September, St. Leon-Rot, Germany).

Y

[Ye04]

Ye, J., & Herbert, J. (2004). User Interface Tailoring for Mobile Computing Devices. Proceedings of the 4th International Conference on Web Engineering ICWE 2004 (28-30 July, Munich).

References

Z

[Zieg04]

Ziegert, T., Lauff, M., & Heuser, L. (2004). *Device Independent Web Applications – The Author Once – Display Everywhere Approach*. Proceedings of the 4th International Conference on Web Engineering ICWE 2004 (28-30 July, Munich).

Annex A. CTT/UsiXML task model

A CTT/UsiXML task model is a hierarchical task structure, where each task is described by:

- An *identifier* and a *name*.
- A *type*, which is determined by the allocation of the task: a task performed by the user (e.g. a cognitive task) is called a *user task*. A task completely executed by the system (e.g. a computation task) has category *application task*. A task performed by the user in interaction with the system (e.g. viewing results, selecting items, editing a field, pushing a button to invoke an application function) is called an *interaction task*. Last, *abstraction tasks* (e.g. booking a flight) are complex tasks whose performance can not be univocally allocated and that can be decomposed into simpler tasks (thus, there must be at least two different task categories among the tasks decomposing an abstraction task).
- Optional attributes such as the task *importance*, *frequency*...

Tasks are linked by two types of relationships:

- *Hierarchical relationships*. Each task can be decomposed into two or more subtasks. Thus, with the exception of the root task, each task has a mother task from which the temporal relationships are inherited.
- *Temporal relationships*. Temporal relationships between the tasks are specified with *temporal operators*. The temporal operators are based upon the LOTOS operators.

Temporal relationships are of two types: unary and binary. Unary operators characterize a single task when binary operators link together two sibling tasks.

There are three unary operators. The first one is the iteration operator (notation: T^*), which means that the task T is repeated until some other task disables it. The second one, is the finite iteration operator (notation $T(n)$), used when the designer knows in advance exactly how many time the task will be performed. The last operator permits indicating that the performance of a task is optional (notation $[T]$).

If we consider two generic tasks $T1$ and $T2$, the binary temporal operators can be described as follows:

1. *Independent concurrency* or *parallelism* ($T1 \parallel T2$): $T1$ and $T2$ can be performed in any order without any constraints. E.g.: filling field 1 and field 2 in a form.
2. *Concurrency* (or parallelism) *with information exchange* ($T1 \parallel\!\!\parallel T2$): $T1$ and $T2$ can be performed in any order but they have to synchronize in order to exchange

Annex A CTT/UsiXML task model

information. E.g. filling field1 and field 2 in a form when there is some coherency check (between a phone number and a city for example).

3. *Deterministic choice* ($T1 \sqcap T2$): Once one task is initiated, the other cannot be accomplished anymore, until the first task is terminated. E.g. log in as a reviewer or as an author on a conference reviewing system.
4. *Non-deterministic choice* ($T1 \pi T2$): Once one task is finished the other cannot be accomplished anymore. E.g., saving one's bank statements to one's desktop computer or printing them in the bank's self-service lobby.
5. *Order independency* or *sequential independence* ($T1 \mid = \mid T2$). This operator is equivalent to ($T1 >> T2$) OR ($T2 >> T1$) E.g., in a hospital, the human task of taking blood samples from patients can be done before or after filling the request form for lab analysis, but both tasks have to be completed before the request is send to the lab.
6. *Disabling* ($T1 \sqsupset T2$): T1 is definitively disabled when T2 (or the first subtask or T2) has been performed. E.g., sending a form disables all tasks that could be achieved in this form.
7. *Suspend-resume* ($T1 \mid > T2$): T1 is interrupted when T2 (or its first subtask) is performed. Once T2 terminated, T1 is reactivated from the state reached before the interruption. E.g., an alarm message indicating that the battery of the device is low interrupts any activity on that device, and the activity is reactivated only when the alarm dialog box is closed.
8. *Enabling* ($T1 >> T2$): T2 is enabled when T1 is terminated. E.g., the authentication of the user allows him/her to access to the restricted area of a Web site.
9. *Enabling with information passing* ($T1 \sqcap >> T2$): T1 enables T2 and provides it some information. E.g., T1 allows the user to specify a query and T2 displays the search results related to the information requested in T1.

Annex B. Discussion of UsiXML's platform model attributes

Component: HardwarePlatform

Attribute	Description	Discussion	Decision
BitsPerPixel	The number of bits of colour or greyscale information per pixel	Does not seem useful in a GD context	Rejected
BluetoothProfile	Supported Bluetooth profiles as defined in the Bluetooth specification	Does not seem to have any impact on the user interface	Rejected
Category	Category of the device	Can be used by GD rules. Example of GD rule : for all transformations between a desktop and a PDA, place the input controls in separate lines instead of going horizontally (moving rule)	Added
ColourCapable	Whether the device display supports colour	Can be used by GD rules (example of GD rule : represent the level of emphasis by differences in font type and size instead of using colours)	Adopted
CPU	Name and model number of device CPU	May be useful in a GD context although general assumptions about the device's performances can be made from other attributes (the model for example)	Adopted
ImageCapable	Whether the device supports the display of images	Can be used by GD rules (substitution rule : replace an image by its textual representation)	Adopted
InputCharSet	List of character sets supported by the device for text entry	May be useful (see SoftwarePlatform CcppAccept-Charset)	Adopted
Keyboard	Type of keyboard supported by the device	Can be used by GD rules (moving rule : when the user is expected to use the soft input panel, place menus at the top of the page and not at the bottom where they would be masked by the soft keyboard or character recognition area).	Adopted
Model	Model number assigned to the terminal device by the vendor or manufacturer	Can be used by GD rules : in combination with the vendor's name, define the precise type	Adopted

Annex B Discussion of UsiXML's platform model attributes

		of hardware and permits to define rules at a quite high precision level. Example: for all transformations between a desktop and a Compaq iPaq Pocket PC, regroup menu bar and toolbar into a command bar (interactor substitution rule: regrouping).	
NumberOfColours	Number of colours the display supports	Can be used by GD rules (substitution rules applied to images for example)	Added
NumberOfGreyScale	Number of grey scale the display supports	See NumberOfColours	Added
NumberOfSoftKeys	Number of soft keys available on the device.	Can be used by complex GD rules that could achieve substitution between menu items or soft buttons and soft keys.	Adopted
OutputCharSet	List of character sets supported by the device for output to the display	May be useful (see SoftwarePlatform CcppAccept-Charset)	Adopted
PointingDevice	Type of pointing device	Can be used by GD rules. Example : delete all tooltips or replace them by labels when a stylus is used instead of a mouse (onMouseOver event no more available)	Added
PointingResolution	Type of resolution of the pointing accessory supported by the device	Can be used by GD rules (example : resizing rules when the pointing resolution changes)	Adopted
RAMMemory	RAM memory on the device (Mb)	Can be used by GD rules (example: interactor substitution rules)	Added
ScreenResolution	Resolution of the device's screen (dpi)	Can be used by GD rules (example: fonts resizing rules, ...)	Added
ScreenSize	The size of the device's screen in units of pixels	Can be used by GD rules (example: moving rules, resizing rules, ...)	Adopted
ScreenSizeChar	Size of the device's screen in units of characters	Can be used by GD rules (example: moving rules, resizing rules, ...)	Adopted
SoundOutputCapable	Indicates whether the device supports sound output	Can be used in advanced GD rules using multimodality	Adopted
StandardFontProportional	Indicates whether the device's standard font is proportional.	Fonts should be linked with the software they are installed with.	Rejected
StorageCapacity	Secondary memory capacity (hard disk, flash memory, ...)	Can be used in advanced GD rules. Example: delete all data intensive tasks on devices where no storage capacity is available.	Added
TextInputCapable	Indicates whether the device	Can be used by GD rules.	Adopted

Annex B Discussion of UsiXML's platform model attributes

	supports alphanumeric text entry	(Example: if the device does not support text entry, try to replace all Text Inputters by Text Choosers).	
TouchScreen	Indicates whether the hardware has a touchscreen or not	Can be used by GD rules. (Example: a soft key could be replaced by a button in the UI if a touchscreen is available)	Added
Vendor	Name of the vendor manufacturing the terminal device	Can be used by GD rules (see « Model »)	Adopted
VoiceInputCapable	Indicates whether the device supports any form of voice input, including speech recognition	Can be used in advanced GD rules using multimodality	Adopted

Component: SoftwarePlatform

Attribute	Description	Discussion	Decision
AcceptDownloadableSoftware	Indicates the user's preference on whether to accept downloadable software	Elements describing the user's preferences should not be part of a platform model	Rejected
AudioInputEncoder	List of audio input encoders supported by the device	Useful for substitution rules due to unavailability	Adopted
CcppAccept	List of content types the device supports	May be redundant with other attributes (AudioInputEncoder, VideoInputEncoder, OutputCharSet) but still useful for substitution rules due to unavailability	Adopted
CcppAccept-Charset	List of character sets the device supports	Can be useful in a GD context. (Example : if the UI contains ISO-LATIN characters and the target device only supports ASCII, replace all characters with diacritics with a counterpart without diacritics)	Adopted
CcppAccept-Encoding	List of transfer encodings the device supports	Does not seem useful in a GD context	Rejected
CcppAccept-Language	List of preferred document languages	Does not seem useful in a GD context	Rejected
DownloadableSoftwareSupport	List of executable content types which the device supports and which it is willing to accept from the network (list of MIME types)	No direct impact on the user interface.	Rejected
HandwritingRecognitionSoftware	List of handwriting recognition software the device supports	May have an impact on the usability of some text input widgets	Added
JavaEnabled	Indicates whether the device supports a Java virtual machine.	Together with the attributes JavaPlatform and JVMVersion, may be useful in order to	Adopted

Annex B Discussion of UsiXML's platform model attributes

		characterize the device's set of available widgets	
JavaPlatform	The list of JAVA platforms and profiles installed in the device. Each item in the list is a name token describing compatibility with the name and version of the java platform specification or the name and version of the profile specification name (if profile is included in the device)	See JavaEnabled	Adopted
JVMVersion	List of the Java virtual machines installed on the device. Each item in the list is a name token describing the vendor and version of the VM.	See JavaEnabled	Adopted
MexeClassmark	ETSI MExE classmark	Standard related to 3 rd generation mobile systems. Describe a terminal's capability and user preference. Seems to be redundant.	Rejected
MexeSecureDomains	Indicates whether the device's supports MExE security domains. "Yes" means that security domains are supported in accordance with MExE specifications identified by the MexeSpec attribute. "No" means that security domains are not supported and the device has only untrusted domain (area).	See MexeClassmark	Rejected
MexeSpec	Class mark specialization	See MexeClassmark	Rejected
OSName	Name of the device's operating system	Together with the OS vendor name and OS version, permits to define the set of native widgets and fonts available on the platform --> useful in all GD rules when native widgets are used	Adopted
OSVendor	Vendor of the device's operating system	See OSName	Adopted
OSVersion	Version of the device's operating system	See OSName	Adopted
RecipientAppAgent	User agent associated with the current request	Identification of the user agent is more suitable for adaptation to the user	Rejected
SoftwareNumber	Version of the device specific software (firmware) to which the device's low-level software conforms	Not precise enough. Which software?	Rejected

Annex B Discussion of UsiXML's platform model attributes

SpeechRecognitionSoftware	List of speech recognition software the device supports	Can be used in advanced GD rules using multimodality, in combination with the hardware attribute VoiceInputCapable	Added
VideoInputEncoder	List of video input encoders supported by the device	Useful for substitution rules due to unavailability	Adopted

Component: NetworkCharacteristics

Attribute	Description	Discussion	Decision
Capacity	Amount of data that can be sent through a given communications circuit per second.	Can be used by some GD rules (example : if the capacity is low, delete tasks requiring access to the server, or modify their priority so that they appear in a less accessible part of the UI)	Added
CostPerVolume	Cost per unit of data transferred	Can be used by some GD rules (see capacity)	Added
CostPerTime	Cost per time using the communications channel	Can be used by some GD rules (see capacity)	Added
CurrentBearerService	The bearer (channel) on which the current session was opened	Too technical to be directly useful for user interface adaptation	Rejected
SecuritySupport	Type of security or encryption mechanism supported	Too technical to be directly useful for user interface adaptation	Rejected
SupportedBearers	List of bearers supported by the device	Too technical to be directly useful for user interface adaptation	Rejected
SupportedBluetoothVersion	Supported Bluetooth version.	Too technical to be directly useful for user interface adaptation	Rejected

Component: BrowserUA

Attribute	Description	Discussion	Decision
BrowserName	Name of the browser user agent associated with the current request	Together with the BrowserVersion attribute, may be useful to establish the set of HTML tags actually supported	Adopted
BrowserVersion	Version of the browser	See BrowserName	Adopted
DownloadableBrowserApps	List of executable content types which the browser supports and which it is willing to accept from the network	Not directly linked to user interfaces design	Rejected
FramesCapable	Indicates whether the browser is capable of displaying HTML frames	Useful to establish the precise widgets set	Adopted
HtmlVersion	Version of HyperText Mark-up Language (HTML)	Useful to establish the precise widgets set	Adopted

Annex B Discussion of UsiXML's platform model attributes

	supported by the browser		
JavaAppletEnabled	Indicates whether the browser supports Java applets.	Useful to establish the precise widgets set	Adopted
JavaScriptEnabled	Indicates whether the browser supports JavaScript.	Together with the attribute JavaScriptVersion, useful to establish the precise widgets set, for non HTML widgets programmed with JavaScript	Adopted
JavaScriptVersion	Version of the JavaScript language supported by the browser	See JavaScriptEnabled	Adopted
MexeSecureDomains	Indicates whether the device's supports MExE security domains. "Yes" means that security domains are supported in accordance with MExE specifications identified by the MexeSpec attribute. "No" means that security domains are not supported and the device has only untrusted domain (area).	Probably of no use for user interfaces design	Rejected
PreferenceForFrames	Indicates the user's preference for receiving HTML content that contains frames	User preferences should not be part of a Platform model	Rejected
TablesCapable	Indicates whether the browser is capable of displaying HTML tables	Useful to establish the precise widgets set	Adopted
XhtmlVersion	Version of XHTML supported by the browser	Useful to establish the precise widgets set	Adopted
XhtmlModules	List of XHTML modules supported by the browser	Useful to establish the subset of Xhtml actually supported	Adopted

Component: WapCharacteristics

Attribute	Description	Discussion	Decision
SupportedPictogramSet	Pictogram classes supported by the device as defined in "WAP Pictogram specification".	Useful to establish the available image set on WAP phone --> useful for some specific GD rules applied to WAP phones	Adopted
WapDeviceClass	Classification of the device based on capabilities as identified in the WAP 1.1 specifications	Redundant with Hardware attribute "Category", but may be interesting to have an industry-standard type	Adopted
WapVersion	Version of WAP supported	The subsets of the WAP protocol useful for user interfaces are already defined elsewhere. (WmlScriptVersion, WmlVersion...)	Rejected

Annex B Discussion of UsiXML's platform model attributes

WmlDeckSize	Maximum size of a WML deck that can be downloaded to the device	Useful for user interfaces in general	Adopted
WmlScriptLibraries	List of mandatory and optional libraries supported in the device's WMLScript VM	Together with the WmlScriptVersion attribute, useful for the dialog part of user interfaces	Adopted
WmlScriptVersion	List of WMLScript version numbers supported by the device	See WmlScriptLibraries	Adopted
WmlVersion	List of WML language version numbers supported by the device	Useful to establish the precise widgets set	Adopted
WtaiLibraries	List of WTAI network common and network specific libraries supported by the device that are URI accessible	Only useful for wireless telephony applications	Rejected
WtaVersion	Version of WTA user agent	Only useful for wireless telephony applications	Rejected

Component : PushCharacteristics

Attribute	Description	Discussion	Decision
Push-Accept	List of content types the device supports, which can be carried inside the message/http entity body when OTA-HTTP is used. Property value is a list of MIME types, where each item in the list is a content type descriptor as specified by FC 2045.	Too specific. No direct impact on user interfaces.	Rejected
Push-Accept-Charset	List of character sets the device supports. Property value is a list of character sets, where each item in the list is a character set name registered with IANA.	Too specific. No direct impact on user interfaces.	Rejected
Push-Accept-Encoding	List of transfer encodings the device supports. Property value is a list of transfer encodings, where each item in the list is a transfer encoding name as specified by RFC 2045 and registered with IANA.		
Push-Accept-Language	List of preferred document languages. If a resource is available in more than one natural language, the server can use this property to		

Annex B Discussion of UsiXML's platform model attributes

	determine which version of the resource to send to the device. The first item in the list should be considered the user's first choice, the second the second choice, and so on. Property value is a list of natural languages, where each item in the list is the name of a language as defined by RFC 3066[RFC3066].		
Push-Accept-AppID	List of applications the device supports, where each item in the list is an application-id on absolute URI format as specified in [PushMsg]. A wildcard ("*") may be used to indicate support for any application.	Too specific. No direct impact on user interfaces.	Rejected
Push-MsgSize	Maximum size of a push message that the device can handle. Value is number of bytes.		
Push-MaxPushReq	Maximum number of outstanding push requests that the device can handle.		

The first attribute of the hardware component deserves more explanation. Intuitively, we see that it is possible to establish categories of devices that share common features. For example, Scott Weiss [Weis02] makes a distinction between desktops, laptops, palmtops and handhelds, where the last category is further segmented into PDA's, mobile phones, pagers and communicators. Weiss's categories are based on two criteria: size and portability. In Weiss's illustration (Figure A-0-1), the four basic types of computing platform are presented by order of increasing portability and decreasing size. The four device types overlap to show that the categories are not discrete.

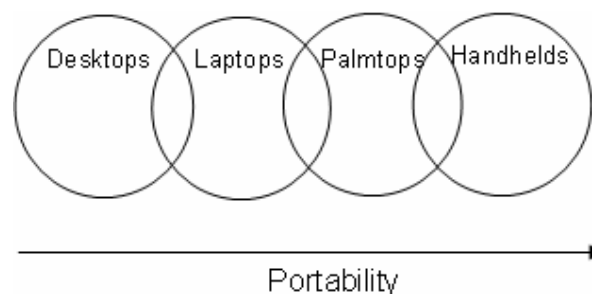


Figure A-0-1 The personal computing continuum (illustration from [Weis02])

Desktops are typically used in a stationary context: they are used while seated and require a table. They require a power cord and are not mobile. They have good storage capabilities

Annex B Discussion of UsiXML's platform model attributes

(hard disks with 80 GB or more, CDRW, etc.) and memory up to 3 GB. They have fast and reliable network connectivity, often at low cost. They have a good screen resolution (a minimum of 800 x 600 pixels), a keyboard and a mouse.

Laptops computers share more of the characteristics of desktops in terms of performance, but they enable mobility. However, they are still large and heavy, require frequent battery recharging and are used in a stationary context.

Palmtops are smaller and lighter than laptops. They have batteries, which have to be recharged frequently. They are best used while seated on a table, which differentiates them from the next category.

Handheld devices are lightweight and small enough to be put into a pocket. They can be used while standing. They are completely mobile and have batteries with enough life length. They have less or no storage capacities and memory up to 64 MB. Input devices may include a stylus, keypad, mini-keyboard, roller wheel. Display sizes top out at 320 x 320 pixels. Figure A-0-2 shows the four types of handheld devices that can be distinguished.

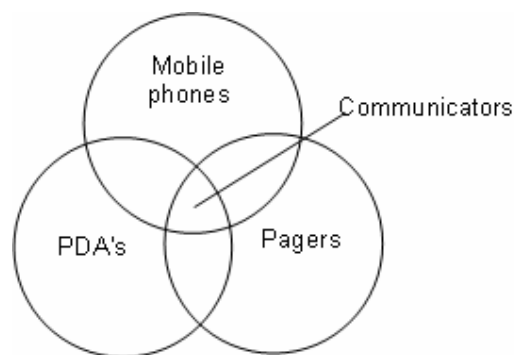


Figure A-0-2 Handheld devices categories (illustration from [Weis02])

The internal classification of handheld devices is based on their differences of use, input method, display size and other capabilities and their expandability (add-ons of software and/or hardware):

- *Mobile phones* are first used for voice calls and SMS, then for WAP. They usually have a 12-key keypad, very small screens (a few text lines only, sometimes pictures can be displayed). They are not expandable.
- *PDA's* are typically used for information storage and retrieval. The available input methods are the stylus, on-screen keypad, some hard buttons, sometimes a small keyboard. The screen resolution is ranging from 160x160 to 320x480 pixels. Additional applications and hardware components can be added.
- *Pagers* are used primarily for email communication, with additional features such as an address book and calendar, and sometimes Web browsing. Pagers have tiny keyboards but no touch screen. Some pagers support application downloads.
- *Communicators* (e.g. Smartphone) combine features of the other categories.

Annex B Discussion of UsiXML's platform model attributes

This classification deliberately excludes portable consumer electronic devices such as calculators, digital cameras or MP3 players, because they lack both the ability to add applications and the connectivity to the Internet, as well as game machines, which usage and design are very different from usual handheld devices. According to Weiss's classification, we can establish a first list of device categories: *Desktops*, *Laptops computers*, *Palmtops*, *Mobile phones*, *PDA*, *Pagers* and *Communicators*.

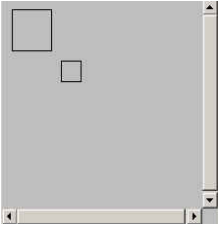
The following computing devices can be added to this list:

- *Tablet PCs* are small, ultra-mobile laptops with convertible screens that can be used in normal laptop mode, or flipped around and used like a tablet, with stylus and on-screen keyboard input. They have handwriting-recognition and sometimes voice-recognition input capabilities. They have roughly the same power and performance of a PC, with a pen and letter size screen.
- *Interactive kiosks* are interactive terminals intended for use in a given public area. They accept user input and display information, with sometimes access to the Web. They are desktops equipped with a touch screen, which is the primary and often the only interface. Sometimes, they have additional peripherals such as a keyboard, printer, card reader, ...
- *Screenphones* are multifunction telephones that also provide access to the Internet. They have a slidable keyboard and a touchscreen, ports for a mouse, printer and other communications and video peripherals. Like a classic desktop, they are expandable and upgradeable.
- *WebTVs* are TV offering access to Internet thanks to a special receiver, and sometimes a special remote control and wireless keyboard.

Following Weiss's criteria, we deliberately omit platforms that neither provide Internet connectivity, nor allow the addition of applications, such as car navigation systems.

In conclusion, here is a list of admissible values for the Category attribute: Desktop, Laptop, Palmtop, Mobile phone, PDA, Pager, Communicator, Tablet PC, Interactive kiosk, Screenphone, WebTV. Of course, this list and the description of the categories are expected to evolve rapidly: new kinds of devices are regularly launched to the market, and existing devices of all categories become increasingly more powerful, are equipped with higher-quality displays, new multimedia peripherals...

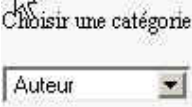
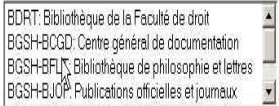
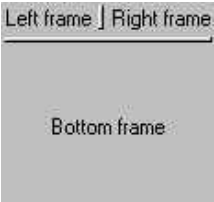
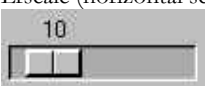
Annex C. An Interactor Model for the QTk Toolkit

Final Widget (QTK toolkit)	Graphical CIO	Extended AIO
Button 	Button	ActionItem Facet: (T: activation D: _ Card: _ IsCentral: true)
Canvas 	Canvas	Graphical Editor Facet: (T: specification D: graphics Card: N IsCentral: true) Facet: (T: consultation D: graphics Card: N IsCentral: false)
Checkbutton 	CheckBox	Checker Facet: (T: selection D: boolean Card: 1 IsCentral: true) Facet: (T: consultation D: boolean Card: 1 IsCentral: false)

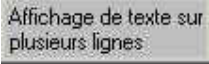
Annex C: An Interactor Model for the QtK Toolkit

N Checkbuttons 	Group of N CheckBoxes	Multiple Text Chooser Facet: (T: selection D: string Card: N (ergonomic rule : $N \leq 7$) IsCentral: true) Facet: (T: consultation D: string Card: N IsCentral: false)
Dropdownlistbox. 	ComboBox	Simple Text Chooser Facet: (T: selection D: string Card: 1 IsCentral: true) Facet: (T: consultation D: string Card: N IsCentral: false) OR Multiple Text Chooser (not typical of this object) Facet: (T: selection D: string Card: $M \leq N$ IsCentral: true) Facet: (T: consultation D: string Card: N IsCentral: false)
Entry 	EditBox (preferred to UsiXML's tag <textComponent isEditable="true">)	Text Inputter Facet: (T: specification D: string Card: 1 IsCentral: true)

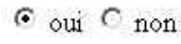
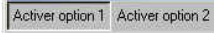
Annex C: An Interactor Model for the QTk Toolkit

Label 	Label (preferred to UsiXML's tag <textComponent isEditable="false">)	Documentation Object Facet: (T: consultation D: string Card: 1 IsCentral: true)
Listbox (+ tdscrollbar) 	ListBox	Multiple Text Chooser Facet: (T: selection D: string Card: $M \leq N$ (ergonomic rule : $N > 6$) IsCentral: true) Facet: (T: consultation D: string Card: N IsCentral: false)
Tdline (vertical separator) or Lrline (horizontal separator) 	Separator	Separator Object Facet: _
Tdrubberframe or Lrrubberframe 	ExtensibleFrame	Container + placement relationship Facet: _
Tdscale (vertical scale) or Lrscale (horizontal scale) 	Slider	Simple Number Chooser Facet: (T: selection D: integer Card: 1 IsCentral: true) Facet: (T: consultation D: interval of N integers Card: 1 IsCentral: false)

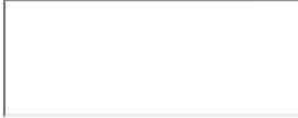
Annex C: An Interactor Model for the QtK Toolkit

Tdscrollbar (vertical scrollbar) ou Lrscrollbar (horizontal scrollbar) 	ScrollBar	Internal Navigator Facet : (T: internal navigation D: _ Card: _ IsCentral: true)
Tdspace (vertical space) or Lrspace (horizontal space)	BlankSpace	Space Facet :_
Menubutton 	Menu	Action Item Facet: (T: activation D: _ Card: _ IsCentral: true)
Message 	Message	Text Display Facet: (T: consultation D: string Card: 1 IsCentral: true)
Numberentry 	SpinButton	Simple Number Chooser Facet: (T: selection D: integer Card: 1 IsCentral: true) Facet: (T: consultation D: interval of N integers Card: 1 IsCentral: false)
Panel 	TabbedDialogBox	Tabbed Navigator Facet: (T: navigation D: _ Card: _ IsCentral: true)

Annex C: An Interactor Model for the QtK Toolkit

Group of Radiobuttons 	Aggregation of N RadioButton	Simple Text Chooser Facet: (T: selection D: string Card: 1 IsCentral: true) Facet: (T: consultation D: string Card: N ergonomic rule: $N \leq 7$ IsCentral: false)
Tbbutton 	DrawButton	Action Item Facet: (T: activation D: _ Card: _ IsCentral: true)
Tbcheckboxbutton 	Combination of DrawButton + CheckBox	Action Item + Multiple Text Chooser Facet: (T: activation D: _ Card: _ IsCentral: true) Facet: (T: selection D: string Card: N IsCentral: true) Facet: (T: consultation D: string Card: N IsCentral: false)

Annex C: An Interactor Model for the QtK Toolkit

Tbradiobutton 	Combination of DrawButton + RadioButton	Action Item + Simple Text Chooser Facet: (T: activation D: _ Card: _ IsCentral: true) Facet: (T: selection D: string Card: 1 IsCentral: true) Facet: (T: consultation D: string Card: N IsCentral: false)
Text 	ExtendedEditBox	Text Inputter Facet: (T: specification D: string Card: 1 IsCentral: true) Facet: (T: consultation D: string Card: 1 IsCentral: false)
Td or Lr	Frame	Container + placement relationship Facet: _

Annex D. An overview of OCL

Use

OCL expressions are bound to a UML model. In the case of class diagrams, OCL can be used to specify invariants or pre- and post-conditions on operations

An OCL expression is always bound to a specific *context*, i.e. to a construct in the UML model. Each OCL expression begins with a context declaration.

Example: **context** Toolkit

Types

The building blocks for OCL expressions are *objects* and object *properties*.

Each object has a *type*. OCL types belong to one of these categories:

- predefined types: basic types (Boolean, Integer, Real, String) and collection types (Collection, Set, Bag, Sequence)
- user-defined types: all classifiers (type, class...) defined within a UML model have a corresponding type in OCL

OCL types are organized in a *type hierarchy*, which determines conformance of the different types to each other.

Properties

OCL expressions can also refer to an object's properties. A *property* can be an attribute, an association end or a method.

The value of an object's property is specified by a dot followed by the name of the property.

Example: **context** Toolkit

self.name

The keyword **self** is a reference to the context object, optional if the context is not ambiguous. The type of the subexpression **self.name** is the type of the attribute **name** (String). When the multiplicity of the attribute is greater than 1, the result type is a collection type.

Associated classes can be referred to by their rolename or, if the rolename is not present, by the classname starting with a lowercase letter.

Example: **context** Toolkit

self.finalWidget

Annex D: An overview of OCL

As the multiplicity of the association between the classes Toolkit and FinalWidgets is 1 to N, `self.finalWidget` will evaluate to a set of FinalWidgets.

Operations

OCL defines a number of operations on predefined types. For example:

- Boolean: operations and, or, not, if-then-else...
- Integer and Real: mathematical operations (+, -, * ...)

Operations are also defined on collection types, for instance:

- `includes(object)`: returns true if the object is an element of the collection
- `includesAll(collection)`: returns true if all elements of the parameter collection are present in the current collection
- `exists(expression)`: returns true if the expression is true for at least one element in the collection
- `forAll(expression)`: returns true if the expression is true for all elements of the collection
- `isEmpty`, `notEmpty`, `size`,...
- `select (expression)`: returns the elements for which the expression is true

Operations on collection types are accessed by using an arrow notation.

Example: **context** Toolkit

`self.finalWidget -> notEmpty`

The methods defined on the model types in a UML model can also be used in OCL, with one restriction: only the methods that return a value but have no side-effects are allowed. The dot notation used for attributes is also used to reference methods:

Example: **context** FinalWidget

`self.getDesiredLength()`

Some operations are defined for objects of every OCL type:

- `obj.oclIsTypeOf (type:OclType)`: evaluates to true if the type of the object is identical to the type of the argument
- `obj.oclIsKindOf (type:OclType)`: evaluates to true if the type of the object is identical to the type of the argument or to any of its subtypes

Pre- and post-conditions in OCL

As stated above, OCL can be used to specify pre- and post-conditions on operations and methods. The context of those pre- or post-conditions is always an operation or a method.

Annex D: An overview of OCL

Example: **context** FinalWidget::getDesiredLength():Integer
 post result = ...

In OCL expressions that are part of a post-condition, it is possible to refer to the value of a property before the operation, using the suffix **@pre**.

It is also possible to define extra operations on a class using the stereotype **<<oclOperation>>**. Such operations are used for specification purposes only, and do not need to be implemented.

Annex E. Description of the rules to be implemented in the GD tool

Panel 1: Resizing rules

Name of the rule :	FontSizeReduction
Label :	Change size fonts
Description :	Reduce font size to a given minimum
UsiXML description :	Decrease value of <code>textSize</code> attribute of <code>graphicalIndividualComponent</code> objects
Parameters :	s, the minimum size of fonts in the target UI. By default, s = 8 pts

Name of the rule :	InputFieldShrinkage
Label :	Resize visible length of text fields
Description :	Reduce the visible length of text fields, without reduction of the maximal length.
UsiXML description :	Decrease value of <code>numberOfColumns</code> attribute of <code>textComponent</code> objects, with constant value of the <code>maxLength</code> attribute
Parameters :	l, the visible length of text components in the target UI. The default value is 10 characters.

Name of the rule :	NumberOfVisibleListItemsDecrease
Label :	Decrease number of visible rows in lists and combo boxes
Description :	Reduce the number of items in list boxes and combo boxes that are

Annex E: Description of the rules to be implemented in the GD tool

	visible without scrolling.
UsiXML description :	Decrease value of <code>maxlineVisible</code> attributes of <code>comboBox</code> and <code>listBox</code> objects
Parameters :	<code>n</code> , the number of visible items in the target UI. The default value is 1

Panel 2: Moving rules

Name of the rule :	VerticalRepositioningInColumns
Label :	Align group boxes vertically
Description :	Vertical repositioning of the boxes structuring the source UI into one or several columns. The difference between the column's sizes must be minimized.
UsiXML description :	Input: a source <code>cuiModel</code> structured into boxes <code>b1, b2, ...bx</code> Output: the same <code>cuiModel</code> where all the boxes belonging to levels $< n$ are repositioned in one or several columns: (1) If $c=1$, <code>b1, b2, ...,bx</code> are inserted into a single vertical box, in such a way that each object that was positioned to the right of another object is now positioned above. (2) If $c>1$, the ordered list produced in (1) evenly distributed between <code>c</code> columns, so as to minimize the difference between the columns' height.
Parameters :	- <code>n</code>, the nesting level where the rule applies. The outermost box is of level 0, the boxes directly embedded in this level are of level 1, etc. By default, n = the innermost level (the rule applies to all boxes) - <code>c</code>, the number of columns on the target platform. The default value is 1

Name of the rule :	VerticalAlignmentOfGroupBoxContent
Label:	Align content of a group box vertically
Description :	Vertical repositioning of all the elements inside the selected boxes (without modification of the relative position of the boxes)

Annex E: Description of the rules to be implemented in the GD tool

UsiXML description :	Replace the value of the type attribute of the box object by “type=vertical”
Parameters :	–

Panel 3: Interactor transformations

Name of the rule :	InteractorSubstitution																																											
Label:	Interactor substitution																																											
Description :	Substitution of an interactor by another interactor supporting the same data type and the same functionalities.																																											
UsiXML description :	Replace a graphicalCIO (1) with another graphicalCIO reifying the same extendedAIO <table border="1"> <tr> <td>button</td><td>↔</td><td>menuItem Option : menu to which the menu item belongs (pre-existing menu or new one, with insertion of a menuBar if none is present)</td></tr> <tr> <td>comboBox</td><td>↔</td><td>group of radioButton</td></tr> <tr> <td>listBox with multiple selection = false</td><td>↔</td><td>group of radioButton</td></tr> <tr> <td>listBox with multiple selection = true</td><td>↔</td><td>group of checkBox</td></tr> <tr> <td>slider</td><td>↔</td><td>spin</td></tr> <tr> <td>tabbedDialogBox</td><td>↔</td><td>set of labels (with hyperlinks)</td></tr> </table> (2) or with another graphicalCIO linked to extendedAIO supporting a supertype of the original task type <table border="1"> <tr> <td>comboBox</td><td>→</td><td>textField</td></tr> <tr> <td>filePicker</td><td>→</td><td>textField</td></tr> <tr> <td>label with hyperlink ≠ void</td><td>→</td><td>button</td></tr> <tr> <td>slider</td><td>→</td><td>comboBox with multiple selection=false</td></tr> <tr> <td>slider</td><td>→</td><td>textField</td></tr> <tr> <td>spin</td><td>→</td><td>comboBox with multiple selection=false</td></tr> <tr> <td>spin</td><td>→</td><td>textField</td></tr> <tr> <td>tabbedDialogBox</td><td>→</td><td>set of buttons</td></tr> </table> (3) or with a set of graphicalCIO linked to extendedAIO		button	↔	menuItem Option : menu to which the menu item belongs (pre-existing menu or new one, with insertion of a menuBar if none is present)	comboBox	↔	group of radioButton	listBox with multiple selection = false	↔	group of radioButton	listBox with multiple selection = true	↔	group of checkBox	slider	↔	spin	tabbedDialogBox	↔	set of labels (with hyperlinks)	comboBox	→	textField	filePicker	→	textField	label with hyperlink ≠ void	→	button	slider	→	comboBox with multiple selection=false	slider	→	textField	spin	→	comboBox with multiple selection=false	spin	→	textField	tabbedDialogBox	→	set of buttons
button	↔	menuItem Option : menu to which the menu item belongs (pre-existing menu or new one, with insertion of a menuBar if none is present)																																										
comboBox	↔	group of radioButton																																										
listBox with multiple selection = false	↔	group of radioButton																																										
listBox with multiple selection = true	↔	group of checkBox																																										
slider	↔	spin																																										
tabbedDialogBox	↔	set of labels (with hyperlinks)																																										
comboBox	→	textField																																										
filePicker	→	textField																																										
label with hyperlink ≠ void	→	button																																										
slider	→	comboBox with multiple selection=false																																										
slider	→	textField																																										
spin	→	comboBox with multiple selection=false																																										
spin	→	textField																																										
tabbedDialogBox	→	set of buttons																																										

Annex E: Description of the rules to be implemented in the GD tool

	supporting data types whose aggregation corresponds to the original data type		
	colorPicker	↔	group of spins to select R-B-G values
	colorPicker	↔	group of sliders to select R-B-G values
	colorPicker	→	group of edit fields
	datePicker	↔	group of comboBoxes to select a year, a month and a day
	datePicker	↔	group of spins to select a year, a month and a day
	fontChooser	↔	2 comboBoxes to select a font type and a font size, 1 group of check boxes to select effects 1 combo box to select a colour
	hourPicker	↔	group of comboBoxes to select hours, minutes and seconds
	hourPicker	↔	group of spins to select hours, minutes and seconds
Parameters :	-		

Panel 4: Image transformations

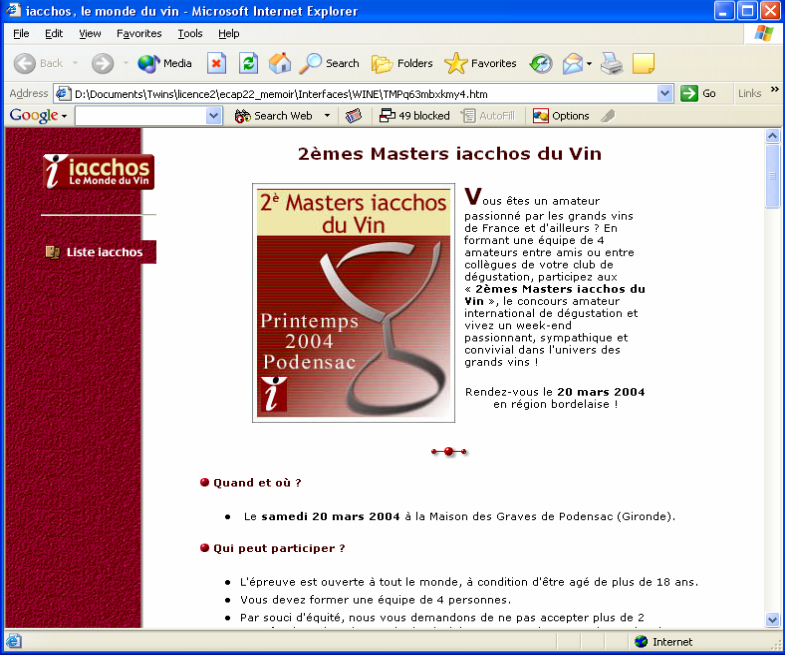
Name of the rule :	ReplaceImageByAlt
Label:	Replace images by their textual counterpart
Description :	Replace images by a textual description (in the case of Web sites, by the content of the <ALT> tag, if any)
Parameters :	–

Name of the rule :	ScaleAndCrop
Label:	Scale and crop images
Description :	Reduce images to their core subject by truncating their edges and minimize their size as much as possible.
Parameters :	–

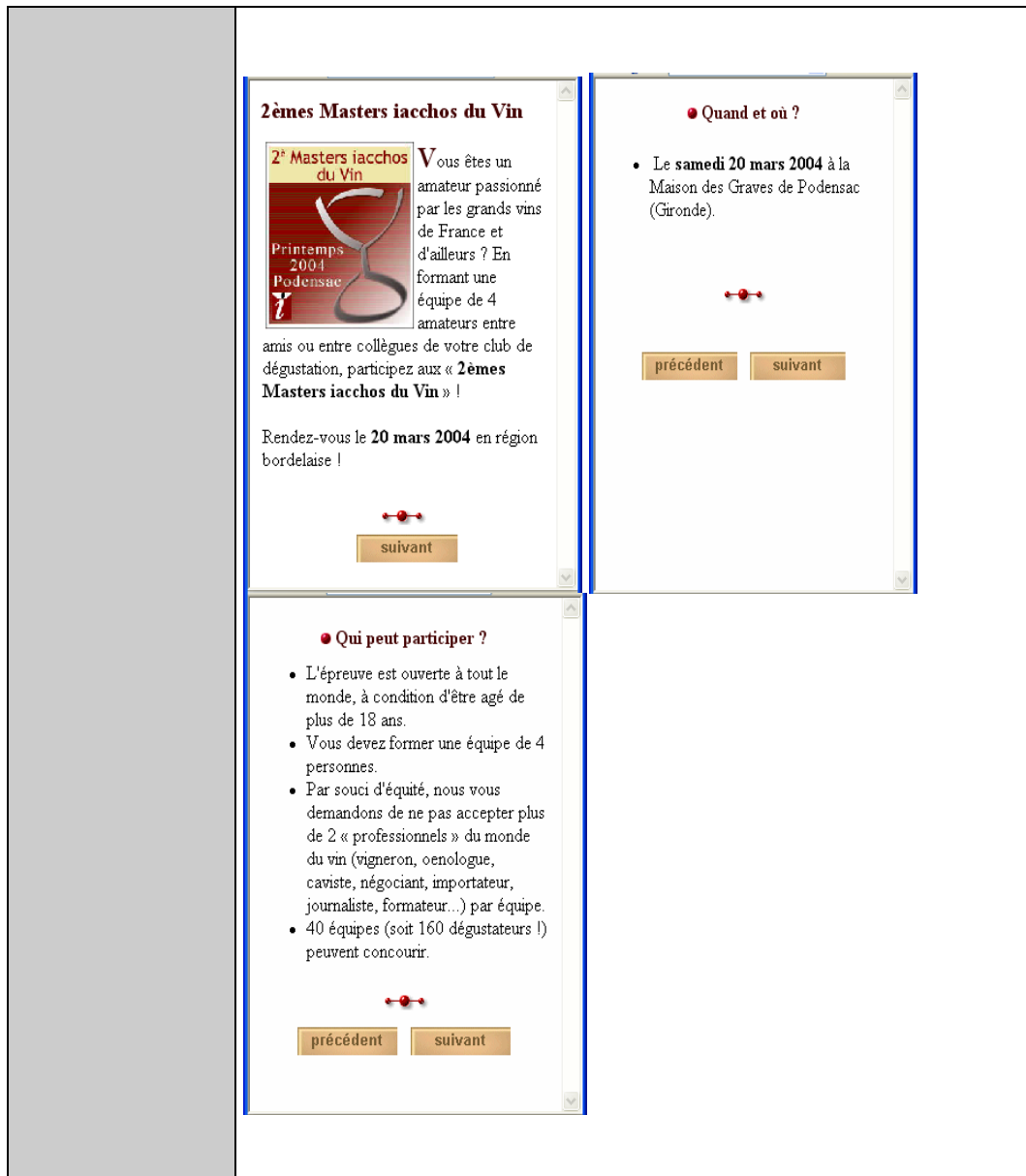
Panel 5: Splitting rules

Name of the rule:	InteractionSpaceSplittingWithLinearNavigation
Label :	Splitting of an interaction space with linear navigation
Description :	Distribution of the content of a source interaction space between

Annex E: Description of the rules to be implemented in the GD tool

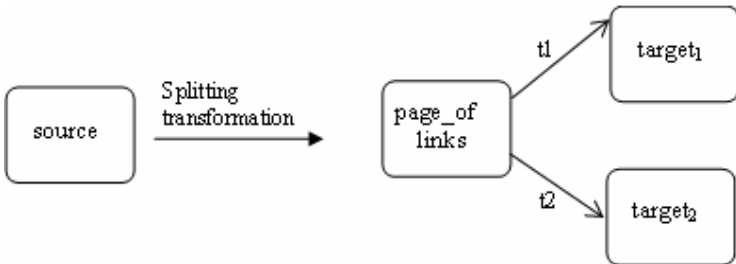
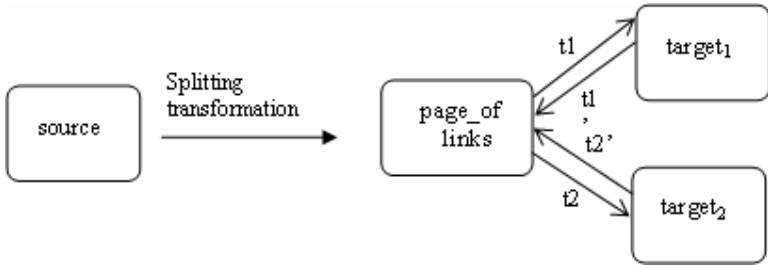
	two or more target interaction spaces, with : - Insertion into the dialog model of a transition between each pair of successive interaction spaces. - Insertion of the related navigation interactors in each target interaction space.  <i>Unidirectional linear navigation</i>  <i>Bidirectional linear navigation</i>
Parameters :	- t, the type of sequential navigation (uni- or bidirectional) - n, the number of interaction spaces at output. By default, n = the number of boxes.
Exemples :	 

Annex E: Description of the rules to be implemented in the GD tool

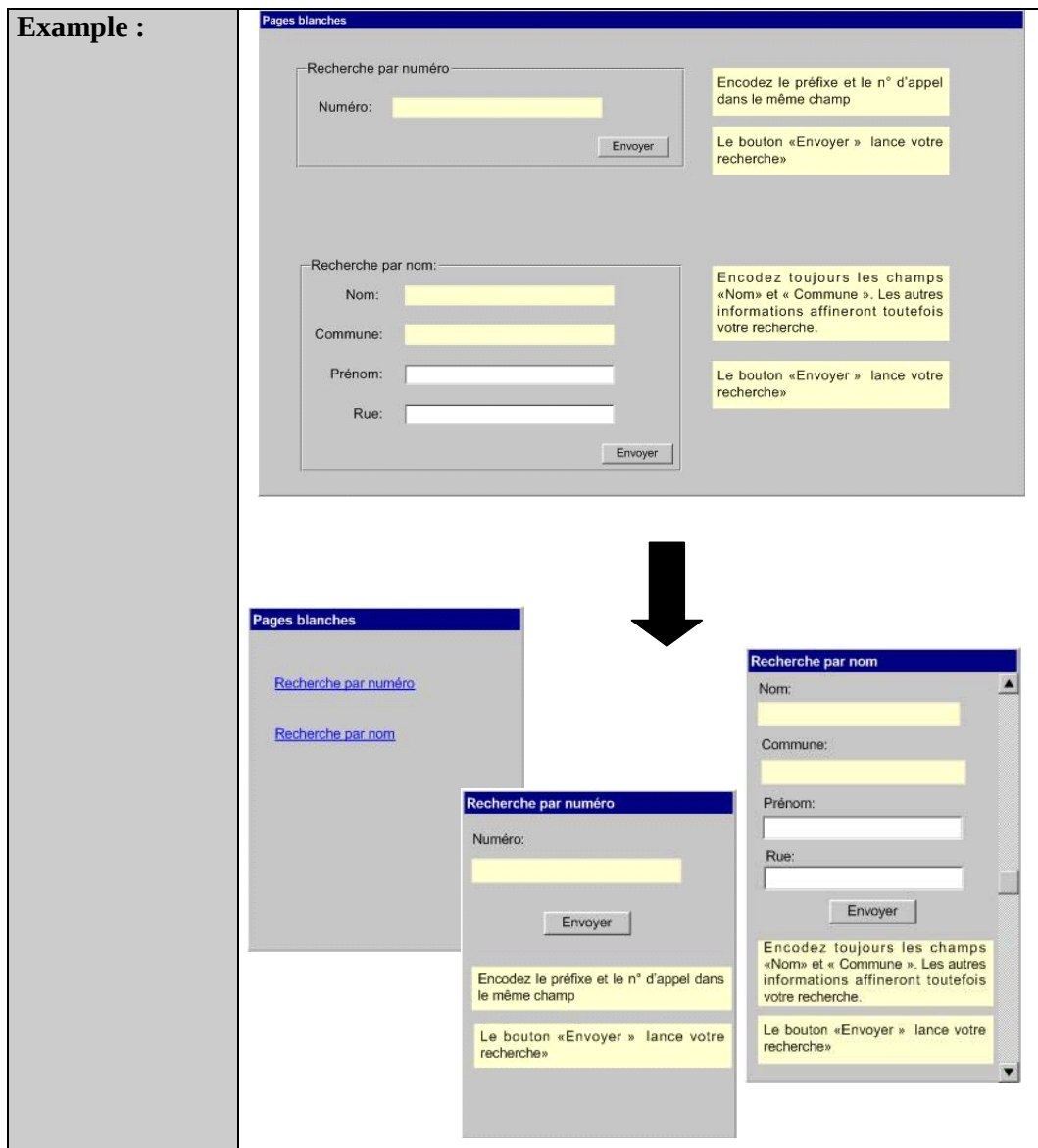


Name of the rule :	InteractionSpaceSplittingWithIndexedNavigation
Label:	Splitting of an interaction space with indexed navigation
Description :	Distribution of the content of a source interaction space between two or more target interaction spaces, with : - Insertion of an additional index page containing navigation

Annex E: Description of the rules to be implemented in the GD tool

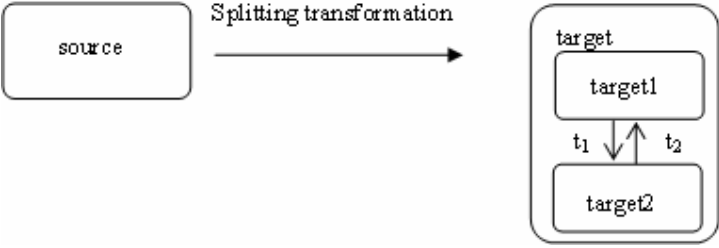
	interactors (hyperlinks...) allowing the transition to every target interaction space. - Insertion into the dialog model of the related transitions.  <i>Unidirectional indexed navigation</i>  <i>Bidirectional indexed navigation</i>
Parameters :	- t, the type of indexed navigation (uni- ou bidirectional) - n, the number of interaction spaces at output (index page not included). By default, n = the number of boxes - the names assigned to the target interaction spaces, which will also serve as labels for the interactors pointing from the index to these interaction spaces.

Annex E: Description of the rules to be implemented in the GD tool



Name of the rule :	InteractionSpaceSplittingWithIntraPageFullyConnectedNavigation
Label :	Splitting of an interaction space with fully-connected navigation
Description :	Distribution of the content of a source interaction space between two or more subsets (e.g ; panels of a tabbed panel) among which only one at the same time will be visible.

Annex E: Description of the rules to be implemented in the GD tool

	 <i>Fully-connected navigation</i>
Parameters :	- n, the number of interaction spaces at output. By default, n = the number of boxes - the names assigned to the target interaction spaces, which will also serve as labels for the tabs of the tabbed panel pointing to these interaction spaces.