Model Voyager: Visualization of CRF UI Models

Iyad Khaddam, Jean Vanderdonckt
Université catholique de Louvain (UCLouvain), Louvain Research Institute in Management and Organizations (LouRIM)
Place des Doyens, 1 – B-1348 Louvain-la-Neuve, Belgium, Belgium
{firstname.lastname}@uclouvain.be

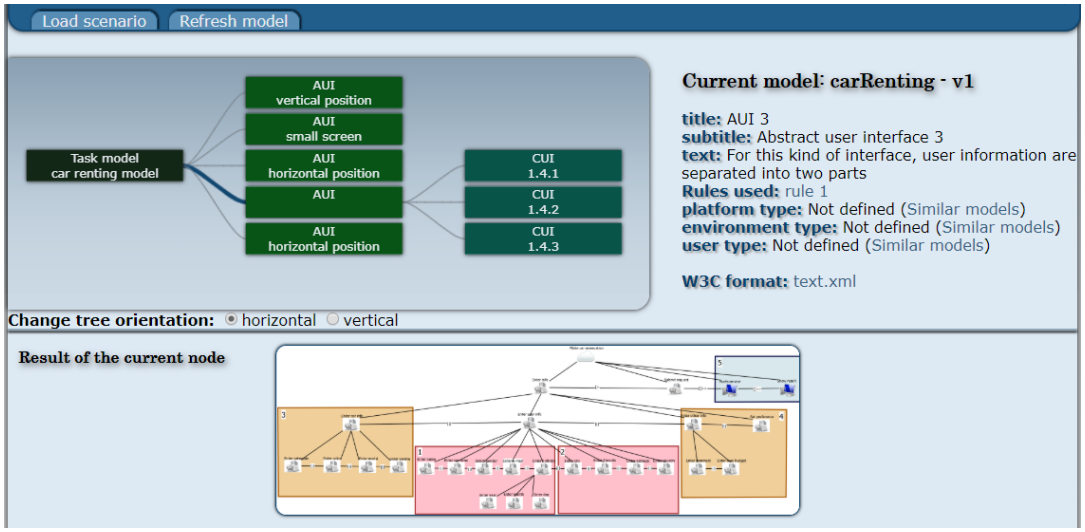

Figure 1: The Model Voyager on the "car rental" case study with hierarchy deployed.

ABSTRACT

This paper presents the Model Voyager, a web-based application for visualizing user interface models structured according to the four abstraction levels of the Cameleon Reference Framework: tasks and concepts, abstract user interface, concrete user interface, and final user interface. This application enables the designer to collect, edit, and manage collections of user interface models for a project or for maintaining an accessible catalogue of models, along with their illustrations. It also introduces three deployment mechanisms: multi-reification, multi-abstraction, and multi-translation. This paper demonstrates the applicability of the Model Voyager on the "car rental" case study, a reference example chosen by the W3C group on model-based user interfaces.

EICS '20 Companion, June 23–26, 2020, Sophia Antipolis, France

ACM ISBN 978-1-4503-7984-7/20/06…$15.00
https://doi.org/10.1145/3393672.3398492

CCS CONCEPTS

• **Human-centered computing → Graphical user interfaces**; *User studies*; *Empirical studies in interaction design*;

KEYWORDS

Cameleon Reference Framework; Design Space Exploration; Development path; Model-based user interface design; User interface models; User interface model visualization.

ACM Reference Format:
Iyad Khaddam, Jean Vanderdonckt. 2020. Model Voyager: Visualization of CRF UI Models. In *ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '20 Companion), June 23–26, 2020, Sophia Antipolis, France.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3393672.3398492

1 INTRODUCTION

Model-Based User Interface Design (MB-UID) [11] consists in applying a model-based approach in order to design or develop one or many [13] user interfaces (UIs) of an interactive system by involving various models such as, but not limited to: task model, domain model, user interface model, platform model, device model, user model, help model, etc. The context of use is considered as a key aspect [3] in context-sensitive user interfaces [2], typically represented by three underlying models: a user model capturing user profiles or

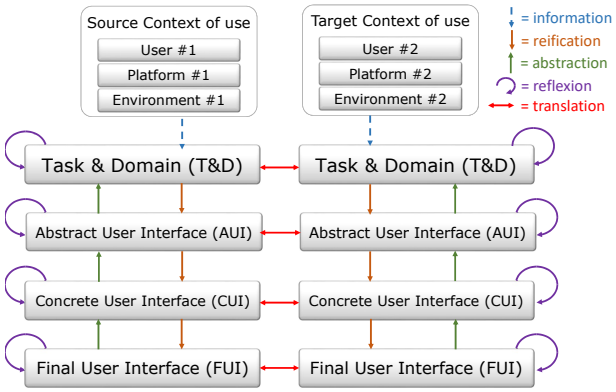

Figure 2: The Cameleon Reference Framework between two contexts of use.

clusters and related data (*e.g.*, interaction preferences, cognitive style), a platform model capturing essential parameters potentially influencing the UI (*e.g.*, screen resolution, interaction capabilities), and an environment model characterizing the physical and socio-organizational conditions where the user is carrying out her interactive tasks with the platform (*e.g.*, location, level of stress, structure). The Cameleon Reference Framework (CRF) [1] structures MB-UID according to four levels of abstraction with their corresponding models as follows [10]: task and domain concepts (T&D), abstract user interface (AUI), concrete user interface (CUI), and final user interface (FUI). Having defined one context of use or more, four types of relations are used to transition among these models [10] (Fig. 2): abstraction from one level to an upward level, reification from one level to a downward level, translation for a model at the same level of abstraction from one context of use to another, and reflexion regarding a single model at the same level of abstraction.

These examples suggest that the wide variety of models involved in MB-UID, along with the traversal of relations among them, leads to a problem of grasping the various development paths [8] and understanding them. In other areas of computer science, there are plenty of software for visualizing various models and the relations among them, including for mapping, transformation, association, etc. For instance, in database management systems, a domain model obtained at the conceptual level is transformed into a logical database model at the logical level and subsequently transformed into a physical database at the physical level. Software exists that support analysts in defining and editing the meta-model associated to these models [4], in establishing and visualizing transformations among these (meta-)models, or for visualizing relations among them [17].

While several software exist for domain modelling, business process modelling, database management, workflow management, there is no counterpart for UI models. In order to fill this gap, the contributions of this paper to the area of Engineering Interactive Computing Systems are as follows:

- Model Voyager, a Web-based application for managing CRF UI models of a project based on this structure.
- A mechanism for transversal navigation among these models following vertical and horizontal relations.
- A definition of multi-reification, multi-abstraction, and multi-translation as three new mechanism for exploring multipe UI alternatives from a single starting point.
- The illustration of the Model Voyager on the "car rental", a reference case study promoted by W3C [10].

2 RELATED WORK AND BACKGROUND

Starting from a source context of use (Fig. 2), itself represented as a user, a platform, and a corresponding environment, the CRF structures the UI development life cycle based on four levels and five relations, which are hereby summarized (see [10] for more details). When another UI needs to be developed for the same project, but for another target context of use, these levels and relations could be traversed to switch from the source context to the target context.

The *task model* is a representation of end users' interaction with the UI and is considered as one of the models characterizing its behavior. It identifies the different types of end users and their related tasks. It is created by the task analyst to demonstrate to the different stakeholders the benefits of the application.

The *abstract user interface* (AUI) allows having a rendering of what the task model and domain model will provide [9]. The AUI is independent of any platform and environment. The AUI is composed of different abstract interaction units (AIUs) and their relationships. Two AIU categories exist:

- *Abstract containers*: represent a group of related (sub-)tasks to be conveyed in the same container, such as in the same window (for an application) or web page (for a web site). Each container can be recursively decomposed into other containers or individual components.
- *Abstract individual components*: represent individual objects that compose containers falling into four categories:
  - Input: a user can input information into the software.
  - Output: an information for the user.
  - Navigation: a transition between two containers.
  - Control: a user can control various methods of the software. In addition to these components, we find in the AUI model some relationship between the different AIU composing the interface.

The *Concrete User Interface* (CUI) is based on the AUI and concretizes it on one hand and is an abstraction of the final user interface on the other hand. It uses Concrete Interaction Units (CIU) and Concrete User Interface Relationships (CUIR). CIUs are abstractions of concrete components. Concrete components examples include buttons, progress bars

and text boxes. CIUs are not dependent of a programming language, while concrete components are. CIUs are divided into two groups: containers and individual objects. The containers are graphical components (sometimes invisible) that contain both other containers or components. Individual CIU objects cannot contain other CIUs. They allow collecting information, manipulating, changing and adapting a given model. Each programming language defines its final individual objects. Based on the target language, we can translate the CIUs into the appropriate corresponding ones on the FUI level. There are also constraints on the CIUs, such as the position, transition, alignment and adjacency. Actions can be attached to CIUs in order to generate on the FUI level.

The *final user interface* (FUI) is the last step of the forward engineering. It is the model with the lowest level of abstraction. It consists in designing the concrete model of the GUI into the chosen language of the platform [16]. At this level, the platform, the user, and the environment are all determined [9]. Still, we may take some decisions on the final design. The CRF levels are linked together by four relations:

- *Reification*: is an operation where a model is transformed into a model on a lower level of abstraction. This operation can be repeated until the FUI is reached.
- *Abstraction*: is an operation where a model is transformed into a model on a higher level of abstraction.
- *Translation*: is an operation where a model is transformed to another model at the same level of abstraction, but for a different context of use. For instance, multi-target user interfaces [1] are obtained by translation: each time another context of use is considered, a translation from the source context to the target context is triggered.
- *Reflection*: is an operation where a model is transformed into another model at the same level of abstraction, for the same context of use.

Since MB-UID could involve all these models and relations between them, many development paths are possible [8], thus making it complex and challenging to manage as they are not represented together. Some software attempt to partially address this problem. For example, Quill [5] is a web based development environment displaying multiple models at the same time. Stakeholders collaboratively work on the UI design following a forward engineering MB-UID. Quill visualize any involved model and relation by employing spring-based visualization techniques to represent, manipulate, and map the models and to enhance the navigability among them. For example, when a model fragment is identified, its decomposition is progressively revealed via hyperbolic trees that are expanded and collapsed depending on space available.

The Model Viewer (MoVi) [6, 7] visualizes multiple CRF models and facilitates the navigation among them. It helps in maintaining a global view of the models inside a project, namely by propagating a change performed at one level to its subsequent levels. Similarly, IdealXML [14] enables the designer to textually create, delete, and edit relations among the CRF UI models in a table. While these relations could be indeed managed, there is no graphical representation.

Gef3D [17] visualizes links between a conceptual view (UML Class diagram), an external view (a structured UI), and an external view (the final UI), thus enabling the end user to better understand how these models and their components are linked to each other, to edit them, and to see the impact. Voyager 2 [4] provides a complete environment for defining meta-meta-models, meta-models, and models governed by them, as well as transformations at these three levels of abstraction. This environment mainly focuses on the domain model and does not cover UI models. Mega-UI [15] is an environment helping analysts to manage all model perspectives (*i.e.*, model, meta-model, transformation) at various CRF levels. It allows to navigate between components in the same model and navigate from one model to another model only in the same system. It does not present the external models from another system that can contribute to the whole design.

Therefore, we observe that available tools are mainly aimed at editing CRF UI models in order to support the UI development life cycle, which is normal. There is no genuine software for visualizing the various models and their relations for a single project, for one or multiple contexts of use, which is precisely the goal of this work.

3 MODEL VOYAGER

The Model Voyager[a] consists of a web application enabling any user to create a new project and to feed it with any CRF UI model required to support its development life cycle and for establishing relations among them. For each model, the visitor can download the model information in an XML format. A graphical schema is displayed on the web site and can be downloaded as well. If the FUI model source code is available, it can be downloaded too. The Model Voyager provides two main functionalities for visitors:

- Display model information: any model is in the Model Voyager according to simple data structure that is automatically displayed when selected.
- Navigate through levels: models involved in a single project are displayed in a dynamic tree that can be expanded and collapsed depending on model navigation.

Displaying Model Information

The Model Voyager stores a project or case study as a hierarchy of directories, which can be recursively decomposed into sub-directories. Each directory contains one CRF model stored in a info.xml file with the following descriptive data:

[a]Freely accessible at https://sites.uclouvain.be/mbui/index.php?t=carRenting/v1

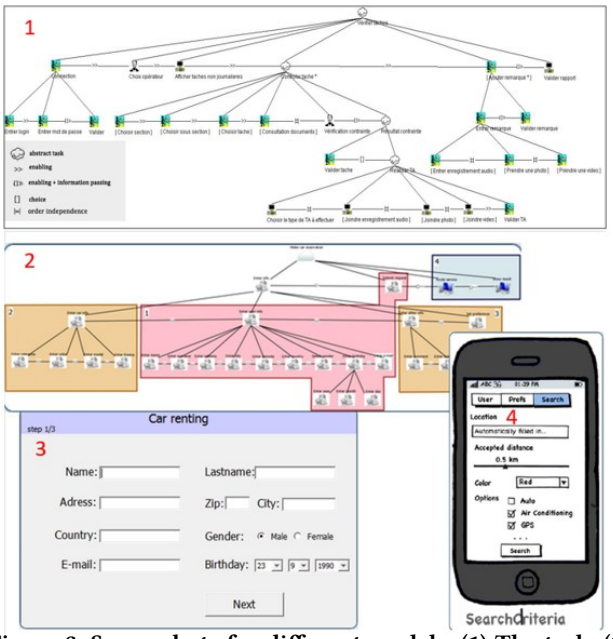

Figure 3: Screenshots for different models: (1) The task, (2) the AUI containers, (3) the CUI and (4) the FUI.

- title: the title of the current node corresponding to one CRF level. For example, "Task model for renting a car".
- subtitle: a more detailed header than the title. For example, "This task model represents the expected way to rent a car via the company".
- text: the explanation of the current node. For example, a textual description of a scenario covered by the task.
- rules used: the different rules used to achieve this model, if any. Each rule can be expressed in natural language, structured query language, or via a link to the web site where all transformation rules have been defined. For example, a set of rules used for deriving an AUI from the "car rental" task model are: change of the size of the interface (sourceInterface.size > outputInterface.size), groups different AIUs by category (*e.g.*, personal information, car information, additional information, and result), decompose the "Enter birthdate" task into three sub-tasks: "Enter year", "Enter month", and "Enter day").
- user type: link to user model in force at this level. For example, link to "Tourist model".
- platform type: link to the platform model in force at this level. For example, link to "Smartphone model".
- environment type: link to the environment model in force at this level. For example, link to "Stressing place".

In addition to the info.xml file, we can add screenshots for the model. We can also add a URL to any external representation of the model. For example, Fig. 3 contains: one for a task model, one for the AUI containers (on the task model), one for the CUI and finally one for the FUI.

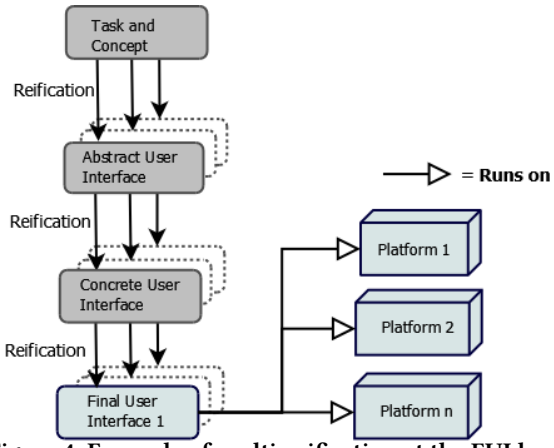

Figure 4: Example of multi-reification at the FUI level.

Navigate through Levels

Forward engineering in MB-UID basically exploits the reification relation from one level to any subsequent level: task to AUI, AUI to CUI, and CUI to FUI. Since this process can be repeated at any level, it gives rise to a tree structure, where each node holds a particular CRF model derived from the previous one. Each tree node holds a particular CRF model stored via its INFO.XML and IMAGE.PNG files: when a node is selected, the dynamic tree expands the sub-nodes to enable the visitor to drill down the models. When this node is selected again, the dynamic tree collapses its sub-graph to let the visitor moving up and down through the tree quickly, while being informed on the current position. When a particular node is activated, detailed information on the model in this node are displayed.

Implementation

To realize this dynamic tree, a PHP script parses the hierarchy of (sub-)directories and its contained files to automatically generate a JSON file used as input to the dynamic tree, rendered by the JavaScript InfoVis Toolkit[b]. Adding a new model at any level simply consists in adding a new directory at the right place in the hierarchy. The Model Voyager updates its visualization by re-running the script each time this structure is modified. When a node is selected, a JavaScript executes an Ajax request on info.xml to display its contents and establish the links following the relations.

Finding Similar Models

For the three types user, platform, and environment, the Model Voyager displays the link "Similar models" next to it to find similar models with related contexts of use. This link opens a new window (Fig. 6-c): the visitor browses different variations of the model per context of use. Similarity is defined dynamically, based on existing models within the tool. When two variations co-exist for the same context of use,

[b]See https://philogb.github.io/jit/

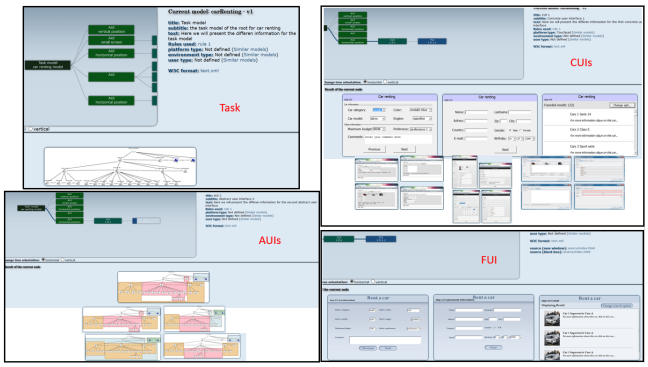

Figure 5: The "Car rental" models in the Model Voyager.

a drop-down list filters the alternatives by context of use. When multi-target UIs are devised in a project, reification is repeated several times from a same starting point. In order to reduce this repetition for all alternatives, we define a new notion aimed at defining multi-target UIs at once: a *multi-reification* consists in applying a synchronized reification at a given CRF level of abstraction for the same project, but for different contexts of use, and to semantically link them. For example, Fig. 4 graphically depicts a multi-reification performed at the CUI level for different target platforms: instead of creating a single reification for each alternative, the designer specifies the desired variations of the context of use (here, three) and applies a multi-reification at once. A multi-reification can be applied at any non-terminal CRF level. Similarly, a *multi-abstraction* consists in applying a synchronized abstraction at a given CRF level of abstraction for the same project, but for different contexts of use. A multi-abstraction can be applied at any non-initial CRF level. A multi-translation applies a synchronized translation at any level for many targets. Model Voyager supports multi-reification by allowing to navigate among these models once the parent model is activated.

Scenarios

The Model Voyager browses the directory of one project or case study at a time, which we call the scenario directory. One can visit another scenario by clicking on the "Load scenario" link on the top-left corner of the main page (Fig. 6-d).

It is possible to add anew scenarios but a user account on the server is required. Users can collaborate by enriching the content of a scenario with different models, and thus collaborate to foster the exploration of the design space. Instructions for this purpose are publicly available[c].

4 A CASE STUDY: THE CAR RENTAL

Among others, the "car rental case study is a reference case study studied in the frame of the W3C Charter Group on MBUID[d]. We will therefore consider it as a reference example to illustrate the capabilities of the Model Voyager. Particularly useful are also those navigations offered by the Model Voyager to traverse various levels of abstraction or at the same level of abstraction, but also to travel along dimensions of the context of use, and thus answering the question: if we maintain the same CRF level of abstraction, what is the consequence of changing the user model, the platform and/or the environment?.

In the Model Voyager, one chooses the "Car Rental" scenario and the version to visit. The first stop in the walk-through is the task model. The task model is displayed in the appropriate section (Fig. 5-Task), using the CTTE notation [12]. On the right side, supplementary information on the task model is displayed. To see how the task model changes on a different platform, the link "Similar Model" should be selected on the same line of "platform type". This will open variations of the task model for different platforms. The same process is valid for the user and the environment. Activating the task model node opens the sub-nodes for the AUIs (Fig. 5-AUIs). In this scenario, five nodes are available representing five AUI variations: two for vertical layouts for smartphones, one for horizontal layout for a landscape tablet, one for small screens and finally a generic one. Selecting any node shows how the distribution of tasks in terms on AIUs is made. On the right side, supplementary information on the AUI model is displayed. Other variations of the AUI model based on the current context of use or for another one are accessible by browsing the "Similar models" link.

Activating an AUI model node opens the sub-nodes for the CUIs (Fig. 5-CUIs). For instance, clicking on the generic AUI node displays the three variations of CUI models. Clicking on any of the CUI model nodes deploys the CUI containers. One remarkable difference is the existence of several CUI models: at the CUI level, we are getting closer to the final UI and the widgets start to take their shapes. On the right side, supplementary information on the CUI model is displayed. The same processes applies for the CUI model concerning adaptation of the model to any context of use.

Activating a CUI model node opens the sub-nodes for the FUIs. For instance, click on the AUI horizontal position, then the CUI models, a node for the FUI appears (Fig. 5-FUI). Selecting a FUI node shows how a CUI is reified into a FUI.

5 CONCLUSION

We presented the Model voyager, a web application managing and visualizing UI models and relations structured according to the Cameleon Reference Framework (CRF). We explained the functionalities of the software, the models presented, and information on these models.

We demonstrated the use of the tool to browse a model through the "Car rental" case study. The goal of Model Voyager is to allow visitors to visualize UI models according to

[c]See http://sites.uclouvain.be/mbui/guide.html
[d]See use case UC1 in https://www.w3.org/TR/mbui-intro/

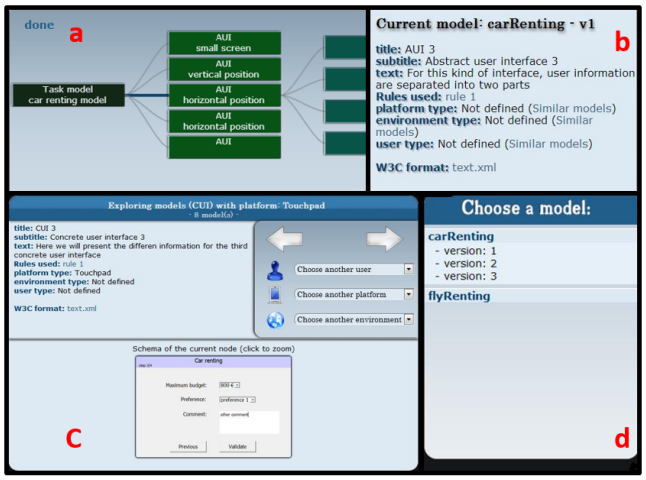

Figure 6: The tool screens: (a) the navigator, (b) model information, (c) similar models and (d) model selection.

the four levels of the CRF. Reification and abstraction relations are supported in the dynamic tree, while reflexion and translation are supported through the "Similar models" link.

The main contributed value of Model Voyager is to visually guide the visitor among the multiple CRF UI models, to explore design alternatives at any level. This approach fosters a design space exploration that could increase the observability and browsability of a same case study, especially when considering different contexts of use. This design space exploration is expected to become useful for practitioners like analysts, designers, and developers who are responsible for managing several UI models, but also for teachers to facilitate understanding the usage of the CRF by students.

We intend to turn Model Voyager into a shared reference for the MB-UID community at large. We are looking after more case studies and contents. This is an open invitation to all interested parties to contact us in order to create accounts for them to share their contents or to ask us to install an instance of the tool on their own server.

REFERENCES

[1] Gaëlle Calvary, Joëlle Coutaz, David Thevenin, Quentin Limbourg, Laurent Bouillon, and Jean Vanderdonckt. 2003. A Unifying Reference Framework for multi-target user interfaces. *Interacting with Computers* 15, 3 (6 2003), 289–308. https://doi.org/10.1016/S0953-5438(03)00010-9

[2] José Creissac Campos and F. Martins. 1996. Context Sensitive User Interfaces. In *Formal Aspects of the Human Computer Interface (*electronic *Workshops in Computing)*, C. Roast and J. Siddiqi (Eds.). Springer-Verlag London, Sheffield Hallam University, UK. http://www.springer.co.uk/eWiC/Workshops/FACHI/

[3] Joëlle Coutaz, James L. Crowley, Simon Dobson, and David Garlan. 2005. Context is Key. *Commun. ACM* 48, 3 (March 2005), 49–53. https://doi.org/10.1145/1047671.1047703

[4] Vincent Englebert and Jean-Luc Hainaut. 1999. DB-MAIN: A Next Generation Meta-CASE. *Inf. Syst.* 24, 2 (1999), 99–112. https://doi.org/10.1016/S0306-4379(99)00007-1

[5] Vivian Genaro Motti, Dave Raggett, Sascha Van Cauwelaert, and Jean Vanderdonckt. 2013. Simplifying the Development of Cross-platform Web User Interfaces by Collaborative Model-based Design. In *Proceedings of the 31st ACM International Conference on Design of Communication* (September 30-October 01, 2013) *(SIGDOC '13)*. ACM, New York, NY, USA, 55–64. https://doi.org/10.1145/2507065.2507067

[6] Mufida Mir'atul Khusna, Gaëlle Calvary, Sophie Dupuy-Chessa, and Yann Laurillau. 2015. MoVi: models visualization for mastering complexity in model driven engineering. In *Proceedings of the 2015 British HCI Conference, Lincoln, United Kingdom, July 13-17, 2015*, Shaun W. Lawson and Patrick Dickinson (Eds.). ACM, 281–282. https://doi.org/10.1145/2783446.2783611

[7] Mufida Mir'atul Khusna, Sophie Dupuy-Chessa, and Gaëlle Calvary. 2016. Mastering Model Driven Engineering complexity by interactive visualization. *Technique et Science Informatiques* 35, 2 (2016), 175–202. https://doi.org/10.3166/tsi.35.175-202

[8] Quentin Limbourg and Jean Vanderdonckt. 2009. Multipath Transformational Development of User Interfaces with Graph Transformations. In *Human-Centered Software Engineering - Software Engineering Models, Patterns and Architectures for HCI*, Ahmed Seffah, Jean Vanderdonckt, and Michel C. Desmarais (Eds.). Springer, 107–138. https://doi.org/10.1007/978-1-84800-907-3_6

[9] Quentin Limbourg, Jean Vanderdonckt, Benjamin Michotte, Laurent Bouillon, and Murielle Florins. 2004. USIXML: A User Interface Description Language Supporting Multiple Levels of Independence. In *Proceedings of Workshops in connection with the 4th International Conference on Web Engineering (ICWE '04). Engineering Advanced Web Applications* (28-30 July, 2004) *(DIWE '04)*, Maristella Matera and Sara Comai (Eds.). Rinton Press, 325–338.

[10] Gerrit Meixner, Gaëlle Calvary, and Joëlle Coutaz. 2014. Introduction to Model-Based User Interfaces - W3C Working Group Note. https://www.w3.org/TR/mbui-intro/

[11] Gerrit Meixner, Fabio Paternò, and Jean Vanderdonckt. 2011. Past, Present, and Future of Model-Based User Interface Development. *i-com Zeitschrift für interaktive und kooperative Medien* 10, 3 (2011), 2–11. https://doi.org/10.1524/icom.2011.0026

[12] G. Mori, F. Paterno, and C. Santoro. 2002. CTTE: support for developing and analyzing task models for interactive system design. *IEEE Transactions on Software Engineering* 28, 8 (2002), 797–813.

[13] Fabio Paternò and Carmen Santoro. 2002. One Model, Many Interfaces. In *Computer-Aided Design of User Interfaces III, Proceedings of the Fourth International Conference on Computer-Aided Design of User Interfaces, May, 15-17, 2002, Valenciennes, France*, Christophe Kolski and Jean Vanderdonckt (Eds.). Kluwer, 143–154.

[14] Francisco Montero Simarro and Víctor López-Jaquero. 2006. IdealXml: An Interaction Design Tool. In *Computer-Aided Design Of User Interfaces V, Proceedings of the Sixth International Conference on Computer-Aided Design of User Interfaces, CADUI 2006 6-8 June 2006, Bucharest, Romania*, Gaëlle Calvary, Costin Pribeanu, Giuseppe Santucci, and Jean Vanderdonckt (Eds.). Springer, 245–252. https://doi.org/10.1007/978-1-4020-5820-2_20

[15] Jean-Sébastien Sottet, Gaelle Calvary, Jean-Marie Favre, and Jöelle Coutaz. 2009. *Megamodeling and Metamodel-Driven Engineering for Plastic User Interfaces: MEGA-UI.* Springer London, London, 173–200. https://doi.org/10.1007/978-1-84800-907-3_8

[16] Jean Vanderdonckt. 2005. A MDA-Compliant Environment for Developing User Interfaces of Information Systems. In *Advanced Information Systems Engineering*, Oscar Pastor and João Falcão e Cunha (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 16–31.

[17] Jens von Pilgrim and Kristian Duske. 2008. Gef3D: A Framework for Two-, Two-and-a-Half-, and Three-Dimensional Graphical Editors. In *Proceedings of the 4th ACM Symposium on Software Visualization (SoftVis '08)*. Association for Computing Machinery, New York, NY, USA, 95–104. https://doi.org/10.1145/1409720.1409737