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

edited by

Gaëlle Calvary
Université Grenoble 1
Grenoble, France

Costin Pribeanu
National Institute of Research & Development in Informatics
Bucharest, Romania

Giuseppe Santucci
Università degli Studi di Roma “La Sapienza”
Roma, Italy

and

Jean Vanderdonckt
Université catholique de Louvain
Louvain-la-Neuve, Belgium

Springer
# TABLE OF CONTENTS

*Sponsors* ................................................................................................................................. vii  
*Programme Committee Members* ............................................................................................... ix  
*Acknowledgements* .................................................................................................................. xi  

## Keynote Paper

1. Generating User Interfaces from Conceptual Models:  
   A Model-Transformation Based Approach ............................................................. 1  
   Ó. Pastor

## Multimodal User Interfaces

2. Towards Object Oriented, UIML-based Interface Descriptions for Mobile Devices .......................... 15  
   R. Schaefer and S. Bleul

3. Towards a System of Patterns for the Design of Multimodal Interfaces ......................................... 27  
   G. Godet-Bar, S. Dupuy-Chessa, and L. Nigay

4. Design Options for Multimodal Web Applications ................................................................. 41  
   A. Stanciulescu and J. Vanderdonckt

5. A Generic Approach for Pen-based User Interface Development ............................................. 57  
   S. Macé and E. Anquetil

## Virtual and Mixed Reality User Interfaces

6. Participatory Design Meets Mixed Reality Design Models: Implementation Based on a Formal Instrumentation of an Informal Design Approach .......................................................... 71  
   E. Dubois, G. Gauffre, C. Bach, and P. Salembier

7. A Method for Developing 3D User Interfaces of Information Systems .................................. 85  
   J.M. González Calleros, J. Vanderdonckt, and J. Muñoz Arteaga

   D. Lingrand, Ph. Renevier, A.-M. Pinna-Déry, X. Cremaschi, S. Lion, J.-G. Rouel, D. Jeanne, Ph. Cuisinaud, and J. Soula

## User Interfaces for Multi-Device Environments

9. Designing and Developing Multi-user, Multi-device Web Interfaces .................................. 111  
   F. Paternò and I. Santos

    M.M. Silva and E. Furtado

11. Transformational Consistency ................................................. 137  
    K. Richter
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Authors</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>Rapid Prototyping of Distributed User Interfaces</td>
<td>J.P. Molina Massó, J. Vanderdonckt, P. González López, A. Fernández-Caballero, and M.D. Lozano Pérez</td>
<td>151</td>
</tr>
<tr>
<td>14</td>
<td>A Generic Approach for Multi-device User Interface Rendering with UIML</td>
<td>K. Luyten, K. Thys, J. Vermeulen, and K. Coninx</td>
<td>175</td>
</tr>
<tr>
<td>15</td>
<td>Device Independent Layout and Style Editing Using Multi-level Style Sheets</td>
<td>W. Dees</td>
<td>183</td>
</tr>
<tr>
<td>16</td>
<td>Automatic Interface Generation through Interaction, Users, and Devices Modeling</td>
<td>E. Bertini, G. Santucci, and A. Cali</td>
<td>191</td>
</tr>
<tr>
<td>17</td>
<td>The Meta Sketch Editor: A Reflexive Modeling Editor</td>
<td>L. Nóbrega, N.J. Nunes, and H. Coelho</td>
<td>201</td>
</tr>
<tr>
<td>18</td>
<td>A Hybrid Tool for User Interface Modeling and Prototyping</td>
<td>H. Trætteberg</td>
<td>215</td>
</tr>
<tr>
<td>19</td>
<td>Towards a Support of User Interface Design by Composition Rules</td>
<td>S. Lepreux and J. Vanderdonckt</td>
<td>231</td>
</tr>
<tr>
<td>21</td>
<td>Integrating Model-based and Task-based Approaches to User Interface Generation</td>
<td>S. España, I. Pederiva, and J.I. Panach</td>
<td>253</td>
</tr>
<tr>
<td>22</td>
<td>Automated Repair Tool for Usability and Accessibility of Web Sites</td>
<td>A. Jasselette, M. Keita, M. Noirhomme-Fraiture, F. Randolet, J. Vanderdonckt, Ch. Van Brussel, and D. Grolaux</td>
<td>261</td>
</tr>
<tr>
<td>23</td>
<td>Automating Guidelines Inspection: From Web Site Specification to Deployment</td>
<td>J. Xiong, M. Diouf, Ch. Farenc, and M. Winckler</td>
<td>273</td>
</tr>
</tbody>
</table>
SPONSORS

Official CADUI Web site

Computer-Aided Design of User Interfaces
http://www.isys.ucl.ac.be/bchi/cadui

Corporate Sponsors

CARE Technologies
http://www.care-t.com

DefiMedia
http://www.defimedia.be

IBM Belgium
http://www.ibm.com/be

OlivaNova Model Execution Software

RedWhale Software Corporation
http://www.redwhale.com

SAP AG
http://www.sap.com

Scientific Sponsors

ACM Special Interest Group on Computer-Human Interaction (SIGCHI)
http://www.sigchi.org

ACM SIGCHI Belgian Chapter (BelCHI)
http://www.belchi.be

ACM SIGCHI Romanian Chapter (RoCHI)
http://www.ici.ro/chi-romania

Association Francophone d’Interaction Homme-Machine (AFIHM)
http://www.afihm.org

Belgian Laboratory of Computer-Human Interaction (BCHI)
http://www.isys.ucl.ac.be/bchi
COST Action n°294 – MAUSE: Towards the MAturation of Information Technology USability Evaluation
http://www.cost294.org

European Cooperation in the field of Scientific and Technical Research (COST)
http://www.cost.esf.org

European Science Foundation (ESF)
http://www.esf.org

Institut d’Administration et de Gestion
http://www.uclouvain.be/iag

National Institute for Research and Development in Informatics (ICI)
http://www.ici.ro

The Open Interface Foundation
http://www.openinterface.org

SIMILAR: The European research taskforce creating human-machine interfaces SIMILAR to human-human communication
http://www.similar.cc

Sixth Framework Program, European Commission
http://cordis.europa.eu/fp6

Université catholique de Louvain
http://www.uclouvain.be

Université Grenoble 1, Equipe IIHM
http://iihm.imag.fr/site

Università’ degli Studi di Roma “La Sapienza”
http://www.uniroma1.it

UsiXML (USer Interface eXtensible Markup Language) Consortium
http://www.usixml.org
PROGRAMME COMMITTEE MEMBERS

Silvia Abrahao, Polytechnic University of Valencia, Spain
Sabin-Corneliu Buraga, UAIC Iasi, Romania
Gaëlle Calvary, Université Grenoble 1, France
Joëlle Coutaz, Université Grenoble 1, France
Olga De Troyer, Vrije Universiteit Brussel, Belgium
Emmanuel Dubois, IRIT Toulouse, France
Antonio Fernández-Caballero, University of Castilla-La Mancha, Spain
Peter Forbrig, University of Rostock, Germany
Elizabeth Furtado, University of Fortaleza, Brazil
Pascual Gonzalez, University of Castilla-La Mancha, Spain
Philip Gray, University of Glasgow, UK
James Helms, Harmony, Inc., USA
Geert-Jan Houben, Vrije Universiteit Brussel, Belgium
Robert J.K. Jacob, Tufts University, USA
Anthony Jameson, DFKI, Germany
Herman Kaindl, Vienna University of Technology, Austria
Nora Koch, University of Munich, Germany
Christophe Kolski, Université de Valenciennes, France
James Lin, IBM Almaden, USA
Victor López-Jaquero, University of Castilla-La Mancha, Spain
Jesús Lorés, University of Lleida, Spain
Maria-Dolorès Lozano, University of Castilla-La Mancha, Spain
Kris Luyten, University of Hasselt, Belgium
Mark T. Maybury, The Mitre Corp., USA
Francisco Montero, University of Castilla-La Mancha, Spain
Laurence Nigay, Université Grenoble 1, France
Erik G. Nilsson, SINTEF, Norway
Monique Noirhomme, Facultés Universitaires N.-D. de la Paix, Belgium
Nuno J. Nunes, University of Madeira, Portugal
Luis Olsina, University Nacional de La Pampa, Argentina
Philippe Palanque, IRIT Toulouse, France
Fabio Paternò, ISTI-CNR, Italy
Paolo Pinheiro da Silva, Stanford University, USA
Horia D. Pitariu, UBB Cluj-Napoca, Romania
Costin Priebeanu, ICI, Romania
Angel Puerta, RedWhale Software Corp., USA
Kai Richter, Computer Graphics Center (ZGDV), Germany
Giuseppe Santucci, Universita’ degli Studi di Roma “La Sapienza”, Italy
Kevin A. Schneider, University of Saskatchewan, Canada
Daniel Schwabe, PUC Rio de Janeiro, Brazil
Ahmed Seffah, Concordia University, Canada
Constantine Stephanidis, ICS-Forth, Greece
Piyawadee Sukaviriya, IBM Thomas Watson Research Center, USA
Gerd Szwillus, University of Paderborn, Germany
Hallvard Trætteberg, Norwegian University of Science and Technology, Norway
Jean Vanderdonckt, Université catholique de Louvain, Belgium
Thomas Ziegert, SAP Research, Germany
ACKNOWLEDGEMENTS

The editors would like to particularly thank the authors, the reviewers, the participants, and the sponsors for their support of the CADUI’2006 edition. We also warmly thank Prof. O. Pastor for giving the keynote address of this conference.

The editors would like to thank Cristian Voicu, who significantly helped in the preparation and the editing of the final version of this book, and Teodora Voicu for her support to the conference participants. This book was supported by the SIMILAR network of excellence, the European research taskforce creating human-machine interfaces SIMILAR to human-human communication (www.similar.cc), funded by the 6th Framework Program, IST Program of the European Commission, under contract FP6-IST1-2003-507609. The editors are also grateful to Juan Manual Gonzalez Calleros for the two first images on the cover and José Pascual Molina for the picture of the multi-monitor display.
Chapter 1

GENERATING USER INTERFACES FROM CONCEPTUAL MODELS: A MODEL-TRANSFORMATION BASED APPROACH

Óscar Pastor
Department of Information Systems and Computation, Valencia University of Technology, Camino de Vera s/n, 46071 Valencia (Spain)
E-Mail: opastor@dsic.upv.es – Web: http://oomethod.dsic.upv.es/anonimo/memcontactinfo.aspx?idMember=2
Tel.: +34- 963877794 – Fax: +34- 963877359

Abstract Traditionally, the Software Engineering community has been interested in defining methods and processes to develop software by specifying its data and behavior disregarding user interaction. On the other hand, the Human-Computer Interaction community has defined techniques oriented to the modeling of the interaction between the user and the system, proposing a user-oriented software construction. This paper aspires to reconcile both visions by integrating them in a whole software production process. An approach based on conceptual-schema centric software development is presented, where conceptual primitives intended to specify static, dynamic and interaction aspects are properly provided. Furthermore, Model Transformation techniques are proposed to go from the problem space, represented by the Conceptual Schema, to the solution space, represented by the corresponding final software product. This proposal is underpinned by some current MDA-based technology, which makes user-oriented, model-based software generation a reality.

Keywords: Conceptual modeling of user interface, Functional requirements, Model-based code generation, User Interaction and Model-driven approach

1. INTRODUCTION

Traditionally, modeling an Information System from a Software Engineering (SE) perspective basically consists in specifying its static (data-oriented) and dynamic (function-oriented) architecture. A lot of methods and techniques have been provided in the past to solve this specification problem,
including well-known data modeling techniques (e.g., the Entity-Relationship Model [5] and its extensions), and process modeling approaches (e.g., Structured Analysis, Data Flow Diagrams). Object-Oriented Modeling was seen in the nineties as the way to encapsulate statics (data) and dynamics (behavior) under the common notion of object, and new methods [3,24] and languages (e.g., UML [2]) have been proposed under this unified paradigm. The focus at the modeling step has always been put on those data and functional system aspects, while one very important issue was normally left at least for design time: the user interaction. Why interaction modeling is not considered at the same level than data and behavior modeling in the vast majority of SE-based software production methods? Isn’t interaction an essential part of the world description, as system data and functionality are?

It has been remarkable to realize that, even if the design and the implementation of User Interfaces (UIs) are recognized to be the most time-consuming step of any software production process, its modeling was rarely considered at the same level of data and function modeling when specifying a system. A whole community emerged to face that problem: the Human-Computer Interaction community (HCI).

To face and solve this dichotomy, one challenging goal in the context of both SE and HCI is to provide proper bridges between their best-known software production methods and techniques. Starting from the idea that SE is considered to be strong in specifying data and functional requirements, while HCI is centered on defining user interaction at the appropriate level of abstraction, a sound software production process must provide ways for specifying in a precise way data, functionality and interaction, all together. If any of those aspects is not properly faced, the whole software production process will fail, because the reality to be modeled is a mix of data, functionality and interaction. Consequently, software production methods that combine the most data-oriented and functional-oriented, conventional requirements specification, with the more interaction-oriented, UI modeling aspects are strongly required.

In this context, Model Transformation technologies (i.e., MDA approaches [13]) make possible to provide a global software process where all the relevant aspects of the analyzed problem (structure, behavior and user interaction) are specified from the beginning (Requirements Model). Those resulting models are first projected onto a Conceptual Schema and onto the final software product later. Based on the use of this Model Transformation approach, the intended contribution of this work is to provide the basis to build such a software production process, with two basic principles in mind:

1. Model Transformation is used as the basic software production paradigm, to automate the conversion of a source Requirements Model into its corresponding Conceptual Model and then converting this conceptual model
Generating User Interfaces from Conceptual Models

into the System Design and Implementation. A Model Compiler should be the responsible of implementing the corresponding mappings.

2. Each modeling step provides appropriate methods to deal properly with the specification of structural, functional and interaction properties. To do that, the conceptual primitives (conceptual constructs) that constitute the basic building blocks of the required models must be properly identified. The definition of Conceptual Patterns constitutes a good strategy to define those conceptual primitives in detail.

Taking advantage of the current research on Model Transformation methods and tools (where we find MDA [13], Extreme Non-Programming [15], and Conceptual-Schema Software Development), we will focus on the Conceptual-Model-to-Software-Product Model Transformation Process. Interaction patterns at the conceptual level of modeling will be introduced, and how to convert them into UIs will be shown, providing an MDA-based approach where not only data models and process models are properly converted into database and programs, but also UIs are created from their corresponding interaction model. How to go from the problem space to the solution space in an automated way will be commented based on the OlivaNova Model Execution (ONME) tool [4], an MDA-based tool that generates a software product that corresponds to the source Conceptual Schema. We intend to demonstrate that conceptual modeling is more powerful when user interaction and system data and behavior are modeled within a unified view at the conceptual level.

To fulfill those goals, the paper is structured as follows. Section 2 presents an overview of model-based UI development environments proposed in the literature. Section 3 introduces an object-oriented, model-based software production process, where system statics, dynamics and interaction aspects are specified in a unified framework. In particular the interaction patterns required for modeling system interaction are introduced in detail, while in Section 4 how to go from the interaction model to the final UI is shown. Finally, the conclusions, some lessons learned derived from the process application, and future works are presented.

2. RELATED WORK

If we look for approaches to design and implement UIs based on modeling, there are two main groups of proposals in the HCI area:

- **MB-UIDEs**: From an HCI point of view, there is a number of model-based UI development environments reported in the literature. In da Silva’s survey [6], several MB-UIDEs are reviewed, distinguishing two
generations of tools. The aim of the first generation was to provide a run-time environment for UI models; some examples are COUSIN [9], HUMANOID [20] and MIKE [17]. The second generation aimed to provide support for interface modeling at a high level of abstraction. Examples of these environments include ADEPT [12], GENIUS [10], MASTERMIND [21], MECANO [19], TADEUS [8], and TRIDENT [1]. Many of the second generation of MB-UIDEs rely on a domain model. This model is often a description of the domain entities and relationships between them, which are represented as a declarative data model (as in MECANO), an entity-relationship data model (as in GENIUS), or an object-oriented data model (as in TADEUS). Some MB-UIDEs like ADEPT, TADEUS, TRIDENT and UsiXML [11,22] propose task models as a primary abstract interaction modeling, from which the abstract interface models (or its equivalent dialogue models) are later derived. It is important to remark that UsiXML is an XML-based interface description language that is supported by a suite of tools [23], ranging from creating UI sketches to generating the final UI. Therefore, we will consider UsiXML [11] as an MB-UIDE for the purposes of this review.

- **UML-based approaches:** WISDOM [16] is a UML-based SE method that proposes a use case-based, evolutive method in which the software system is iteratively developed by incremental prototypes until the final product is obtained. The UML notation has been enriched with the necessary stereotypes, labeled values and icons to allow a user-centered development and a detailed UI design. Three of its models are concerned with interaction modeling at different steps: the Interaction Model, at the analysis stage; and the Dialog Model and the Presentation Model during the design stage, as refinements of the Interaction Model. Another important proposal is UMLi [7]. It is a set of UI models that extends UML to provide greater support for UI design. UMLi introduces a new diagram: User Interface Diagram. UMLi is the first reliable proposal of UML to capture the UI formally. However, the models have so detailed that do the modeling very difficult. Problems of middle size are very difficult to specify, that is what probably explains why UMLi is not adopted in industrial environments.

When we compare the reviewed approaches, in general we find a poor lifecycle support in most MB-UIDEs, and a lack of integration between models to provide a full software production process. In addition, none of the reviewed MB-UIDEs allow the specification of the system functionality. The result is that the application being modeled cannot be completely generated. There are some efforts leading to properly map the elements of a task model to the elements of a domain and interface models by defining a transformation model and the corresponding support tools, but although there are
tools that deal with the final UI generation, no business layer is generated due to the lack of a precise functional model.

The UML-based approaches try to solve the problem by integrating all the different perspectives through the use of UML. But being UML so imprecise in terms of semantics of its conceptual primitives, what they finally provide is a mess of models and notations where model transformation is not possible because the formal semantics of those basic building blocks of the different models are not precisely defined.

From a SE point of view, some development methods and environments have been proposed. They normally use a class diagram-like model to capture the system structure and a process model to fix the functionality that the system is supposed to provide. In addition, in recent years some CASE tools (e.g., Together, Rational Rose, Poseidon) have been proposed with the objective of providing some kind of automation to manage these models. Anyway, interaction modeling is not a usually key issue when requirements and conceptual modeling is represented in a software production process.

Being how to achieve this whole integration in a model transformation-based platform a basic goal, in the next section we are going to see how all those modeling pieces (data, behavior, and user interaction) can be properly put together in a sound Software Production Process.

3. MODEL-BASED INTERFACE DEVELOPMENT WITH OLIVANOVA MODEL EXECUTION

In this section, we present a complete software production process that combines functional requirements specification, user interaction design, and implementation. It is defined on the basis of the OO-Method [18] and the OlivaNova Model Execution (ONME) tool [4], an implementation of the OO-Method. OO-Method is a model-based environment for software development that complies with the MDA paradigm [13] by defining models of a different abstraction level. Fig. 1 shows the parallelism between the models proposed by MDA and the models dealt with in OO-Method [18].

At the most abstract level, a Computation-Independent Model (CIM) describes the IS without considering if it will be supported by any software application; in OO-Method, this description is called the Requirements Model. The Platform-Independent Model (PIM) describes the system in an abstract way, having in mind that it will somehow be computerized but still disregarding the underpinning computer platform; this is called the Conceptual Model in OO-Method. ONME implements an automatic transformation of the Conceptual Model into the source code of the final user application. This
is done by a *Model Compilation* process that has implicit knowledge about the target platform. This step is equivalent to the Platform Specific Model (PSM) defined by MDA. In the following, we explain the main steps of such a software production process, focusing on the Conceptual Model to final Software Product transformation.

![Figure 1. An MDA-based development framework for UI development](image)

### 3.1 Modeling Data and Behavior

In order to cover the data and behavior perspectives, OO-Method defines a PIM called *Conceptual Model* consisting of: the Object Model, the Dynamic Model, and the Functional Model, which can all be designed using the Requirements Model as input. The *Object Model* specifies the object structure and its static interactions. It is modeled as an extended UML class diagram. The *Dynamic Model* represents the control, the valid sequences of events and the interaction between the objects. The *Functional Model* specifies how events change the object states. Therefore, the behavior of the system is modeled by the Functional and Dynamic Models working together.

Detailed information of the Conceptual Patterns required specifying data and behavior through these three models can be found in [18]. In this paper, we want to emphasize how to do a similar work from the system interaction point of view. This means that we are going to work with Interaction Model. This Interaction Model will be the last component of the Conceptual Modeling step, and we are going to see which basic building blocks are required to construct a correct specification. It must be remarked that each interaction pattern exists because a corresponding software component counterpart is associated to it. This is why the design and implementation of a Model Compiler will be the logical consequence of the approach.

### 3.2 Modeling Interaction

To model the interaction between the system and the user, OO-Method incorporates an *Interaction Model*. Based on a set of basic interaction
patterns, it allows describing the interaction by means of three levels of presentation patterns as shown in Fig. 2.

These three levels are:

- **Level 1. Hierarchy of Actions Tree (HAT):** It organizes in a tree the functionality that will be presented to the different users who access the system. Each intermediate node of the tree acts as a container with a label. Each leaf node contains a label and a link to an interaction unit. As
we will see later, under a model transformation perspective, these data will be automatically mapped to menus (in windows environments) or pages and links (in web environments) in the implementation phase. Using a tree structure is a good technique to support the Gradual Approach. The interaction units referenced in the tree leaves are those basic, level 2 interaction units that are described next.

- **Level 2. Interaction Units (IUs):** They represent abstract interface units that the user will interact with to carry out his/her tasks. There are three main types of interactions to be considered:

  1. **Service UI.** A Service IU models a dialogue whose objective is to help the user execute a service. Normally, the need of a Service IU is the consequence of the existence of a functional requirement - coming from the Requirements Model - that the modeler will identify as a service associated with a certain class. In terms of specification, the Service IU encapsulates the interaction unit to supply a service in the interface. In this context, the service specification can be completed asking the user: *What input data are needed for this service?*, *How should input data be grouped?*, *Are the input fields inter-related?*, *What kind of feedback do you need for each selected object?*, etc. The answers to these questions will be specified by using other auxiliary simple patterns that are the lower level interaction patterns –level 3, introduced next–. Being the service argument the main granule, the questions to be answered are related to: *Introduction* aspects (constraining the edition of values), *Defined Selection* (defining by enumeration the valid values), *Population Selection* (expressing how to select objects for OO arguments), *Dependency* (expressing dynamic interdependencies among UI components), *Supplementary Information* (providing extra feedback for object identifiers), *Status Recovery* (recovering argument values from attributes), and *Grouping* (logical grouping of arguments).

  2. **Instance UI.** It specifies how to visualize one selected instance of a class. An Instance IU models the data presentation of a class instance and supports the required interaction with it. It is oriented to object manipulation. In user terms, the Instance IU arises from the necessity to observe single objects. In addition, the user may want to change the state of the selected object (by means of services) and/or to navigate to related objects. In the problem domain the analyst identifies a certain class, e.g., Vehicle. Once the class is defined, he could ask the user the following questions: *Which attributes of the object do you wish to view?* *What actions can be done on the object?* *Which additional info (relationships) do you want to reach in the
Generating User Interfaces from Conceptual Models

(interface by navigation? Again, the answer to these questions is provided by the lower-level, auxiliary interaction patterns (level 3), that will fix the concrete functionality. In particular, the specification of the Instance Pattern is defined using three elementary patterns: a Display Set (fixing what properties to show), Actions (specifying what services could be executed by the instance) and Navigation (defining the links to the association/aggregation relationships which can be reached from the corresponding instance). In OVID, the concept of view plays a similar role to the Instance Presentation Pattern.

3. Population UI. This interaction pattern deals with the necessity of working with object collections. It allows specifying how to filter and search among the selected objects, how to order them when presented to the user, how to navigate from a selected object of the collection to others attached to it, and what actions can be executed from an object. Again, these properties are dealt with through the use of the corresponding lower-level patterns –elementary patterns–. We will introduce a Filter, an Order Criterion, a Display Set, a Navigation path, and a list of Actions respectively. From the methodological point of view, the modeler identifies a class with its attributes, services, and relationships that will be used as component bricks for the specification of this kind of user interaction. More complex interaction can be specified based on combinations of the previous ones. For instance, a Master/Detail IU could be defined as a complex interaction unit that deals with master/slave presentations.

- Level 3. Elementary Patterns (EPs): These patterns constitute the primitive building blocks of the UI and allow the restriction of the behavior of the different interaction units. In the current OO-Method Interaction Model proposal there are 11 Elementary Patterns, which have been introduced associated with their corresponding relevant Interaction Unit. As a matter of example, let us see how to characterize the Filter EP. A filter is a criterion specified for searching for objects. Once a class has been picked, the user needs to search for objects satisfying a given condition. Example, searching cars by fare or color in a Car Rental Service. By fixing the required attributes to be used for the query, and their intended values –if any-, the system can create the corresponding filter criterion. If the modeler has identified the Vehicle class in the problem domain, he can ask the user: How do you need to search for Vehicles? Each answer from the user constitutes a component of the filter criterion to be built. Example, the formula for searching red cars with special fare is: color="red" and fare.code="special".
4. GENERATING THE SYSTEM

Generating the System is better understood if tools are used. As OO-Method has ONME tool as implementation, we will use it to explain the model-based code generation strategy. As we are focusing on User Interaction, we will center the presentation on how to generate the UIs. How to generate the Database and the set of programs that implement the system data and functionality respectively can be seen in detail in [18] from a more theoretical point of view, and in [4] from a more industrial-oriented perspective.

In MDA terms, once we have completed the PIM, the next step is to take advantage of ONME automatic production of the source code (Code Model). Nowadays, ONME implements a Model Compilation process to automatically transform the information described in the Conceptual Model into a full software system in the following platforms: Visual Basic, C#, ASP .NET, Cold Fusion and JSP; using as a repository SQL server 2000, ORACLE or DB2. This results in a three-layer application that includes the interface tier, the business tier, and the persistence tier. As an explanation of the transformation patterns implemented by the ONME, Table 1 shows the correspondences between PIM elements and concrete widgets for the Visual Basic platform [14].

<table>
<thead>
<tr>
<th>Presentation model</th>
<th>VB component</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hierarchical action tree (HAT)</td>
<td>Application menu</td>
</tr>
<tr>
<td>Interaction unit</td>
<td>MDI child form</td>
</tr>
<tr>
<td>Service IU</td>
<td>MDI child form with entry controls</td>
</tr>
<tr>
<td>Simple type service argument</td>
<td>Entry control</td>
</tr>
<tr>
<td>Object-valued service argument</td>
<td>Generic control OIDSelector</td>
</tr>
<tr>
<td>Filter</td>
<td>Filter control</td>
</tr>
<tr>
<td>Filter variable</td>
<td>Entry control</td>
</tr>
<tr>
<td>Actions</td>
<td>Button panel</td>
</tr>
<tr>
<td>Action item</td>
<td>Button</td>
</tr>
<tr>
<td>Service throw</td>
<td>“Accept” button and code to validate and invoice the service</td>
</tr>
<tr>
<td>Cancellation</td>
<td>“Cancel button” and code to close the formulary</td>
</tr>
<tr>
<td>Services invocation</td>
<td>Code for linking business logical layer with service invoice</td>
</tr>
<tr>
<td>Data presentation</td>
<td>Code for formatting and data recovery</td>
</tr>
</tbody>
</table>

To better understand the UI Generation Process, let’s comment how a concrete Service IU specification (at the Problem Space level) is properly transformed into a Concrete User Interface (at the Software Space level).
From a conceptual perspective, when specifying a class service in the Class Model—as for instance, a New Expense Report service as shown in Fig. 3—, we have a set of arguments of a corresponding type. For such a service argument, a couple of label and text edit controls will be introduced. Position, format and length are interaction properties that can be incorporated in this specification level. Additionally two buttons will be added: one for activating the service (ok) and another to cancel it (cancel).

![Figure 3. A Service UI](image)

Additionally, some service arguments will provide the facility of value selection from a given condition, defined on the corresponding target class attributes. In this case, a population selection EP is specified determining those attributes to be used for building the selection condition. The transformation process will indicate this situation using a magnifying glass icon. An associated execution semantics fixes the final implementation. Clicking this icon, a selection window will allow the user to introduce the desired values for the attributes of the condition to be built, and the corresponding query will be triggered to obtain the set of potential valid values; one of them will be selected as a service argument. The following table shows each interaction model component that is used to define a class service window and its interface component implementation.

<table>
<thead>
<tr>
<th>Presentation model</th>
<th>Final interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hierarchical action tree (HAT)</td>
<td>File Changes Others Window</td>
</tr>
<tr>
<td>Service IU</td>
<td>The whole window</td>
</tr>
<tr>
<td>Service throw and Cancellation</td>
<td>OK</td>
</tr>
<tr>
<td>Simple type service argument</td>
<td>Cancel</td>
</tr>
<tr>
<td>Introduction pattern for a date field</td>
<td>Identifier:</td>
</tr>
<tr>
<td></td>
<td>Code for validating date format</td>
</tr>
</tbody>
</table>
As a second example let’s analyze the software representation of a Population IU at the Solution Space. As said before, what we have to specify in this case at the modeling step is which class attributes are to be seen, what filters could be required to help with the process of selecting instances, which actions could be activated when a given instance is selected, and what navigation is feasible in terms of other’s class information reachability. We can see in the previous figure a concrete representation of those interaction patterns in terms of a UI, where

1. The specified filters are in the top of the window.
2. The main screen includes the area where the data (attributes) of the objects that satisfy the corresponding filter condition are shown.
3. The available actions for a particular instance are represented in the set of icons aligned vertically at the right part of the window.
4. And where finally the valid navigation options are represented in the lower part of the window.

There are many other options, but as soon as a particular template for the final UI is defined, it is possible to implement a model transformation process where full automation becomes a real chance.
5. SUMMARY AND FUTURE WORK

Software production methods need a complete software production process that properly integrates system functionality, behavior, and user interaction from the early stages of the system lifecycle. This process should also allow the modeling, and implementation of UIs, properly integrated with the more conventional, SE-oriented modeling and implementation of system data and functionality. In accordance with these ideas, we have presented the basic principles to embed user interaction modeling as an essential part of the Conceptual Modeling phase, and converting the resulting conceptual schema into a final software product where all UIs, programs and database are properly generated; a Model Transformation-Based approach has been proposed, where the automatic generation of the complete, final software becomes an affordable dream. Some lessons learned while applying the proposed software production process include the need of providing design capabilities to face the “beautification” of the “to-be, final UI”, while modeling the higher-level interaction aspects. One open issue is to which degree interaction properties can be fully included in an Interaction Model. With the help of tools as ONME that already incorporates the proposed ideas into industrial, Model-Transformation-based Software Production, the proposed process is being empirically validated to prove its effectiveness.

REFERENCES


Chapter 2

TOWARDS OBJECT ORIENTED, UIML-BASED INTERFACE DESCRIPTIONS FOR MOBILE DEVICES

Robbie Schaefer¹ and Steffen Bleul²

¹C-LAB, Paderborn University, Fuerstenallee 11 – D-33102 Paderborn (Germany)
E-Mail: robbie@c-lab.de – Web: http://www.c-lab.de
Tel.: +49 5251 606107 – Fax: +49 5251 606065

²Distributed Systems Group FB16 - Elektrotechnik/Informatik, Kassel University
Wilhelmshöher Allee 73 – D-34121 Kassel (Germany)
E-Mail: bleul@uni-kassel.de – Web: http://www.vs.uni-kassel.de/~bleul/
Tel.: +49 561 804 6276 – Fax: +49 561 804 62 77

Abstract
To avoid multiple works in designing user interfaces for different devices and interaction modalities, the use of a meta-language like UIML seems to be appropriate in order to start with a general UI description and provide mappings to different target devices. For the support of mobile devices and to improve re-usability of user interface definitions, we introduce the language DISL that modifies several parts of UIML and propose object oriented extensions with the conceptual language ODISL.

Key words: Mobile devices, Re-usability, User Interface Markup Language

1. INTRODUCTION

Mobile devices and disappearing computers rise significant challenges in User Interface (UI) development as each device comes with different characteristics which require different interfaces. In order to avoid the development of a new interface for each available platform, a model-based approach is required to abstract from platform specific details. In fact even abstractions from certain interaction modalities are useful. Since many mobile devices have severe limitations regarding memory and processing power, further constraints have to be considered.

For these reasons, a renaissance of model-based UI development can be observed for example in [6], where several techniques are described or in
Being a meta-language, the User Interface Markup Language (UIML) [2] uses the approach of separating the interface from the logic and presentation, where the presentation for different platforms is made available through dedicated mappings. While this proved to be feasible, UIML can be improved in order to support UI development for mobile devices: reducing the amount of connections to the backend, providing means to describe UIs modality independent and allowing the re-use and refinement of UI objects. This is achieved by focusing on abstract interface elements and a powerful dialog model that is the basis for the behavioral description.

After having reflected the related works we present the language DISL which simplifies UIML in order to provide renderers for different modalities and which can be optimized for handsets with limited capabilities. By improving the dialog model, foundations are laid for an object oriented version which is introduced with the conceptual language ODISL. Followed by some examples, the architecture and implementation of a DISL-based rendering system for mobile devices is sketched and the chapter closes with some conclusions.

2. RELATED WORK

Model based approaches for UI development often start at a high level, e.g., with a task model as in [12]. UIML however is already too specific as it provides presentation elements and a behavioral section which implements an event based dialog model. Nevertheless, a sound dialog model provides a perfect abstraction from the concrete UI while being fine grained enough to model the human system interaction.

Several classical approaches exist which focus on the description of user dialogs like Dialogue-Nets [9], Petri Nets [15], UAN (User Action Notation) [7], and DSN (Dialog Specification Notation) [5]. They refer all to the same principle concepts, are based on variants of parallel Finite State Machines, and mainly differ in their description means and hierarchical decomposition into components. They define the user dialog by means of states and state transitions, which are triggered by events from the UI elements. An interesting observation is that several of these approaches provide or have been extended with object oriented features, for example as hierarchical Petri Nets [8] or with ODSN [18].

Much research has been dedicated to introduce UI development in object oriented frameworks which makes perfectly sense since for example the separation of application logic and the presentation encourages such an
approach. The Model View Controller Concept [16] may serve as an early and well known example. More recently, efforts have been done to include UI modeling methods in the UML family as in [11] and [13].

As we are particularly interested in the reusability of User Interface Objects with all their axes, the PAC model [4] provides valuable insights: There, an interactive application is structured into abstraction, presentation and control which can be recursively applied. In contrast, UIML is more oriented towards the MVC as it is based on the Meta Interface Model which can be considered as a finer grained version of the MVC. An important part of this model is the behavioral section, which describes the control model and can be based on one of the previously introduced dialog models. In the next section we discuss how UIML can be improved by integrating object oriented concepts and abstractions from concrete views.

3. UIML FOR DIFFERENT DEVICES AND MODALITIES

UIML can be used to build multi-platform UIs [1]: first, a generic vocabulary is used to specify a more abstract UI and in a second step, the generic UI is transformed to a concrete UI representation such as Java Swing or HTML. This principle works well, for different platforms that share the same interaction philosophy, as UIML supports linear mapping with the peers section and a generic vocabulary can thus be transformed quite straightforwardly. For example all widgets of an HTML-Form are available with most APIs for Graphical User Interfaces. So if a generic vocabulary supports forms, a mapping to other platforms is mostly a matter of diligence.

Problems occur however, when we include platforms with completely different interaction modalities such as voice and gestures. In this case not only the naming of widgets and their properties have to be changed, but sometimes also the complete structure of the interface, which can be hardly achieved with UIML in its current form. Therefore we propose to abstract the vocabulary even more and combine it with an external transcoding approach or the usage of an interpreter for the required modality. Both will be detailed in the next subsection.

3.1 DISL - An Extended Subset of UIML

In order to allow generic and modality independent dialog descriptions, we used UIML as a base, simplified it by removing the capability of being a meta-language and improved the behavioral part for storing and processing
several local UI- and system states. The changes which resulted in the language DISL (Dialog and Interaction Specification Language) are motivated and explained in the next paragraphs.

As one design goal was the support for mobile devices, we intended to allow renderers on the device. Therefore there was a need to simplify UIML. Being a meta-language, UIML uses a vocabulary which is needed to map UIML to concrete interfaces. For DISL we wanted to avoid different vocabularies, so that one UI model can be used and an interpreter for devices with limited storage and processing power could render the model. For devices where there is no DISL renderer available, we rely on external transformations e.g., DISL to WML.

As we do not have a vocabulary-based mapping from UIML to toolkit specific widgets, we defined modality independent generic widgets which are inspired amongst others by [14]. This concept does not describe real widgets but only general interaction elements for input, output and logical grouping. For example we defined a variablefield for output that could be rendered as text in a GUI, synthesized speech in a voice driven interface, or as beeps or light flashes.

Even more important are changes done to the behavioral part of UIML, as we introduced variables and arithmetic operations. The variables can be used to store system states, which allow processing multiple states together with events and compares to dialog models which allow parallel transitions. In fact, the control model of DISL is based on the DSN [5] notation.

By allowing basic arithmetic operations on variables representing system states, a UI can have some logic directly in its behavior. The aim is not to weaken the separation of concerns but to minimize unnecessary calls to the backend application. As a simple example consider a small interface which shows the number of button presses. For each button press, the backend application has to be called in order to provide an updated counter. So, on a mobile phone, the UI freezes every time the button is pressed. If the increment of the counter is implemented in the interface, the backend has only to be called when important input has to be transmitted or the event processing requires synchronization.

3.2 ODISL - Towards Object Oriented UIML

For re-using parts of a UI specification, UIML supports templates which can be defined for the most important parts of a UI description, amongst others presentation, structure, style, content, behavior and logic. Templates in UIML form a viable concept which is in some aspects close to the object oriented paradigm for the following reasons:
• Templates support the principle of inheritance, as a template may consist of other templates which are included via the source attribute. So, UIML documents could be defined as a template which serve as a base class and through inclusion of this base template, another UIML document can inherit from the first document.

• Parts of a template can be specified as hidden.

• In principle, polymorphism is supported through UIML templates, as the restructure element has cascade and replace attributes which can be used to modify “inherited” properties.

However, object oriented features are only implicitly available through the UIML template concept and since they are not designed by purpose to be object oriented, some difficulties occur as described in the open issues document by the UIML standardization committee [3]. For example, there should be a way to restrict access to certain elements of a template by making them immutable or making parts of a template optional could be desired. Another open issue is addressing the parameterization of templates in order reuse the behavior specification. We think that a profound introduction of real object orientation to UIML will solve the issues described before. As templates only support some aspects of object orientation, they fail in other aspects, when it comes to apply real objects that are instances of classes, since the subclassing of templates constitutes mainly of a textual changes to the UI-tree via replace, union and cascade.

Key to an object oriented UI markup language is the possibility to inherit and encapsulate the behavior of the UI. Therefore the behavioral extensions proposed by DISL prove to be very useful, as we now have a way to capture states and describe behavior that is intrinsic to local UI objects. All that is needed is a way to really encapsulate these parts and make them accessible to child objects and allow instantiation of UI objects. So, DISL has been modified and UIs are now referred to as interfaceclasses which can be instantiated through their identifier. Interfaceclasses may inherit from other interfaceclasses.

```xml
<method id="setVisible" access="public">  
  <params>  
    <param id="visible" type="boolean"/>  
  </params>  
  <returns>  
    <return id="isVisible" type="boolean"/>  
  </returns>  
  <action>  
    <statement>  
      <variable-content id="isVisible"/> <param-content id="visible"/>  
    </statement>  
  </action>
```
Figure 1. A method definition in ODISL

Each interfaceclass consists of the elements structure, style, behavior and methods. If such an element is not stated in the interfaceclass, the class inherits the element from the parent class. If an element is specified, the content extends the inherited content. Furthermore, access rights as public or private can be set for most DISL-elements. Methods are introduced to operate on the elements of the interfaceclass they where defined in. They take a list of input parameters and return values. Inside a method, an action part is used to set values or trigger events. A similar part is needed to specify how the result values are constructed e.g., by assignments with the content of internal variables. Since this can happen only after the action part is resolved, this element is called “afterresolving”. The code excerpt in Fig. 1 shows how a method is constructed in ODISL.

4. IMPLEMENTATION AND EXAMPLES

In order to visualize the concepts of the previous sections, some examples are provided, after having detailed a client-server architecture that provides UI descriptions for mobile devices. This architecture allows controlling applications on the mobile device, on the server or using the device as a universal remote control as it is done within the pebbles project [10]. Having a UI server allows also a more flexible handling of UI descriptions as they can be transformed into specific target formats for mobile devices, which do not have dedicated DISL renderers.