

UsiXML 2010

Proceedings of the 1st International Workshop on
USer Interface eXtensible Markup Language

Sponsored by:

UsiXML, ITEA2 project #2008026

ITEA2 Program of the European Commission

Eureka Project #3674

Also sponsored by:

ACM's Special Interest Group on Computer-Human

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

Supported by:

Thales Research & Technology France, Paris (France)

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

Foreword

UsiXML'2010, the 1st International Workshop on User Interface eXtensible Markup Language, was held in Berlin, Germany (June 20, 2010) during the 2nd ACM Symposium on Engineering Interactive Computing Systems EICS'2010 (Berlin, 21-23 June, 2010). This workshop is aimed at investigating open issues in research and development for user interface engineering based on User Interface eXtensible Markup Language (UsiXML), a XML-compliant User Interface Description Language and at reviewing existing solutions that address these issues. In particular, the “ $\mu 7$ ” concept is explicitly addressed by discussing how and when each dimension can be supported: multi-device, multi-user, multi-linguality, multi-organisation, multi-context, multimodality, and multi-platform. The workshop received 28 full paper submissions, from which 23 have been accepted by the International Program Committee.

Once again, we acknowledge the support of the ACM through its special interest group SIGCHI. We also wish to thank our corporate sponsor Thales Research & Technology France and the support of our academic sponsor: Université catholique de Louvain.

Finally, we thank Marco Winckler of University of Toulouse for assisting with the conference web site and Marco Blumendorf for all the efforts put in organizing EICS'2010. We hope that you find this year's program exciting and we are looking forward to meeting you in Berlin!

David Faure and Jean Vanderdonckt

Workshop General Co-Chairs

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

Published by Thales Research and Technology France, June, 2010.

ISBN: 978-2-9536757-0-2

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



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

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

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

More Efficient Model-Based Approach

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

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

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

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

Helping Address The Whole European Market

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

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

Adding Versatile Context-Driven Capabilities

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

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

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

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

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

Demonstrating Real Benefits In Use

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

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

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

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

UsiXML Project technical data

Project start: October 2009

Project end: September 2012

Partners

Thales Research & Technology, Université catholique de Louvain, PY Automation, Gostai, DefiMedia, See & Touch, Namahn, ProDevelop, Company for Software and Development, VR Sense, Baum Engineering, Institut Télécom, Telefónica I+D Investigación y Desarrollo, Institutul National de Cercetare-Dezvoltare in Informatica, Université Joseph Fourier, Laboratoire d'Informatique de Paris 6, University of Namur, DAI-Labor, University of Rostock, Universidad Politecnica de Valencia, University of Castilla-La Mancha, University of Madeira, Informatics and Telematics Institute, Université de Lausanne, Agence Wallonne des Télécommunications.

Countries

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

Contacts

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

David.Faure@thalesgroup.com

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

Jean.Vanderdonckt@uclouvain.be

Project website:

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

UsiXML project web site: <http://itea.defimedia.be/usixml-france>

Support

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



UsiXML 2010 Workshop Organization

General Co-Chairs: David Faure (*Thales Research & Technology, France*)
Jean Vanderdonckt (*Université catholique de Louvain, Belgium*)

Program Committee: C. Agüero (*Thales Research & Technology NL, The Netherlands*)
B. Bomsdorf (*University of Fulda, Germany*)
J. M. Cantera Fonseca (*Telefonica, Spain*)
A. Coyette (*Université catholique de Louvain, Belgium*)
S. Dupuy-Chessa (*Université Joseph Fourier, France*)
D. Faure (*Thales R&T, France*)
E. Furtado (*University of Fortaleza, Brazil*)
J. M. González Calleros (*Université catholique de Louvain, Belgium*)
G. Gronier (*Centre de Recherche Public Henri Tudor, Luxembourg*)
S. Kanai (*University of Hokudai, Japan*)
S. Lepreux (*Université de Valenciennes, France*)
V. M. Lopez Jaquero (*University of Castilla-La Mancha, Spain*)
S. Madelénat (*Thales R&T, France*)
H. Mendonça (*Université catholique de Louvain, Belgium*)
S. Praud (*Thales R&T, France*)
D. Roscher (*DAI-Labor, Germany*)
G. Rossi (*University of La Plata, Argentina*)
A. Seffah (*Université de Lausanne, Switzerland*)
K. Sousa (*Université catholique de Louvain, Belgium*)
J. Vanderdonckt (*Université catholique de Louvain, Belgium*)

Table of contents

Storyboard creation as an entry point for model-based interface development with UsiXML	1
<i>Luyten, K., Haesen, M., Ostrowski, D., Coninx, K., Degrandt, S., Demeyer, S.</i>	
Comparative Analysis of Collaborative Approaches for UsiXML Meta-Models Evolution	9
<i>Boukhebouze, M., Koshima, A., Thiran, P., Englebert, V.</i>	
Designing Workflow User Interfaces with UsiXML.....	15
<i>Guerrero Garcia, J., Vanderdonckt, J.</i>	
Prototyping Digital, Physical, and Mixed User Interfaces by Sketching	27
<i>Coyette, A., Vanderdonckt, J.</i>	
GUILayout++: Supporting Prototype Creation and Quality Evaluation for Abstract User Interface Generation	39
<i>Montero, F., López Jaquero, V. M.</i>	
A Framework for Accessibility Testing of Virtual Environments based on UsiXML	45
<i>Kaklanis, N., Moustakas, K., Votis, K., Tzovaras, D.</i>	
3D User Interfaces for Information Systems Based on UsiXML.....	55
<i>González-Calleros, J. M., Vanderdonckt, J.</i>	
Implementation of Model-based User Interfaces for Pictorial Communication Systems using UsiXML and Flex.....	65
<i>Pitkänen, A., Hickey, S.</i>	
WallShare: A Multi-Pointer User Interface.....	75
<i>Villanueva, P. G., Tesoriero, R., Gallud, J. A.</i>	
Association between Requirements of Interaction of iDTV Learning applications and UsiXML models by addressing the need to structure TVilos in Crossmedia	83
<i>Militão, J. G., Furtado, E., Matos, Í</i>	
Model-driven Generation of Ergonomic User Interfaces in Visualization Services	91
<i>Gronier, G., Kubicki, S., Vagner, A., Schwartz, L., Montecalvo, E., Altenburger, T., Halin, G.</i>	
Service-Oriented Architecture for Supporting Collaborative User Interface Development.....	99
<i>Mendonça, H., Sousa, K., Vanderdonckt, J.</i>	
TRIAD: Triad-based Rich Internet Application Design.....	111
<i>Martínez-Ruiz, F. J., Vanderdonckt, J., Muñoz Arteaga, J.</i>	
OpenInterface: A Lightweight Open Source Platform for Rapid Prototyping of Multimodal Applications.....	121
<i>Lawson, J-Y. L., Vanderdonckt, J., Macq, B.</i>	
Pattern-Based Design of User Interfaces in UsiXML	131
<i>Montero, F., Vanderdonckt, J.</i>	
User Interfaces Composition with UsiXML.....	141
<i>Lepreux, S., Vanderdonckt, J., Kolski, C.</i>	
Rule Based Conversion of UsiXML-based GUI Description.....	153

<i>Paskalev, P., Serafimova, I.</i>	
Adapting UsiXML User Interfaces to Cultural Background.....	163
<i>Khaddam, I., Vanderdonckt, J.</i>	
A Comparative Study of Task Concepts in UsiXML and CTML and their Applications.....	173
<i>Wurdel, M., Forbrig, P.</i>	
Generating User Interface for Information Applications from Task, Domain and User models with DB-USE.....	183
<i>Tran, V., Vanderdonckt, J., Kolp, M., Wautelet, Y.</i>	
A Proposal for Enhancing the UsiXML Transformation Meta-Model.....	195
<i>Aquino, N., Panach, J. I., Pastor, O.</i>	

On stories, models and notations: Storyboard creation as an entry point for model-based interface development with UsiXML

**Kris Luyten, Mieke Haesen,
Dawid Ostrowski, Karin Coninx**
Hasselt University - tUL -IBBT
Expertise Centre for Digital Media
{kris.luyten, mieke.haesen, dawid.ostrowski,
karin.coninx}@uhasselt.be

Sylvain Degrandart^{1,2}, Serge Demeyer¹
¹Dept. Mathematics and Computer Science,
Universiteit Antwerpen
²University of Mons - UMONS
{Sylvain.Degrandart,
Serge.Demeyer}@ua.ac.be

ABSTRACT

Storyboards are excellent tools to create a high level specification of an interactive system. Because of the emphasis on graphical depiction they are both an accessible means for communicating the requirements and properties of an interactive system and allow the specification of complex context-aware systems while avoiding the need for technical details. We present a storyboard meta-model that captures the high level information from a storyboard and allows relating this information with other models that are common for engineering interactive systems. We show that a storyboard can be used as an entry point for using UsiXML models. Finally, this approach is accompanied by a tool set to make the connection between the storyboard model, UsiXML models and the program code required for maintaining these connections throughout the engineering process.

Author Keywords

UsiXML, Storyboards, Model Transformations

INTRODUCTION

For over a decade, model-based interface development has been subject of ongoing research to improve the methods and tools on the one hand, and to bridge the gap with traditional software engineering on the other hand. Model-driven engineering seems to be one of the central methods that can connect model-based interface development with software engineering because of the concepts they share: models that describe different aspects of an interactive system. Despite the availability of various tools [6,9,10,11], most models require domain expertise or technical knowledge. As such, they fail to serve as a means of communication for the whole team including all stakeholders and their usage is limited to particular stages in the engineering process. Specifically for engineering complex systems such as context-sensitive or distributed multi-user systems, maintaining involvement of the whole team and providing consistency checks with initial requirements is hard.

We believe a simple and graphical tool can resolve many issues, given it can be easily connected with the typical models that are used throughout the engineering process.

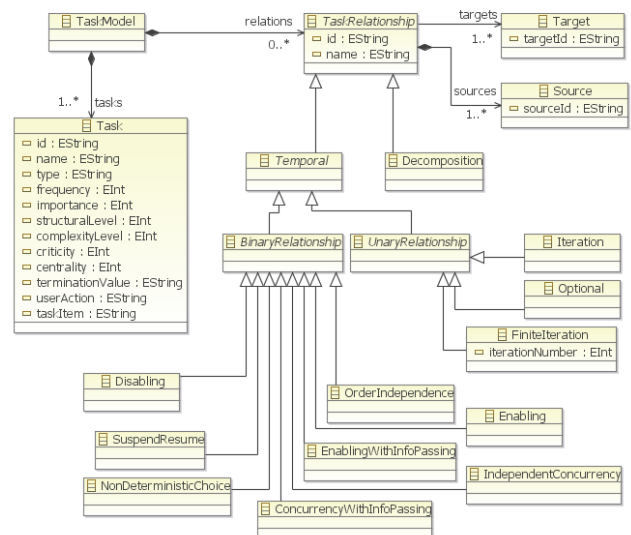


Figure 1. The meta-model for the UsiXML Task Model.

ABOUT THE MODELS AND NOTATIONS

Model-based interface design is hard to get started with: most models use an abstract notation and require at least some domain expertise. The task model is the one model that is often used as a starting point for model-based interface development processes. A task modeling notation such as the ConcurTaskTrees (CTT) notation [11] uses a graphical notation for identifying the different task types and has a specialized notation for specifying how tasks behave in time with relation to each other in time. UsiXML does not provide a separate graphical notation such as CTT, but does include the semantics for similar elements in its specification. This leaves the choice for a graphical notation up to the tool developer. Figure 1 shows the meta-model for UsiXML compliant task models.

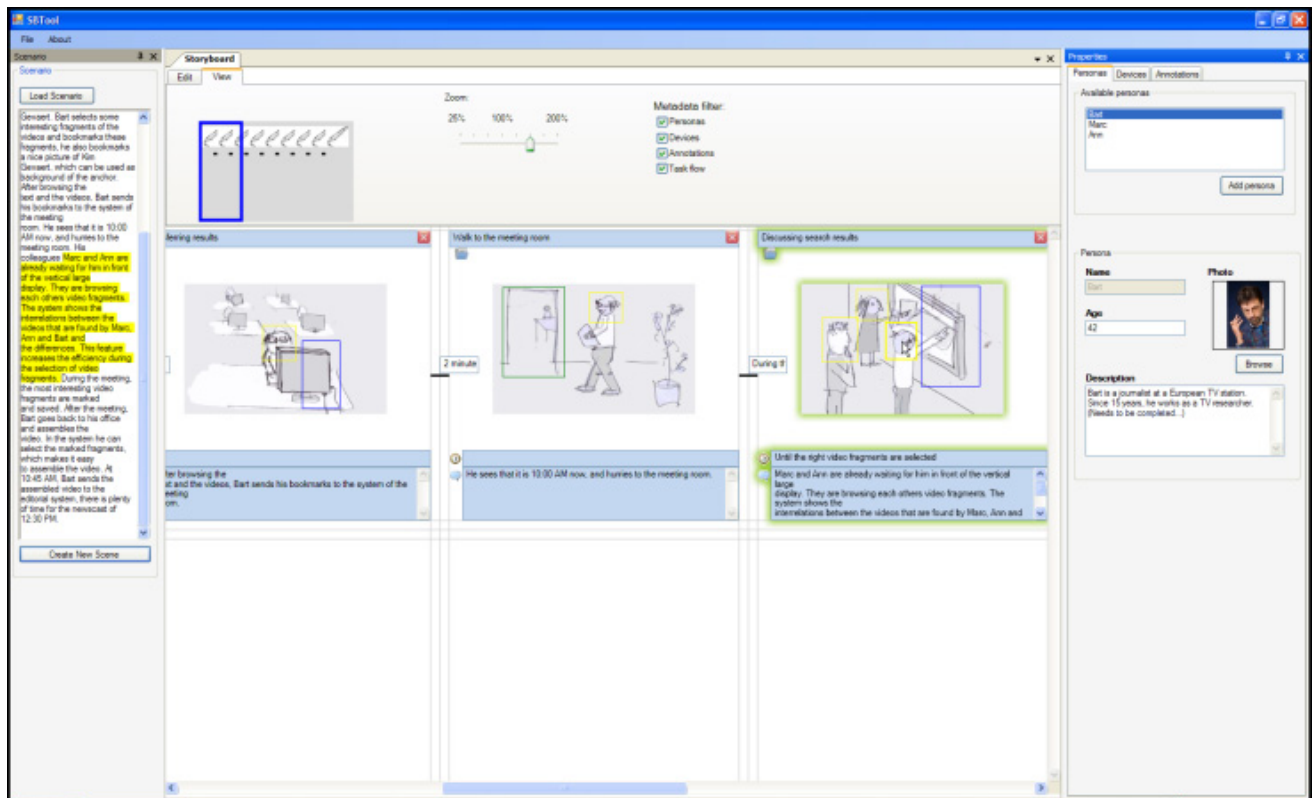


Figure 2. Screenshot of the COMuICSer tool. The centre panel shows the storyboard, while the left and right panels are reserved for the scenario and annotations.

Just as is the case with CTT, there are tools that allow specifying the task model by using the UsiXML notation, e.g. IdealXML [10] or KnowiXML [6]. We see that the CTT notation serves as an important inspiration for most of these tools and similar graphical notations are being used. UsiXML does provide well-defined relationships with other typical models such as the dialog, context and presentation model. For this purpose, it is based on the well-known Model-Driven Engineering (MDE) [13] approach and uses (semi-automatic) transformations between models to progress from abstract toward concrete models.

UsiXML supports an MDE approach by defining a set of meta-models that contain all elements and relations included in the different models [16]. It tries to cover all models that are required for user interface analysis and design, and encompasses models that are commonly used in model-based interface development. We propose the addition of another model that can serve as an entry point for the whole process to engineer an interactive system: the storyboard model. In the next section of this paper we define a storyboard model and we show how it can be connected to other models contained in the UsiXML specification.

STORYBOARDS: GRAPHICAL NARRATIVE MODELS

Scenarios [2] are suitable to be used by all team members in user interface design to define the first concepts of a future system. Unfortunately, technical people encounter difficulties when translating these scenarios into technical

specifications and models [5,7]. Furthermore, when using scenarios, it is necessary that all team members have read the narrative. Nevertheless, the scenario can be interpreted ambiguously [15]. To bridge the gap between scenarios and models, and to increase involvement of the entire team, storyboards can be created. We define a storyboard as: a sequence of pictures of real life situations, depicting *users* carrying out several *activities* by using *devices* in a certain *context*, presented in a *narrative* format. Each scene in the storyboard can depict a fragment of the scenario, and hence provides the connection between the scenario and the storyboard. This specific definition immediately provides us with a clear overview of the four primary pieces of information that can be found in a storyboard: *users*, *activities*, *devices* and *context*. COMuICSer is our approach for storyboarding that embraces these four elements. The COMuICSer tool supports a graphical representation of a scenario and allows a smooth integration with structured engineering models such as the UsiXML models. The COMuICSer tool is shown in Figure 2. In this figure, the tool is being used to visualize a scenario of how a multimedia search interface can be used on several platforms by an editorial board of a TV channel.

In contrast to a scenario, this graphical representation contains more structural information, while the entire team can still easily understand the information conveyed by the storyboard.

scenes of the storyboard. Our initial storyboard meta-model is shown in Figure 3. It is by no means our intention to formalize or restrict storyboarding activities with this meta-model. We merely want to connect this model with other artifacts that are used to design an interactive system. Our approach still provides all degrees of freedom typical for storyboard creation and mainly comes into play when the transformation of a storyboard into other models is required. In this section we provide an overview of the different elements that make up the storyboard meta-model.

There is one central element in the meta-model: the *Scene*. A set of scenes that are related using *TemporalRelationship* elements is a *Storyboard*. The *TemporalRelationship* element is based on Allen's interval algebra [1]. Though, similar to comics [8], the most common relationships used in storyboarding are the "before" and "meets" relationships, we think parallel activities should be supported since they are common in collaborative and multi-user activities. "Before" indicates one scene happened before another, and there is undefined time progress in between scenes. "Meets" indicates one scene is immediately followed by another scene, and the time progress between two scenes is virtually none. It also allows us to define more precise temporal relationships between scenes that can be exploited later on, e.g. by mapping them on the temporal relationships that are used in the task model.

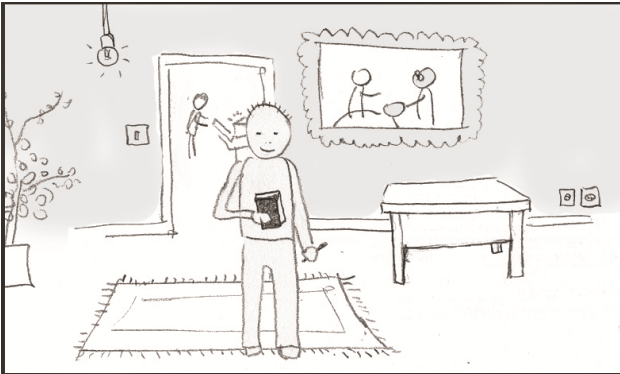


Figure 4. Scene from a storyboard that can be juxtaposed with labels that show the associated meta-data. The storyboard meta-model provides a framework to capture and serialize this meta-data for further usage.

A *Scene* is annotated with different types of information: *SceneImage* is a graphical representation of the scene, *Personas* specify archetypical users involved in the scene, *Devices* present what type of computing devices and systems are used within the scene. Because of the graphical depiction used in the *SceneImage*, it describes the context (situation) of the activities of the users in a comprehensive way. In contrast to traditional approaches in which there is some predefined structure for defining the context of use, images can be used to infer the context of use. When constructing a storyboard, the drawings or photographs used, will often

contain a lot of contextual information. Dow *et al.* show storyboarding, especially contextual storytelling, is useful for context-aware application design (in their case ubicomp applications) but lacks a good way of formalizing the context data [5].

By providing tagging of scenes, we support a rudimentary way of translating the context inferred from the *SceneImage* into a readable format. *Objects* are physical objects in a scene that are also sufficiently important to label. Take the scene depicted in Figure 4 for example: two objects that might be of interest are the light bulb (needed for watching the book) and the table in the room (needed to hold the books to read). This implies these objects could be taken into account later in the engineering cycle and thus need a representation in the storyboard meta-model. *Personas* on its turn have a set of related activities (or tasks as you might wish) and are both related to the whole *Storyboard* as well as the separate *Scenes* they participate in.

Though tagging is used to allow for a more comprehensible description of the context-of-use in a scene, several other elements of the storyboard meta-model already provide additional contextual information. Consider the UsiXML context meta-model depicted in Figure 5. This graphical depiction is created based on the UsiXML context meta-model scheme and was clearly inspired by browser profiles that were popular at the time this model was created. One can identify other one-to-one relationships between the UsiXML context meta-model and the storyboard meta-model. The *Persona* and *Device* classes can be used to generate *userStereoType* and *hardwarePlatform* classes respectively. One valuable extension is to integrate the custom tags with Wordnet (<http://wordnet.princeton.edu/>) to extract richer semantics of these tags. This could lead to a more complete transformation than is currently supported.

We showed a graphical depiction (*SceneImage*) could have high value to obtain a usable model of the context of use in previous work [17]. Figure 4 shows a scene from a storyboard that was tagged with labels about the information included by the storyboard meta-model. Several information items contribute to the context model as defined by UsiXML in Figure 5.

We do not consider our current meta-model to be complete. It does include a set of minimal elements for the storyboard to be connected with other models in a MDE process. The COMuICSer tool can generate model instances for this meta-model, which in turn can undergo transformations to enable integration with other models. We currently use this approach for two goals: first, to define mappings and consequently generate partial models (e.g. generate a set of tasks and initial temporal relationships based on the scene orderings and included activities); second, to check for consistency (e.g. the sequence of an activity diagram might contradict the temporal relationships from a storyboard).

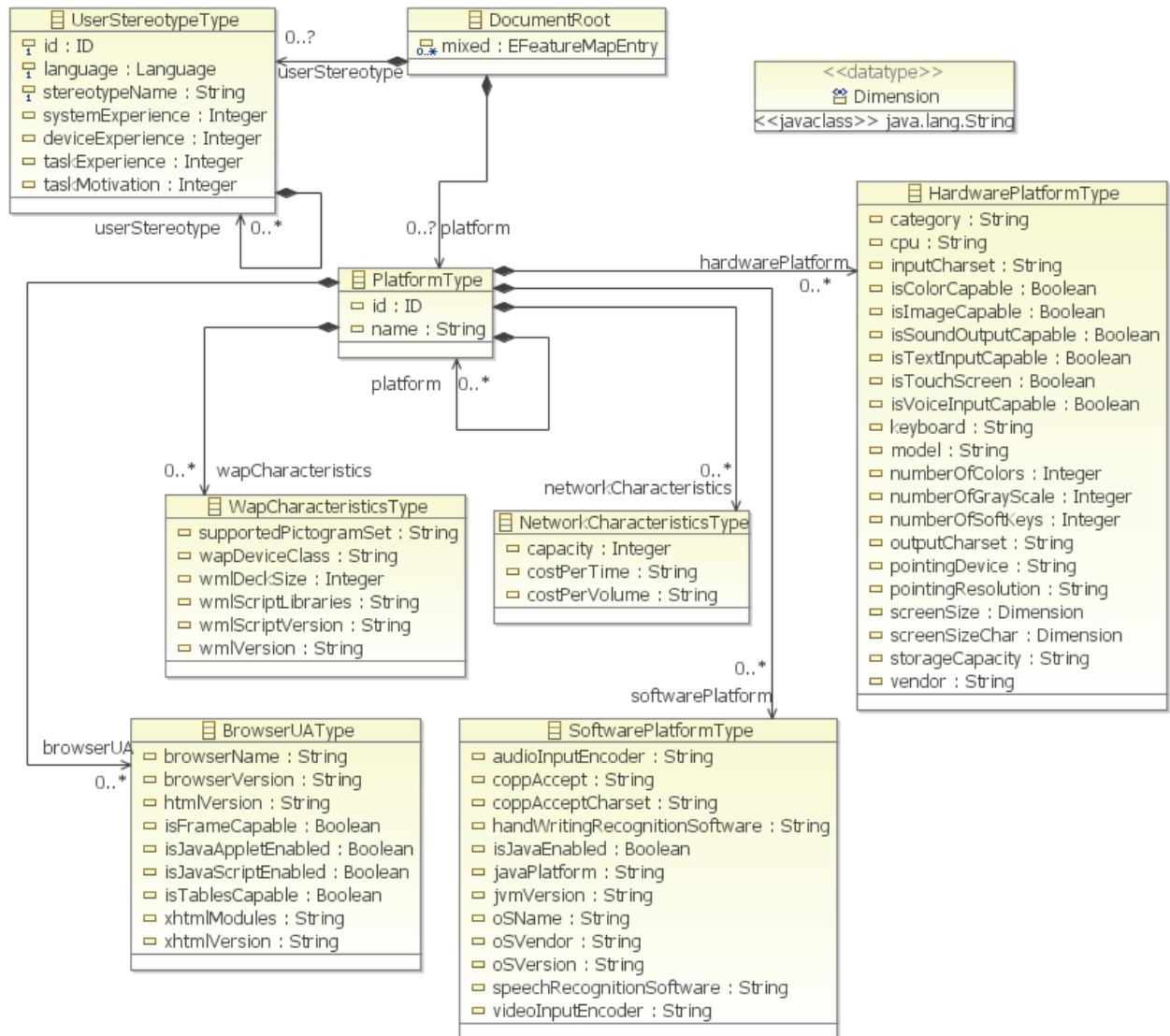


Figure 5. The meta-model for the UsiXML context model.

MAPPING BETWEEN STORYBOARDS AND USIXML

Now that we have the meta-model for the storyboard in place, we can connect it with other (UsiXML) models. Since the storyboard is often used as the initial model at the start of the engineering cycle, it can be used to generate parts of other models or to check for consistency when later models are created. The usage of a MOF-compliant meta-model ensures later integration with other MOF-compliant models is also possible. In this paper we mainly focus on integration with UsiXML models, but we envision integration more closely with traditional software engineering models in the future (e.g. activity diagrams or state charts). To show the possibilities of a storyboard meta-model, two example transformations are presented. First, by defining a mapping rule between the storyboard model and the UsiXML task model (depicted in Figure 1), we can partially generate a UsiXML task model or check its con-

sistency. Second, by exploiting the metadata from the storyboard we can add detail to the UsiXML context model (Figure 5).

Partial Task Model Generation and Consistency Check

Each scene in a COMuICSer storyboard depicts one or more activities. With the mapping editor (see next section), a developer can specify a mapping rule stating each activity from the scene should become a leaf task in a task tree. To enforce these tasks to be leaf tasks, we need to define a constraint that imposes this task cannot be decomposed any further.

All activities within the same scene are also valid during the same period of time. Thus, we can define that all activities of a scene are mapped on leaf tasks from the same enabled task set. An enabled task set is a definition from the ConcurTaskTrees language that specifies a set of tasks is valid during the same period of time. Since scenes can be

related using the Allen interval Algebra [1], for some leaf tasks it can be derived whether they need to be executed in parallel with other tasks or should use an enabling operator with its siblings. The transformation into enabled task sets could be defined using a constraint. There are clearly extensive possibilities exploiting the temporal relations and activities that occur in the storyboard to feed the task model or check the task model for consistency. It is mainly a matter of defining the rules one wants to apply and write these down as transformations that take the storyboard model as an input. Unfortunately, the Allen interval operators do not offer one to one mapping relationships with the typical CTT temporal operators. For example, two scenes s_1 and s_2 containing each two activities, $activities(s_1)=\{a,b\}$ and $activities(s_2)=\{c,d\}$, can be translated into either $(a \parallel b) >> (c \parallel d)$ or $((a > c) \parallel d) \parallel b$.

The personas, containing a description of a hypothetical archetype of an actual user, can add the distinction of different roles to a model. The different personas contained by the scene of a COMuICSer storyboard, can be connected to one or more tasks of a task model. When several personas are available in a storyboard, the personas can be the foundation for a cooperative task model.

The following code listing shows an ATL transformation for generating tasks from activities within scenes:

```
module activity2task;
create OUT : MUsiXmlTask from IN : Mstoryboard2;
rule activity2task{
  from
    a : Mstoryboard2!Activity
  to
    t : MUsiXmlTask!Task (name <- a.title)
}
```

Partial Context Model Generation

Another UsiXML model that has a clear relationship with the storyboard model is the UsiXML context model. We depicted the context model in Figure 5 for the reader's convenience. From literature it is clear that context is a vague term that encompasses many elements. The UsiXML context model makes a trade-off of what is useful in the UsiXML model-driven engineering process, but we think the context model will be subject to change in the future. In the storyboard, we think the combination of the graphical depiction, showing the situation of the users, the pre-defined elements in the meta-model and the arbitrary tags that are allowed, provide a powerful means to describe the context-of-use as specific as possible. For "hidden" context properties, such as the platform specification of a device, the COMuICSer tool provides dialogs to fill in this data. There are separate dialogs for adding the device profile, the persona descriptions and describe activities. Notice the UsiXML mapping model does not provide explicit support for a changing context over time, so we need to generate multiple UsiXML context model instances. The UsiXML

event definition links to a context in which the event is executed, so this could be used to describe changes in context too. Using the storyboard meta-model we can now construct a mapping to generate parts of the context model:

For each scene instance in the storyboard model:

- Each *Persona* element will be mapped on a new *userStereoType* element, as it is the closest to a persona UsiXML currently offers.
- Each *Device* element will be mapped on new *hardwarePlatform* and *softwarePlatform* elements (including properties such as *screenSizeAspect*, *osName*,...).
- Each *tag* that describes the *surroundings of the user and matches*:
 - *Noise* generates the *isNoisy* property and set it to *true*.
 - *Light* generates the *lightningLevel* property.
- If the scene description indicates the user is in a stressful environment generate the *isStressing* property and set it to *true*.

Now that we have a set of context models, each of them can be related by a set of tasks that are grouped per scene, of which the generation was explained in the previous subsection. To make this set of models useful and usable, there still needs to be tool support that allows a designer to specify user interfaces according to the context of use. One example of such a tool is Gummy, presented in [9], but has only rudimentary support for UsiXML. Another candidate tool could be SketchiXML [3].

This approach makes it more convenient to specify the context, since most information is already contained in the storyboard. We believe this approach could also lead to a more complete definition of context, created from experience rather than using a pre-defined description. The combination of images and tags makes there are little to no limitations in defining context of use.

TRANSFORMATIONAL DEVELOPMENT TOOLS

We created tool support for progressing from the storyboard models toward UsiXML models. In spirit of openness of our approach, the Eclipse setup in which the tools are embedded is provided as free software and can be downloaded from the project website that can be found at the following URL: <http://research.edm.uhasselt.be/kris/research/projects/StoryBoardML>.

Our tool allows defining Model-to-Model transformation rules by creating mappings between entities contained in the source and target meta-models. Rule definitions are essentially data-driven. Each definition specifies a set of element types from the source model as input and an element type from the target model as output. The rule can be applied to instances of the source model and create the target elements according to this input. A screenshot of the tool is shown in Figure 6. It shows a mapping editor that allows

defining a mapping rule. We consider this a proof of concept for showing the capabilities of our approach.

According to [4] our solution can be classified as an approach that uses a target-oriented organizational structure of unidirectional rules, and follows a strategy of deterministic rule application. Our solution also supports reuse of rules by means of inheritance or composition. The tool is able to generate method stubs in Java, which can be used later on to program all the details of data passing between entities. I.e. the transformation mechanism can be further refined in the method stub and techniques such as property value transfer, type lookup, type casting, data trimming and decoration of elements are all available. The mapping tool is mainly a starting point for further refinement and definition of complex transformations. Our aim is not to provide a full set of transformations, rather to enable a developer to define transformations on a concrete level when facing a set of high-level transformation rules that need to be enforced.

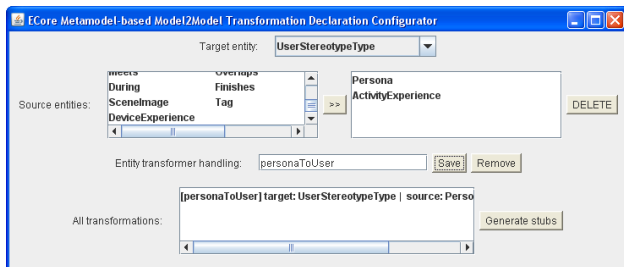


Figure 6. Mapping editor to define a target-oriented unidirectional mapping rule for generation partial UsiXML models from a storyboard model.

Our tool support is built on top of the Eclipse environment and makes extensive use of the Eclipse Modeling Framework Core (EMF core or ecore) [14]. Thanks to EMF, developers who create new rules can also use automatically generated code that supports manipulation and persistence of the instances of any meta-model. To provide smooth integration of COMuICSer and parts of UsiXML, we created an ecore version of storyboard meta-model. The UsiXML schemes were also converted in pure ecore models (model descriptions in XMI) so they can be easily used within the Eclipse environment. This has two major advantages: first, one all the MDE tools that are available in Eclipse can be used (querying, transformation, validation) and second, editing tools can be automatically generated for ecore compliant models. The Eclipse Modeling Framework can be used through a powerful API, which has allowed us to rapidly generate code templates based on our ecore conversions of the UsiXML models.

CONCLUSION

This paper presented a transformational approach to integrate storyboarding with various UsiXML models. For this purpose we created a storyboarding meta-model that serves as the starting point for applying the transformations. In

contrast to the storyboard meta-model definition, the transformations themselves are not hard-wired in our approach. By using the features of the Eclipse Modeling Framework a developer can edit custom mapping rules using a simple interface. A custom set of UsiXML models using pure XMI was generated to enable smooth integration in the Eclipse environment. These mappings can be applied to the storyboard model to generate partial UsiXML models or check consistency with the storyboard when editing the UsiXML models. The work presented here is still work in progress. The tools and models described in this paper are fully functional, though they are not integrated to the point they allow full consistency checks between UsiXML and other models.

ACKNOWLEDGMENTS

This work is supported by the FWO project *Transforming human interface designs via model driven engineering* (G. 0296.08) and IWT project *AMASS++* (SBO-060051).

REFERENCES

1. Allen, J. F. Maintaining knowledge about temporal intervals. *Communications of the ACM* 26,11 (1983), pp. 832–843.
2. Carroll, J.M. *Making use: scenario-based design of human-computer interactions*. MIT Press, Cambridge (2000).
3. Coyette, A., Kieffer, S., and Vanderdonckt, J. 2007. Multi-Fidelity Prototyping of User Interfaces. In *Proc. of 11th IFIP TC 13 Int. Conf. on Human-Computer Interaction INTERACT'2007* (Rio de Janeiro, September 10-14, 2007). Lecture Notes in Computer Science, Vol. 4662. Springer-Verlag, Berlin (2007), pp. 149–162.
4. Czarnecki, K. and Helsen, S. Classification of Model Transformation Approaches. In *Proc. of OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture* (Anaheim, 2003).
5. Dow, S., Saponas, T. S., Li, Y., and Landay, J. A. External representations in ubiquitous computing design and the implications for design tools. In *Proc. of the 6th Conf. on Designing Interactive systems DIS'2006*. ACM Press, New York (2006), pp. 241–250.
6. Furtado, E., Furtado, V., Sousa, K., Vanderdonckt, J., and Limbourg, Q. KnowiXML: A Knowledge-Based System Generating Multiple Abstract User Interfaces in UsiXML. In *Proc. of 3rd Int. Workshop on Task Models and Diagrams for user interface design TAMODIA'2004* (Prague, November 15-16, 2004). Ph. Palanque, P. Slavik, M. Winckler (Eds.). ACM Press, New York (2004), pp. 121–128.
7. Johansson, M. and Arvola, M. A case study of how user interface sketches, scenarios and computer prototypes structure stakeholder meetings. In *Proc. of the 21st British CHI Group Annual Conf. on Human-Computer In-*

- teraction *BCS-HCI '2007* (Swinton, 2007). British Computer Society, Cambridge (2007), pp. 177–184.
8. McCloud, S. *Understanding Comics: The Invisible Art*. 1994.
 9. Meskens, J., Vermeulen, J., Luyten, K., and Coninx, K. Gummy for multi-platform user interface designs: shape me, multiply me, fix me, use me. In *Proc. of the ACM Working Conf. on Advanced Visual Interfaces AVI'2008*. ACM Press, New York (2008), pp. 233–240.
 10. Montero, F. and López-Jaquero, V. IdealXML: An Interaction Design Tool-A Task-Based Approach to User Interfaces Design. In *Proc. of 6th Int. Conf. on Computer-Aided Design of User Interfaces CADUI' 2006* (Bucharest, 6-8 June 2006). Chapter 20, Springer-Verlag, Berlin (2006), pp. 245–252.
 11. Mori, G., Paternò, F., and Santoro, C. CTTE: Support for Developing and Analyzing Task Models for Interactive System Design. *IEEE Trans. on Software Engineering* 28, 8 (2002), pp. 797–813.
 12. Pruitt, J. and Adlin, T. *The Persona Lifecycle: Keeping People in Mind throughout Product Design*. Morgan Kaufmann (2006).
 13. Schmidt, D.C. Model-Driven Engineering. *IEEE Computer* 39 (2006).
 14. Steinberg, D., Budinsky, F., Paternostro, M., and Merks, E. EMF: Eclipse Modeling Framework, 2nd Edition. Addison-Wesley Professional, Eclipse Series (2008).
 15. Truong, K. N., Hayes, G. R., and Abowd, G.D. Storyboarding: an empirical determination of best practices and effective guidelines. In *Proc. of the 6th Conf. on Designing Interactive systems DIS'2006*. ACM Press, New York (2006), pp. 12–21.
 16. Vanderdonckt, J. Model-Driven Engineering of User Interfaces: Promises, Successes, and Failures. In *Proc. of 5th Annual Romanian Conf. on Human-Computer Interaction ROCHI'2008* (Iasi, September 18-19, 2008), S. Buraga, I. Juvina (eds.). Matrix ROM, Bucarest (2008), pp. 1–10.
 17. Vanderhulst, G., Luyten, K., Coninx, K. Photo-based User Interfaces: Picture it, Tag it, Use it. In *Proc. of the 4th Int. Workshop on Ontology Content (OnToContent'09)* (Vilamoura, November 1-6, 2009).

Comparative Analysis of Collaborative Approaches for UsiXML Meta-Models Evolution

Mohamed Boukhebouze, Amanuel Koshima, Philippe Thiran, Vincent Englebert

PReCISE research center, University of Namur, Belgium,
{mohamed.boukhebouze, ammanuel.koshima, philippe.thiran, vincent.englebert}@fundp.ac.be

ABSTRACT

The UsiXML project is dedicated to define a user interface description language called UsiXML. This language specifies the different abstract levels of a user interface by using set of model according to Model-driven engineering (MDE). In the UsiXML project, several industrial and academic members work in collaborative manner to evolve meta-models of UsiXML models. Several collaborative approaches can be applicable. For instance, the control of meta-models modifications can be centralized or not. In the paper, we aim at proposing the most appropriate approach for the UsiXML meta-models evolution. To achieve this, we present a comparative study of the different collaborative design approaches.

Keywords

UsiXML, Meta-Model evolution, Collaborative design, Centralized approach, Decentralized approach, control of modification.

INTRODUCTION

UsiXML (USeR Interface eXtensible Markup Language) offers a common way to specify a User Interface (UI) independently of any target programming language that is intended to implement it [1]. This language allows describing an UI in four main levels of abstraction: task & domain, abstract UI, concrete UI, and final UI. On the basis of these four levels, UsiXML proposes a set of models. For each model, UsiXML defines their meta-model by using the UML Class diagram notation. Each meta-model can then be expressed in XML (XSD [2], XMI [3]), or in OWL [4]). All these notations are required by standardization organizations like OMG or W3C.

The evolution of the UsiXML meta-models is carried out by an ITEA project involving industrial and academic members [5]. Evolving UsiXML meta-models requires a collaborative system that supports interactive work between members of the ITEA project. This collaborative system is intended to manage the UsiXML meta-models evolution by:

- Offering a repository of the UsiXML meta-models,
- Managing the members and their roles,
- Supporting the evolution of UsiXML meta-models in collaborative way; which involves managing concurrent modifications.

Such a system can be designed in different ways. First, the meta-models can be stored in a *centralized repository* (a unique repository for all the members) or in *decentralized repositories* (a local repository per member). Secondly, the modifications can be either *controlled* by a central authority (called the owner) or *delegated* to every member. In the first case, the modifications issued by the members are sent to the owner of a meta-model who can commit or reject them. In the second case, each member has the control of his own version and can integrate the modifications of the peer-designers in his version. The convergence of these meta-models to a unique consolidated version is the result of a workgroup communication. In both cases, a reconciliation process is always required since syntactic or semantic discrepancies have to be resolved [6].

Each of these approaches has its strengths and its weaknesses. Let us mention the most relevant ones. On one hand, the centralized approach allows controlling the reconciliation process by an owner but this exclusive role can lead to slow down the modifications frequency. On the other hand, the decentralized approach can support a higher rate of modifications but it requires a more complex reconciliation process.

In this paper, we propose a comparative study of the different collaborative design approaches. The study is aimed at proposing the most appropriate approach for the UsiXML meta-models evolution.

The rest of this paper is organized as follows. In Section 2, we describe the different approaches for the collaborative design of meta-models. In Section 3, we conduct a comparative study on these approaches based on a different comparison criterion. In Section 4, we apply our comparative study on the collaborative design of UsiXML meta-models. We conclude this paper by discussing the most appropriate approach for UsiXML.

COLLABORATIVE META-MODEL EVOLUTION APPROACHES

According to Schmidt et al. [6], *collaborative work* is characterized by a mutual interdependence of work among multiple actors to produce a specific product or service. *Interdependence work* means that actors share the same field of work (i.e. document, codes of a program). *Mutual work* means that the work of one actor relies on the quality and timeliness of other actors.

By analogy with the Schmidt's definition, *collaborative meta-model evolution within a project* is characterized by a mutual interdependence of meta-model evolution among multiple project members. All members share a common meta-model by using a central repository or a local one and they interact with each other by modifying the version of their meta-model.

For this reason, four approaches can be considered:

- Centralized approach with modification control;
- Centralized approach without modification control;
- Decentralized approach with modification control;
- Decentralized approach without modification control.

Note that these approaches define how to design, store, and evolve the meta-model. In the following sections, we detail these approaches.

Centralized approach with modification control

In this approach, we consider that there is one central meta-model repository. The meta-model has its owner who is the only member that can modify it. The other members can participate in the design of a meta-model by issuing a modification request to its owner. It is under the responsibility of the owner to commit or not the modification requests. As such, the owner controls the modification of the meta-model repository (Figure 1).

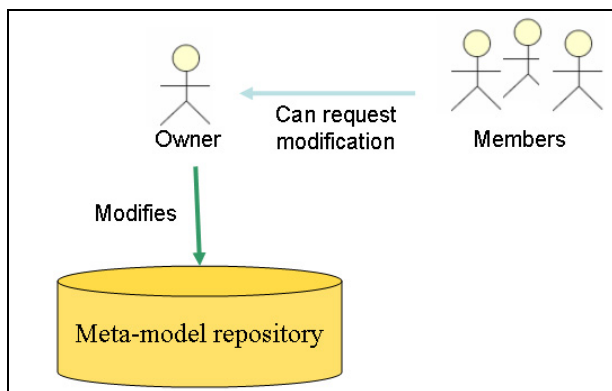


Figure 1. Architecture of centralized approach with modification control.

Centralized approach with modification control implements an administrative hierarchy, like any classical design, project by using a central authority, which is called owner. This owner can be the leader of project or the expert designer. However, when the design project is large, the dynamism of modifications can become slow. This is due to the fact that a large project implies a high number of members. Each member can request a modification. This can imply a considerable time to handle all the request modifications by the owner. Finally, note that this approach can be implemented on the top on CVS system as proposed in [7]. In CVS system, the changes are committed to a master repository (a owner in our case) before they are propagated to other users (members in our case).

Centralized approach without modification control

This approach considers that there is one central meta-model repository that can be modified by any project member at any time. This assumes that a modification can overwrite another modification without any control (Figure 2). This means that any member can freely modify meta-model elements without informing the other members. Moreover, the repository stores only the recent modifications as they always overwrite the previous modifications on a same element.

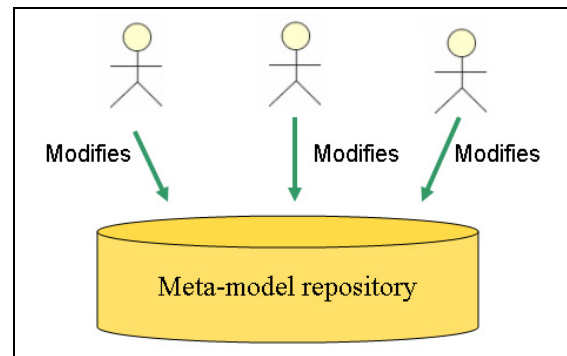


Figure 2. Architecture of centralized approach without modification control

The absence of modification control lets designers modify meta-models faster in comparison with the above approach. However, they are continuously being modified so that it is difficult to get a stable version. Indeed, obtaining a stable version is problematic even if there exists a methodological agreement between members on how to cooperate. A way to circumvent this problem is to partition the meta-model into distinct areas of interest. As such, only one member can modify one area. Consequently, conflicts are avoided during concurrent accesses of views. However, partitioning of meta-model into different distinct areas of interest is a challenging issue.

Finally, note that this approach is quite suitable for the collaborative text editing where user can create and edit documents while collaborating in real-time with other users (e.g., Google document - <http://googledocs.blogspot.com/>).

Decentralized approach with modification control

In this approach, we consider that all the members have their own local repository. Each member is free to access or modify his local repository. As such, a member can create, read, update or delete (CRUD) an element of local meta-model. In the decentralized approach, we suppose that all the CRUD actions on a local repository are logged and stored in a local journal. This journal keeps traces of the local meta-model modifications [8]. The local journals are exchanged between members via a coordinator (Figure 3). The coordinator has the responsibility to control the interactions among members based on different policies such as:

- *Centralized consolidation*: the coordinator aggregates and consolidates the modifications sent by the mem-

bers. The consolidated modifications are then sent to all the members so that they can modify their local copies. As such, their local meta-model repository can be consistent each other.

- *Decentralized consolidation*: the coordinator propagates each modification (one by one) to members (without aggregation and consolidation). This means that modifications are propagated in FIFO order to members. It is under the responsibility of each member to consolidate his meta-model wrt the propagated modifications. Local members cannot reconsider them. Hence, their current modifications can be overwritten and invalidated by the propagated modifications.

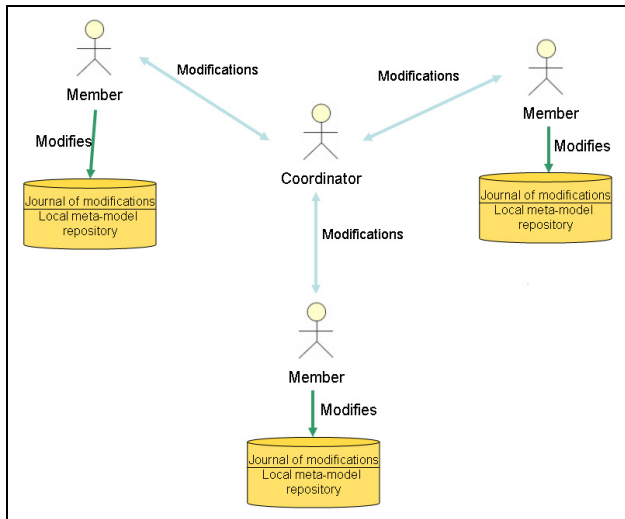


Figure 3. Architecture of decentralized approach with modification control.

This approach gives a freedom to members in terms of time (work at any time) and space (work in a local repository). In the second policy, a member who firstly sends modifications could be determined how the meta-model is evolving. As result, there might be competition among members to become the first one to send modification. As such, this could hamper the collaborative work.

Decentralized approach without modification control

In this approach, we consider that all the members can freely modify the meta-model repository. As the previous approach, the CRUD actions are stored in a local journal. Since there is no coordinator to control meta-model modifications, a member sends his local journal (modifications) to all the other members (Figure 4). Afterwards, each member votes yes/no for the proposed change; if the majority of members accept the modification, it is implemented in all local repository of each member. If not, the modification is rejected. Like the third approach, this approach ensures freedom in time and space for members. However, the reconciliation process is more difficult than all the above approaches. This is due to the fact that it is not easy to get consensus about the suggested changes among all the

members. Finally, note that this approach can be implemented by using a peer-to-peer architecture as it is proposed by Mougenot *et al.* [9]. However, unlike the decentralized approach without modification control, Mougenot *et al.* propose a reconciliation strategy based on an order computed by Lamport clocks.

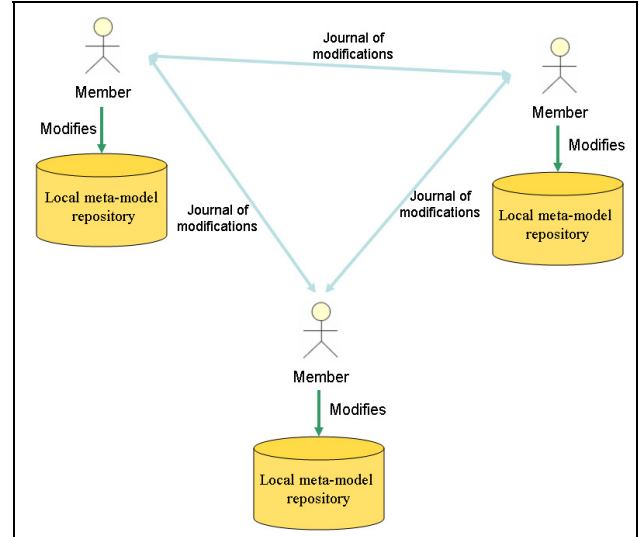


Figure 4. Architecture of decentralized approach without modification control.

COMPARATIVE ANALYSES OF COLLABORATIVE META-MODEL EVOLUTION APPROACHES

In the previous section, we presented several approaches for supporting the collaborative editing of meta-models. Each of them addresses specific needs that we now compare according to three categories.

Project

Modification dynamism: a project may accept a slow rate of modifications (in terms of days), or may require a faster rate (in terms of minutes).

Project size: a project can involve a small working group (e.g. open space or department of an enterprise) or a medium working group (e.g. enterprise or organization) or a large working group (e.g. multi-enterprise collaboration).

Meta-model

Expression of meta-model: a meta-model can be expressed formally by using a formal language (e.g. XMI, XSD) or informally by using a bitmap image or the natural language.

Links between meta-models: the meta-models can be linked each other by sharing some elements or by defining binding between meta-models. In this paper, we suppose that the meta-models are not related together. This hypothesis is restrictive, as it is not confirmed in the UsiXML realm. Nevertheless, as the links between the meta-models have not been so far formally elicited, we can consider that problem as out of scope of this paper.

		Project criterion		Meta-model criterion	Implementation criterion		
		Dynamism of modifications	Size of project	Expression of meta-model	Request of modification	Role of member	Reconciliation
Centralized approaches	With control	Slow	Large	Formal or informal expression	Need requests	Role based	A priori
	Without control	Fast	Small	Formal or informal expression	Optional	Not role based	No reconciliation
Decentralized approaches	With control	slow/medium	Large	Formal expression	Optional	Role based	A priori
	Without control	Fast	Medium	Formal expression	Optional	Not role based	Posteriori

Table 1. Comparative analyses of collaborative meta-model evolution approaches.

Implementation criterion

Request for modification: requests for modifications can be expressed in a formal or informal way. This depends on the language used to define the meta-models. A formal request could be expressed as “patch file”, OCL statements, etc. Informal requests are just textual descriptions. The first form enables an automation of the reconciliation process while the second one cannot be automated.

Role of member: some implementations require assigning roles (owner or not) to members to let the cooperative scenario go off.

Reconciliation: the reconciliation process can be applied *a priori* to avoid divergent versions. It can also occur *a posteriori* to solve the problem once it is observed. These approaches are also called resp. pessimistic and optimistic [10]. Note that it is also possible to have no *reconciliation* if it is not required to have a common stable version. In this case, we assume that each member maintains his own version of the meta-model in a consistent way. The member is free to integrate or not the modifications of the other members [11].

Table 1 depicts a summary of this criterion. This table can be used as a decision table based on the nature of the project. As explain above, collaborative approaches with control cannot support a fast dynamism of modifications. This is due to the fact that these approaches require a central point that handles the reconciliation process. On the contrary, the collaborative approaches without control support a fast dynamism of modifications. However, they cannot be applicable to a large project because it is difficult to achieve a consensus among members about modification.

The collaborative approaches with control can support a large project. Indeed, these approaches implement a central authority that can be the leader of project or the expert designer who manages modifications. Hence, large projects could be easily managed.

According to Table 1, the decentralized approaches require a formalized meta-model since these approaches use it to exchange journals of modifications among members. Moreover, formal expression of meta-models could avoid conflicts and confusions. On the top of that, the formalization of meta-model helps to automatically generate the CRUD journal when a meta-model is modified. Whereas centralized approaches could express meta-model into formally or informally.

Finally, the aforementioned controlled approaches implement *a priori* reconciliation to ensure that the modification is consistent with the central point. On the other hand, the above approaches without control use *posteriori* reconciliation. This is due to the fact that there is no central point that evaluates the proposed modifications.

USiXML PROJECT

In this section, we apply our comparative study on UsiXML project. UsiXML is a User Interface Description Languages (UIDL) that uses a Model-Driven Engineering (MDE) for specifying a UI at an implementation-independent level [12]. The UI specifications are usually specified in different models: each one denotes a facet of the interface characteristics. The complete set of UsiXML models comprises [13]: (1) *Task model*: it describes the interactive tasks as viewed by the end user interacting with the system; (2) *Domain model*: it is a description of the classes of objects manipulated by a user while interacting with a system; (3) *Abstract User Interface (AUI) model*: it is a model that represents a canonical expression of the renderings and manipulations of the domain concepts and functions in a way that is independent of any interaction modality and computing platform; (4) *Concrete User Interface (CUI) model*: it is a model that allows the specification of the presentation and behavior of a UI with elements that can be perceived by the users; (5) *Mapping model*: it contains a series of related mappings between models or elements of models; (6) *Transformation model*: it contains a

set of rules enabling the transformation of one specification (at a certain level of abstraction) into another or to adapt a specification for a new context of use.

For each model, UsiXML defines their meta-model. These meta-models are upgraded in a collaborative way by all members. In the next section, we will detail the characteristic of the UsiXML project and UsiXML meta-model.

UsiXML project characteristics

UsiXML project is a large project (Section 2.1), because it involves more than twenty-eight industrial partners (e.g., DefiMedia¹, Thales Research & Technology², Telefonica³) and academic partners (e.g., University of Namur⁴, Université catholique de Louvain⁵, University of Grenoble⁶, University of Valencia⁷) [5]. One of the goals of the UsiXML project is to propose the first stable versions of all the UsiXML meta-models. This requires the all the UsiXML meta-models must be reviewed, updated, and improved by the members in order to address the multiple challenges posed by model-driven engineering of user interfaces [16]. As such, the project requires supporting a medium/fast rate of modifications.

UsiXML Meta-models

UsiXML meta-models are expressed with the UML class diagram notation [14] by using CASE tools that can export these models in a formal standard exchange format (XMI). The Figure 5 shows an example of the meta-model of task expressed in UML. Besides the UML class diagram notation, each UsiXML meta-model comes with free textual annotations aimed at eliciting the element semantics (for instance, the semantic definition of the task class is: “*task is the basic structure that composes the task model. Tasks are activities that have to be performed to reach a goal*” [13]).

DISCUSSION

According to Table 1, the collaborative UsiXML meta-model evolution can be designed using a centralized approach with modification control. This is motivated by the fact that this approach supports a large project. In addition, the UsiXML project requires that a scientific leader is nominated for the control of the meta-model definition. In a centralized approach with modification control, an administrative hierarchy can be implemented by considering the leaders as owners. In this way, the divergent versions are avoided by ensuring the reconciliation process is done a priori way. However, such an approach can lead to a high

rate of modifications. Moreover, the large size of the UsiXML implies many frequent requests of modifications. This could lead up the owner to be overloaded.

The second approach that can be applied to UsiXML is the decentralized approach with modification control. Indeed, this approach is compliant with projects of large size. In addition, it offers a fast rate of change especially if we consider that the coordinator can be automated [15]. This automated coordinator helps to exchange the journal of modifications between members project. However, the reconciliation process can be complex in comparison with the centralized approach with modification control. This is due to the fact that each member can modify his own local meta-model. This requires a reconciliation process in which a coordinator controls the different interactions between members. This process can be complex when the number of members is high.

Finally, note that the two approaches without modification control cannot be applicable to UsiXML project because these approaches cannot support a large project.

ACKNOWLEDGMENTS

We gratefully acknowledge the support of the ITEA2 Call 3 UsiXML project under reference 20080026 and the support of the Région Wallonne.

REFERENCES

1. UsiXML, User Interface eXtensible Markup Language, Available online at <http://www.usixml.org>, 2010.
2. W3C, XML Schema Part 0: Primer Second Edition, in REC-xmlschema-0-20041028, 2004.
3. OMG, XMI Mapping Specification, v2.1.1, formal/07-12-0, 2007.
4. W3C, OWL Web Ontology Language 2.0, in REC-owl-features-20040210/2004.
5. ITEA2, UsiXML Full Project Proposal, December 23, 2009.
6. Schmidt, K. and Simone, C. Coordination Mechanisms: Towards a Conceptual Foundation of CSCW System Design. Computer Supported Cooperative Work. In the Journal of Collaborative Computing 5 (1996), pp. 155-200.
7. Altmanniner, K., Seidl, M., and Wimmer, M. A Survey on Model Versioning Approaches. Int. Journal of WIS 5,3 (2009) pp. 271–304.
8. Constantin, C., Englebert, V., Thiran, P., A Reconciliation Framework to Support Cooperative Work with DSML, in Proceedings of the First International Workshop on Domain Engineering held in conjunction with CAiSE'09 Conference, collection CEUR-WS.org , volume 457.

¹ <http://www.defimedia.be/>

² <http://www.thalesgroup.com/>

³ <http://www.telefonica.com/>

⁴ <http://www.software-engineering.be/>

⁵ <http://www.uclouvain.be/>

⁶ <http://www.ujf-grenoble.fr>

⁷ <http://www.uv.es/>

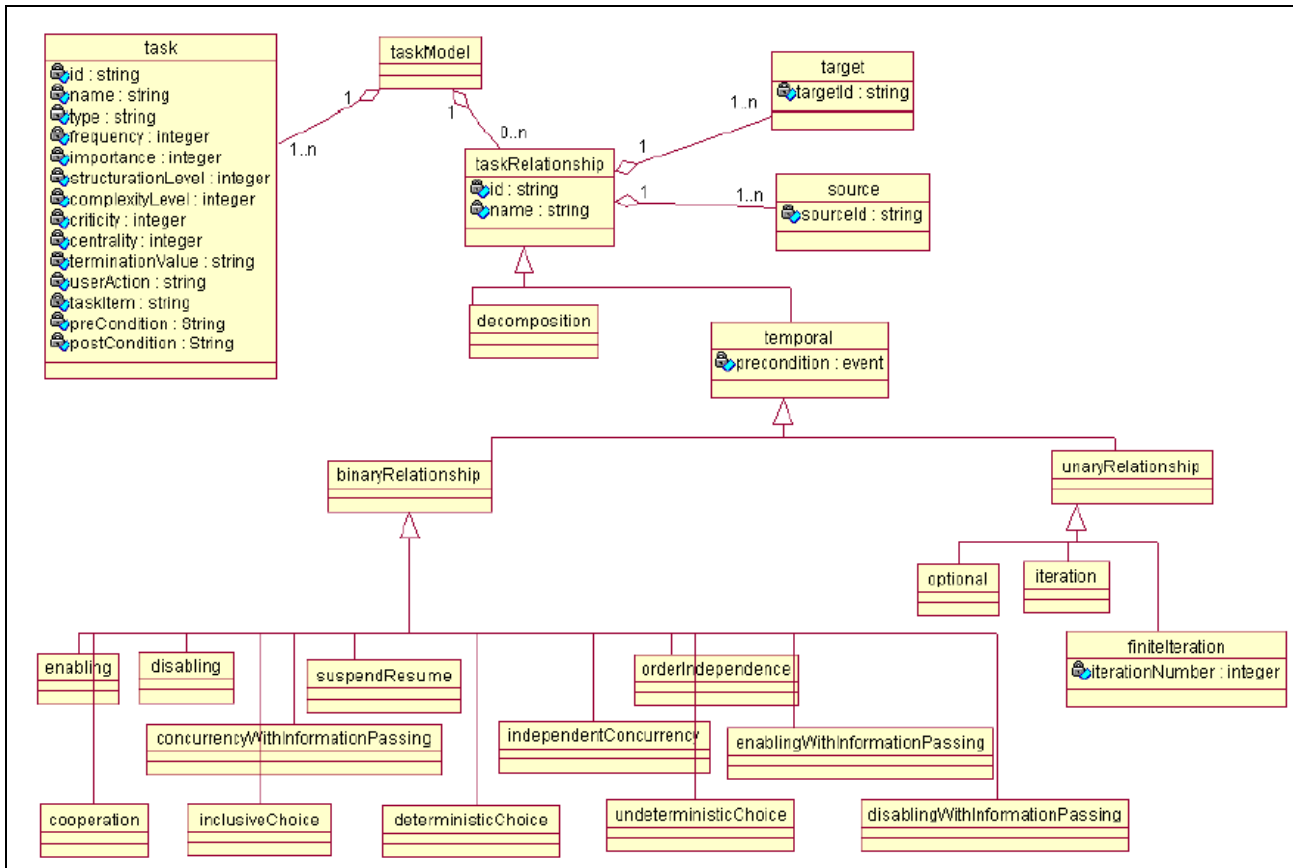


Figure 5. Meta-model of task.

9. Mougenot, A., Blanc, X., Gervais, M.P.: D-Praxis : A Peer-to-Peer Collaborative Model Editing Framework. In *Distributed Applications and Interoperable Systems 2009*, Volume 5523/2009, 16-29 P.
10. Uwe, M., Johann, S. Computer-Supported Cooperative Work, In Book isbn 9783540669845, Oxford University Press, 2000.
11. Grebici, K., Yee, M., Goh, S., Zhao, B., McMahon, C. Information Maturity Approach for the Handling of Uncertainty within a Collaborative Design Team. In *IEEE CSCWD 2007*.
12. Stanculescu, A., Limbourg, Q., Vanderdonckt, J., Michotte, B., and Montero, F. A Transformational Approach for Multimodal Web User Interfaces based on UsiXML. In *Proc. of 7th Int. Conf. on Multimodal Interfaces ICMI'2005* (Trento, 4-6 October 2005). ACM Press, New York (2005), pp. 259-266.
13. UCL, UsiXML V1.8 Reference Manual, February 2007.
14. Object Management Group, Unified Modeling Language 2.0. In formal/2007-02-03, 2007.
15. Saeki, M., Oda, T. A Conceptual Model of Version Control in Method Engineering Environment. In *CAiSE Short Paper Proceedings 2005*.
16. Vanderdonckt, J. Model-Driven Engineering of User Interfaces: Promises, Successes, and Failures. In *Proc. of 5th Annual Romanian Conf. on Human-Computer Interaction ROCHI'2008* (Iasi, September 18-19, 2008), S. Buraga, I. Juvina (eds.). Matrix ROM, Bucharest, 2008, pp. 1-10.

Designing Workflow User Interfaces with UsiXML

Josefina Guerrero Garcia, Jean Vanderdonckt

Université catholique de Louvain, Louvain School of Management, Place des Doyens, 1
B-1348 Louvain-la-Neuve (Belgium)
{josefina.guerrero, jean.vanderdonckt}@uclouvain.be

ABSTRACT

Supporting business processes with the help of workflow systems is a necessary prerequisite for many companies to stay competitive. An important task is the specification of workflow, i.e. these parts of a business process that can be supported by a computer system. This paper is about the definition and development of user interfaces for workflow information systems. XML-based user interface description languages express various aspects of the user interfaces, including the abstract and concrete elements of the user interface, the tasks to be performed by the users, and the user interface dialogue. We have developed a framework for expressing workflow aspects, and using UsiXML for rendering the user interfaces.

Categories and Subject Descriptors

D2.2 [Software Engineering]: Design Tools and Techniques – *Modules and interfaces; user interfaces*. D2.m [Software Engineering]: Miscellaneous – *Rapid Prototyping; reusable software*. H.1.2 [Information Systems]: Models and Principles – *User/Machine Systems*. H5.2 [Information interfaces and presentation]: User Interfaces – *Prototyping; user-centered design; user interface management systems (UIMS)*.

General Terms

Design, Experimentation, Human Factors, Verification.

Author Keywords

Workflow, user interface description language, task model, organizational model.

INTRODUCTION

In every company there are procedures aimed at providing increased efficiency, consistency, and quality. Among them are concepts such as business process and workflow management. Their objective is to clarify, improve, and coordinate complex activities and interactions. The goal of a business process is to deliver specific products or services while ensuring the organization's overall interests. Examples for business processes are processing purchase orders over the phone, processing insurance claims, and processing a car registration. The objective of *workflow management* is to support the execution of business processes. The parts of a business process that can be supported by a computer system are called *workflows*. A workflow management system (WFMS) is a computer system that implements workflow management functionality.

In this context, an important aspect to consider is the design of the user interfaces of the system. In the area of workflow

there has been increasing interest in developing methods and languages to design user interfaces, as the work of [1] and [12]. In Human-Computer Interaction, a user interface description language express various aspects of the user interfaces, including the abstract and concrete elements of the user interface, the tasks to be performed by the users and the user interface dialogue. Nevertheless, there are several opportunities to improve on these languages; in this paper, we will explore a systematic way to design the user interfaces for a workflow information system.

The workflow model defines what processes and tasks need to be fulfilled and their possible ordering, hence the workflow model is a “framework” for creating task models; hence the task model is suitable for designing user interfaces.

The remainder of this paper is structured as follows: Section 2 presents an overview of the related work. Section 3 introduces a model for specifying the workflow, and then we describe a method for designing workflow user interfaces based on UsiXML (Section 4). Section 5 presents a case study and the software supporting the method. Section 6 summarizes our work deriving conclusions and future work.

RELATED WORK

In order to provide a meta-model of workflow concepts which are related to user interface issues, this section presents an overview of the existing literature. It has been divided in three related parts: the workflow characteristics, the task model concepts, and the user interface description languages to generate user interfaces. There is a plethora of methods, tools, and models in the literature, including open source systems and vendors systems, research into all of them is not the objective of this section, and we just present some of them.

Workflow theory

Workflow models focus on how work is done to accomplish organizational goals; it defines how task, information, and documents are passed from one participant to another in the organization [28]. According to [16], the essential workflow characteristics are: *tasks/activities* that are performed by (role-playing) *persons*, using supporting *tools* that give access to a variety of *shared information resources*. Due to the importance of workflow nowadays, several workflow notation, descriptions, and software support have been proposed to design and specify it.

Some notations used are: *Petri Nets* [22] as a technique for modeling and analyzing processes; *Statechart Diagrams* [27] is a graph that represents a state machine describing the response, of an object of a certain class, to the receipt of outside stimuli; *Business Process Modeling Notation* (BPMN) [18] is a standardized graphical notation for drawing business process in a workflow; and *UML Activity Diagrams* [4] provides a view of the behavior of a system by describing the sequence of actions in a process. UML activity diagrams are intended to model both computational and organizational processes. Currently, several models and design methods support the development of complex workflow-based applications providing notations for business process and including tool support. Some of them are: *The Progression Model* [21] has incorporated some of the managing concepts of workflow to increase the flexibility in information systems; *Microsoft Windows Workflow Foundation* (WWF) [5] is an extensible framework for developing workflow solutions on the Windows platform. It provides a single, unified model to create end-to-end solutions that span categories of applications, including human workflow and system workflow; *WebSphere® MQ Workflow* (IBM) [10] supports long-running business process workflow as they interact with systems and people. Automates and tracks business process in accordance with business design; *Workflow on Intelligent and Distributed database Environment* (WIDE) [3] defines an advanced conceptual model for describing both the flow of activities and the organizational environment in which these activities are performed; *Business Process Visual ARCHITECT* (BP-VA) [26] is a visual modeling tool that provides the most extensive support for BPMN.

Due to the large amount of existing workflow products we came to a point where it is very difficult to analyze and compare their capabilities on a common scheme. However, they can be gathered in a collection of workflow patterns that provide the basis for an in-depth comparison of commercially available workflow systems. *Control-flow patterns* [23] identified useful basic routing constructs such as sequence, parallel split, synchronization, exclusive choice. From a data perspective, there is a series of characteristics that occur repeatedly in different workflow modeling paradigms. *Workflow data patterns* [19] are aimed at capturing the various ways in which data is represented and used in workflows. *Workflow resource patterns* [20] correspond to the manner in which tasks are allocated to resources, that is any entity that is capable of achieving some work unit.

Task theory

A common definition for a task is “an activity performed to reach a certain goal” [25]; task models play an important role because they indicate the logical activities that an application should support to reach user’ goals. While the purpose of task analysis is to understand what tasks should be supported and what are their related attributes, the aim of task modeling is to identify more precisely the relationships among such tasks. Task models are explicit represen-

tations of user tasks that can help support certain rigorous forms of task analysis. In [6] we discuss some well-known and widely used task notations, examining which characteristics they exhibit and which attributes they cover. Task models play an important role in UI design because they support the systematic representation of the user activity as opposed to the system activity.

User interface description languages theory

Model-based user interface design is intended to assist in designing UIs with a more formal computer supported methodology; a user interface description language (UIDL) is intended to capture the details of what a user interface could or should consist. There are solutions for developing UIs that are based in eXtensible Mark-up Language (XML). There is a plethora of user interface description languages that are widely used, with different goals and different strengths. On one hand we have software vendors UIDLs and, on the other hand, there are free license UIDLs to use. A review of user interface description languages was produced [9] that compares a significant selection of various languages addressing different goals, such as multi-platform user interfaces, device-independence, and content delivery. UsiXML [12] is a XML-compliant markup language capturing the essence of what a UI is or should be independently of physical characteristics. It has been selected as the UIDL to be used in the remainder of this work because of its capabilities of extensiveness, availability, central storage of models, and its model-driven approach.

A WORKFLOW MODEL

After reviewing the available literature, we can identify the elements needed to model the workflow. We propose a model called *FlowiXML* (user interfaces to workflow based on *UsiXML*) [7] to support workflow information systems, which are advocated to automate processes based on requirements and processes of the organization. The methodology applies to:

- 1) Integrate human and machines based activities, in particular those involving interaction with IT applications and tools.
- 2) Identify how tasks are structured, who perform them, what their relative order is, how they are offered or assigned, and how tasks are being tracked. Figure 1 presents an overview of the meta-model. *FlowiXML* is divided in different models.

Workflow model

The workflow model describes how the work in organization flows by defining models of: process (what to do?), tasks (how to do it?), and the organizational structure (where and who will perform it?). Also, it contains information relevant to controlling and coordination of the execution of its constituent tasks (e.g. required skills, possible actors, and execution requirements). A workflow model has at least one process and each process has at least two tasks. Workflows are described with a *name* and *ID*.

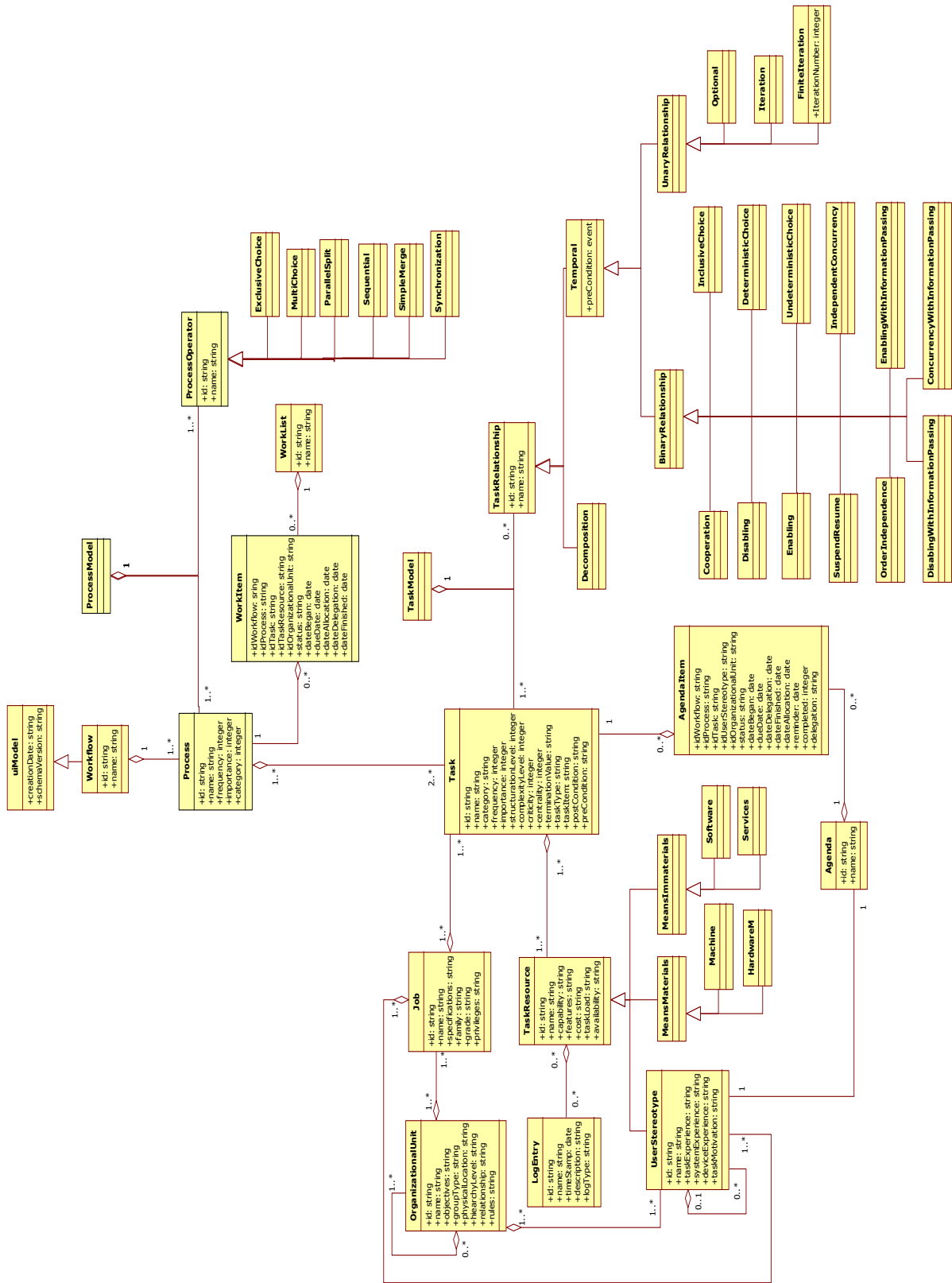


Figure 1. Workflow meta-model.

Process model

The definition of a process indicates the ordering of tasks in time, space, and resources. The representation of this model is an adaptation of the Petri Nets notation proposed in [22] and is compatible with the workflow resource patterns proposed in [20]. The process model is composed of:

- *Process*. A process consists of a number of tasks and a set of relationships among them. The definition of a process indicates which tasks must be performed and in what order.
- *Process operators*. They indicate the different ways in which the processes could be executed.
- *Work list*. It is a list of work items related with a given workflow.
- *Work item*. It is the representation of the work to be processed.

Task model

Task models are used to collect the requirements of a workflow system. A task is an activity that has to be performed by users to reach a given goal related to the processes. Introducing task models description to the workflow models corresponds, but is not limited, to the following reasons:

- Task models describe, opposed to process models, end users' view of interactive tasks while interacting with the system. This allows describing how a task is performed.
- It is true that in a process model we can add the detail desired, with process hierarchies, to represent a detailed task description. However, we consider that specific temporal operators, iteration, suspend/ resume, applied to task, can be more naturally defined in a task model rather than in a process model, that implies the creation of dummy transitions.

The task model described in [12] [23] benefit from a conceptual contribution; the benefits consist mainly in added some task attributes and relationships. The condition is an attribute not considered in previous version and essential to the occurrence of the task.

- *preCondition*. Before a task can be performed it must fulfill certain conditions.
- *postCondition*. A task can be considered finished if it fulfills the necessary requirements.

The binary relationships that were introduced are:

- *Cooperation*. These relationships specify the cooperation that could be between two or more tasks.
- *Inclusive choice*. These relationships specify two tasks that: both could be executed or just one of them or neither of them.
- *Disabling with information passing*. These relationships occur if one task is completely interrupted by another task; and the information produced in the first task is used as an input for the second task.

Organizational components

Although information systems are an integral part of organization; the key elements of an organization are its people, structure, business processes, politics, and culture. We propose an organization framework representing the places where work is performed and the users that perform it. This part contributes to UI adaptation to different categories of users and security of IS by blocking access to UIs when the user does not have the permission to perform the task.

The attributes of this model are:

- *Organizational unit*. An organizational unit is a formal group of people working together with one or more shared goals or objectives. It could be composed of other organizational units.
- *Task resource*. A task resource is an entity that is directly or indirectly involved in carrying out the work.
- *User stereotype*. This class represents the set of users sharing the same values. Each user stereotype may in turn be decomposed into sub-user stereotypes.
- *Means materials*. It is a type of resource that is physically tangible and is a non human resource.
- *Means immaterials*. It is a type of resource that is physically intangible; it does not have a material form or substance.
- *Log entry*. LogEntry describes specific characteristics that resources may possess. Each resource may have a log Entry associated with him.
- *Job*. Jobs are the total collection of tasks, duties, and responsibilities assigned to one or more positions which require work of the same nature and level.
- *Agenda item*. Agenda items are the tasks that a userStereotype has to perform.
- *Agenda*. The agenda is a list of agendaItem that are assigned to userStereotypes. A userStereotype has one and only one agenda and an agenda belongs to one and only one userStereotype.

Mapping model

UsiXML is composed of various models [23], among them the mapping model serves to gather a set of pre-defined, inter-model relationships that are semantically related.

This mapping model benefits from a conceptual contribution in order to link the tasks with resources (Figure 2). Based in [20] we use the workflow resource patterns to capture the various ways in which tasks are allocated. We extend the mapping model by adding:

- *Is grafted on*. It grafts a task on another one. This relationship is useful when a task (T_j) has been executed, and a task complementary (T_i) is defined to realize the first task. T_i is completely autonomous to T_j .
- *Is defined by*. It refers to a task defined by a userStereotype.
- *Is allocated to*. It corresponds to a task that is assigned to a resource.

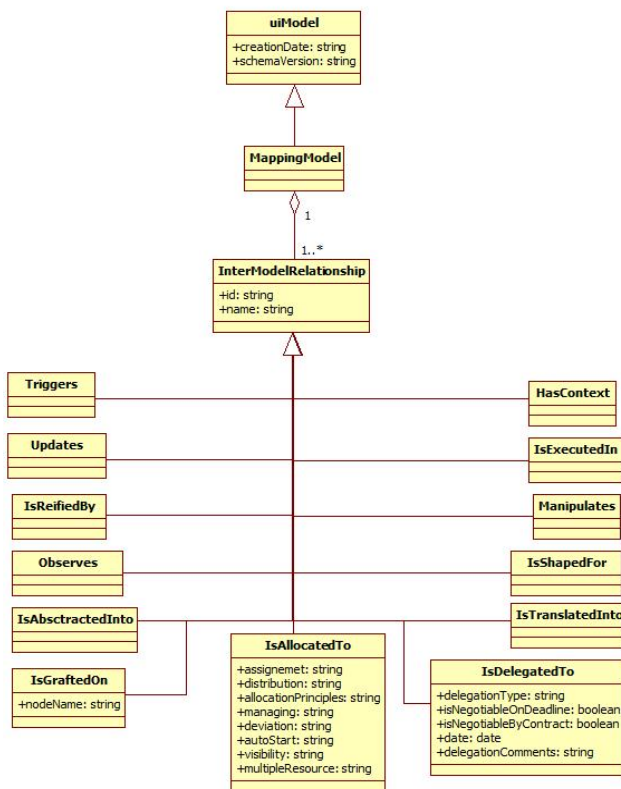


Figure 2. Mapping model.

Workflow users

When a workflow is designed, several *userStereotypes* are involved in the process:

- The *Workflow designer* is in charge of the conceptual process and the drawing of the organization. Direct communication is needed to the *workflow manager* of the organizations to have the global view of the processes to be modeled. The results provided are mock-up of the workflow of the processes to be modeled.
- The *Workflow manager* is a person who is responsible for the handling of a whole workflow. Through the modeling the knowledge required to model the organization is captured with meetings with supervisors, workers, etc.
- The *Process manager* is the person who is responsible for a particular process.
- The *end user* is any *userStereotype* that belongs to the organizational model (internal) or is part of other organization (external). It can be the manager, an economist, the lawyer, any valid user declared in the workflow specification.

The above roles could be assigned to different persons or combined into one single person. Apart from the conception of a workflow system, the final use of the system could impact several levels in the hierarchy of the organization.

DEVELOPING WORKFLOW USER INTERFACES

Developing user interfaces from a workflow specification has several dimensions to be tackled. We need user interfaces to:

- Support user's tasks specified in task models
- Support user's communication with agendas that must be updated accordingly as users received a new item or when they finish a task
- Support tasks allocation, resource patterns, by different means such as: allocation, offering, delegation.

In order to develop UIs, we consider (Figure 3):

- Elicitation of the organization. We assume that there are means to collect information that will serve as input.
- Workflow modeling of the problem which includes the definition of: the organizational units, jobs and user stereotypes, process models, workflow allocation patterns and task models.
- Mapping the workflow specification into a workflow information system.
- Manage the workflow through the workflow manager.

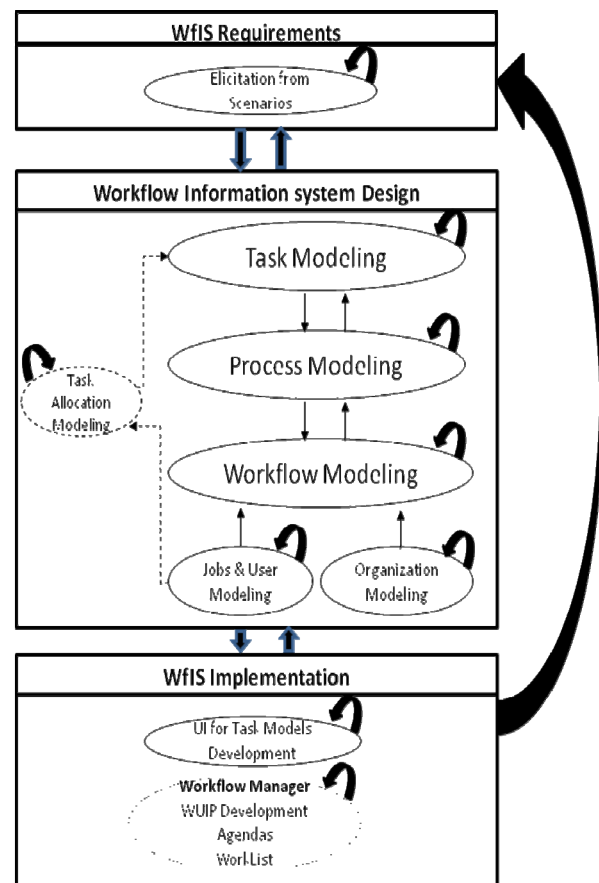


Figure 3. Method overview.

Forward and backward arrows denote the propagation of information from one model to another. For instance, a new task model must make available a task for a process model and vice versa a new task in a process model might be detailed with a task model. Jobs, user stereotypes and organizational modeling just affect the workflow model. Then the workflow model makes them available for process modeling and task modeling. This particular aspect of concepts propagation was significantly useful for the software tools that support our method. We will explain the steps of this method, even they are presented consecutively, it is not mandatory to follow this order.

Step 1: What to do? Processes specification

As we pointed before, the definition of a process indicates which tasks must be performed and in what order. As notation we use Petri Nets, a tool for modeling and analyzing processes.

Step 2: How to do it? Tasks specification

For each process a task model can be specified to describe in detail how the process/task is performed. By exploiting task model descriptions different scenarios could be conducted. Each scenario represents a particular sequence of actions that can successfully be performed to reach a goal. Task models do not impose any particular implementation so that user tasks can be better analyzed without implementation constraints.

Step 3: Where to do it? Organizational units' specification

We introduced an organizational unit concept; it describes the places where work is carrying out. It can be a physical chamber in a builder (e.g. an operating room), but it can also be a complete floor in a hospital. The elements corresponding to the actual organizational structure in a specific organization are specified during the organization design.

This part contributes to UI adaptation to different categories of users and security of IS by blocking access to UIs when the user does not have the permission to perform the task.

Step 4: Who will carry out it? Job and user specification

This step consists in the description of all the users involved in the performance of tasks. Jobs are ways to structure the crew of people inside the organization. It involves the complete collection of knowledge and practices needed by a definite human resource to perform a task.

Step 5: Whom? Applying workflow resource patterns

Actual assignment of tasks to resources is performed according to workflow resource patterns [20]. The way in which tasks are advertised to resources is essential; therefore we need to consider the different states that task goes through, from creation to termination. We propose a life cycle (Figure 4) of the task from the time that it is created to final completion (or failure or cancel).

Workflow resource patterns have been identified that capture the different manners in which resources are presented and used in workflows. The rationale for identifying these patterns was the need to master the many ways according which work can be distributed.

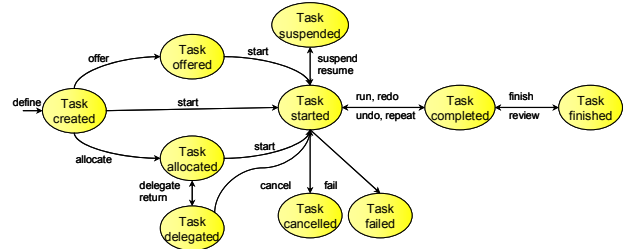


Figure 4. Task life cycle.

In [8] a collection of user interface design patterns for workflow information systems is presented. We give below only a snapshot (Figure 5) for facilitating the understanding and for illustration purpose.

Step 6: Mapping the workflow to user interfaces

Once the workflow has been defined, we part from the task models for developing the workflow UIs and then we follow the different steps until provided the final user interface (Figure 6). We completely rely on UsiXML which is compliant with the Cameleon Reference Framework [1].

Name: Direct allocation

Identifier: R-DA

Synopsis: The ability to specify at design time the identity of the resource that will execute a task

Strengths: To prevent the problem of non-suitable allocation

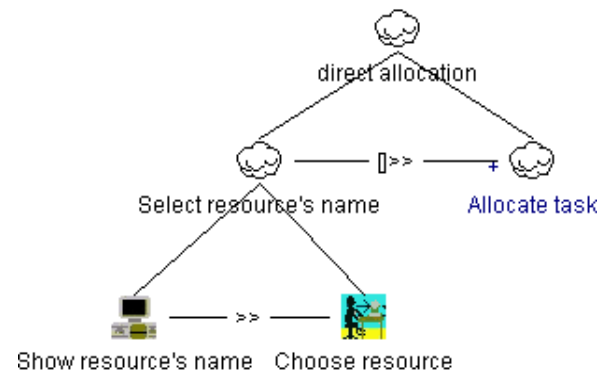
Weakness: No opportunity to change the resource if he is not available to perform the task

Opportunities: To ensure task is routed to specific resource

Problem: This pattern effectively defines a static binding of tasks to a single resource

Solution: Probably the use of deadline and escalation mechanisms when the resource becomes overload and cannot deal with his assigned workload in a reasonable timeframe

Example: "Ask reviewers preferences" task must only be undertaken by "Joshua Brown"



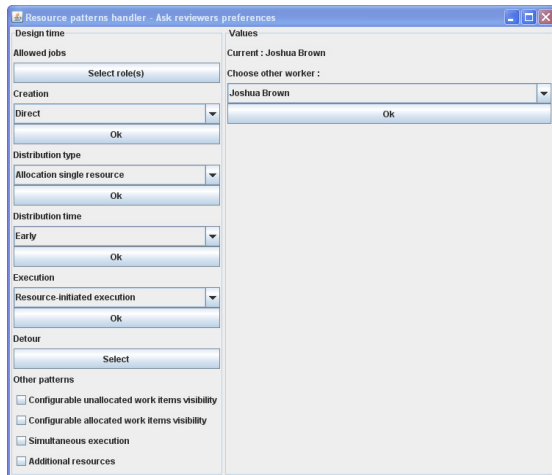


Figure 5. Direct allocation pattern.

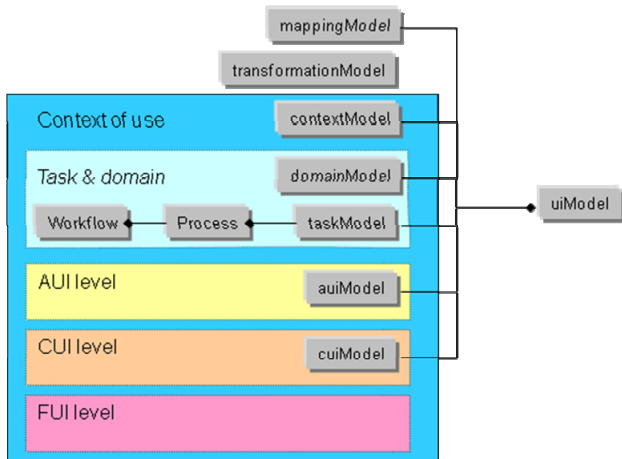


Figure 6. Cameleon Reference Framework and UsiXML.

Case study and workflow editor

In order to support the generation of UIs from a workflow model, a workflow editor has been developed. The case study will be elaborated and described among to the use of the editor and the concepts involved in each phase.

Identification of workflow, process and task. The method starts from a scenario describing the case study to be addressed in a textual format containing the existing or envisioned system with as much details as possible depending on the information available. The way this information is collected and its completeness is not in the scope of our research. Modeling workflow, processes and tasks consists of a fundamental activity that initiates user-centered design in user interface development. It is therefore important to reach the best possible model and that modeling activities remains consistent when the modeler changes. For this purpose, we introduced a set of criteria; see Table 1, in order to identify models during modeling in an unambiguous way that result into a model exhibiting desired properties of quality such as completeness, consistency. In addition, starting and stopping criteria provide designers with guidance on when and how to start and finish each model.

What? Workflow specification. The workflow specification, depicted in the process model, takes place inside of the organizational unit framework. Concretely, the workflow represents the business process and determines the right resource for the right task at the right time (Figure 7).

Table 1. Identification criteria.

	Time	Space (location)	Resource	Type
Workflow	Series of time periods	Different locations; same organization	Same or different groups of resources	-
Process	Series of time periods	Different locations	Within groups, group as a whole, or among groups	Primary (production), secondary (support), or tertiary (managerial)
Task	Same time period	Same location	One or two types of resources	User, interactive, system, abstract, or machine task

Where? Organizational units' specification. This step consists in specifying the location in which the work must be done. Organizational units' attributes are then specified in the editor and graphically the workflow designer identifies the different components of the organization. Organizational units are represented by rectangles (big rectangles in Figure 7), which will contain a set of ordered tasks and the available resources. It is the way to locate those elements inside the organization. The following organizational units are the structural decomposition of the hospital: (i) *reception*: patients coming to this unit will be dispatched through the medical units of the hospital; (ii) *general medicine*: diagnostic and simple medical acts are realized in this unit; (iii) *surgery*: patients will be operated in this unit; (iv) *dermatology*: unit involved in every dermatological resource and the performance of the related medical acts; (v) *payment service*.

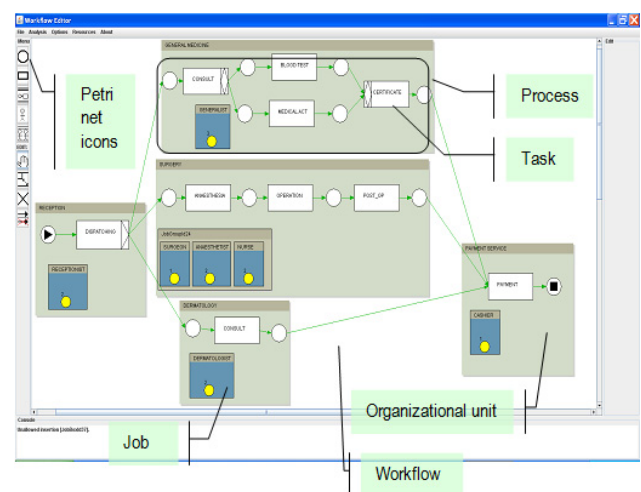


Figure 7. FlowiXML editor.

Who? Specification of jobs and user stereotypes. This step consists in the description of all the actors involved in the workflow. For this purpose we define different levels of users, who are the resources that will be in charge of performing the organization work. Jobs are ways to structure the crew of people inside the organization (Figure 8). It involves the complete collection of knowledge and practices needed by a definite human resource to perform a task. Jobs specified in the definition of the case study are the following: Receptionist, Generalist, Surgeon, Anesthetist, Nurse, Dermatologist and Cashier. Once jobs are defined it is possible to incorporate user stereotypes, people able to carry out tasks of a particular job.

Figure 8. Job handler editor.

The workers editor (Figure 9) is used for this purpose. Workers are defined in terms of attributes (name, experience, hierarchy level) and the list of jobs they can perform. For instance, we define a user stereotype called Robert Wink, having 4 years experience in the third hierarchy level. He is able to carry out tasks as a generalist and surgeon. A data base of workers can be specified using this editor. Also, it is necessary to assign them a place into the organizational scheme. Hierarchically, a worker being in an organizational unit also belongs to every unit included in the main one. The graphical representation used for the workflow editor is based on a first resource container inside the organizational unit. It allows the workflow designer to group resources. Job boxes are put inside of the main resource box. Each job box is instantiated by user stereotypes able to perform the job of the box. This leads to the kind of representation given in Figure 7 (small rectangles). The organizational unit contains a resource box made of three job boxes. Every job box instantiates user stereotypes of a certain job (there are two surgeons, one anesthetist and one in the given example). This lets managers know which

resources are available for execute a task in an organizational unit.

Figure 9. Workers editor.

Whom? Defining workflow resource patterns. It is important to specify who will be in charge of what. For that purpose, we use workflow resource patterns to assign or offer tasks. As, we have already defined jobs and user stereotypes, now we add rules defining the way work will be undertaken. The resource pattern editor (Figure 10) allows the workflow designer to specify resource patterns. At first a list of jobs required to carry out task is specified in the editor. The workflow designer selects one or more jobs allowing a user stereotype to realize the task. For the moment, some workflow resource patterns have been incorporated so that the designer may apply them directly using a predefined UI.

Figure 10. Workflow resource patterns editor.

How? Task models specification. For each process a task model can be specified to describe in detail how the task is performed. By exploiting task model descriptions different scenarios could be conducted. Each scenario represents a particular sequence of actions that can successfully be performed to reach a task goal (Figure 11). We use a plug-in of IdealXML [17].

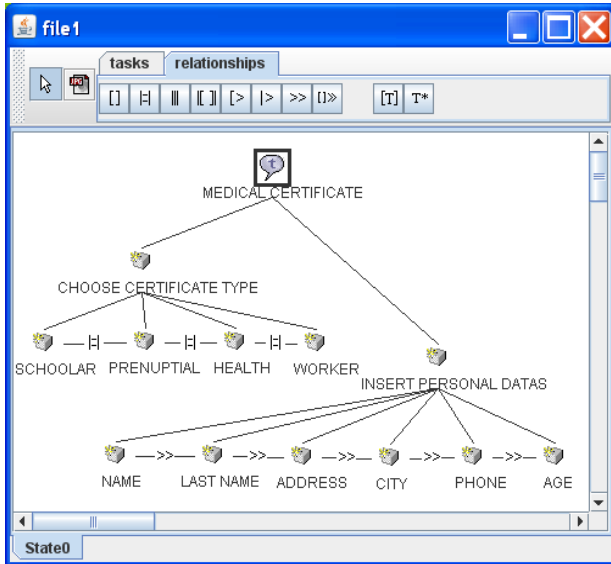


Figure 11. IdealXML editor [17].

Mapping the workflow to UI. Finally we have to deal with the problem of generating the complete UIs set to support all the designed workflow in run-time. This step is achieved by relying on the UsiXML method that progressively moves from a task model to a final user interface. This approach consists of three steps: deriving one or many abstract user interfaces from a task model, deriving one or many concrete user interfaces from each abstract one, and producing the code of the corresponding final UIs.

Figure 12. User interface derived from task model.

To ensure these steps, transformations are encoded as graph transformations performed on the involved models expressed in their graph equivalent. For each step, a graph grammar gathers relevant graph transformations for accomplishing the sub-steps. For instance, applying this method to the task model we obtain its correspondent UI (Figure 12).

CONCLUSION

This paper defined a method for designing UI of workflow information systems where UI are directly derived from a model of the workflow, which is decomposed into processes to end up with tasks. The goal here is not to provide yet another model of workflows, but to identify a meta-model of concepts that are considered fundamental in order to address challenges posed by UI of these workflows. This could be achieved by identifying existing related models and consolidate them.

Based on workflow patterns, it is possible to model an entire workflow with high-level mechanisms and automatically generate the workflow specifications and their corresponding UIs. All models are uniformly expressed in the same XML-based specification language so that mappings between models are preserved at design-time and can be exploited at run-time in needed. Then, the different steps of the approach have been properly defined based on the underlying models and a tool has been developed to support the method enactment. The major benefit of the above method is that all the design knowledge required to progressively move from a workflow specification to its corresponding UIs is expressed in the model and the mapping rules.

The method preserves continuity (all subsequent models are derived from previous ones) and traceability of its enactment (it is possible to trace how a particular workflow is decomposed into processes and tasks, with their corresponding user interfaces). In this way, it is possible to change any level (workflow, process, task, and UI) and to propagate the changes throughout the other levels by navigating through the mappings established at design time. In order to partially support this method, a software tool has been developed in Java 1.5 that supports the graphical editing of the concepts introduced in an integrated way. This method has been so far validated on several real-world case studies. As future work we would like to include workflow analysis to our approach in order to get statistics of the users of the workflow, managing of exceptions (it is important to handle abnormal events that may happen during the execution of a task), and include further implementation experiences, to allow the refinement and refreshment of our approach.

FlowiXML web site

More information, including the tool, a video demo, link to other case studies, can be found at <http://www.usixml.org>.

Acknowledgements

We acknowledge the support of the ITEA2 Call 3 UsiXML (User Interface extensible Markup Language – <http://www.usixml.org>) European project under reference #2008026 and its support by Région Wallonne, the support of the CONACYT program by the Mexican government (<http://www.conacyt.mx>). Thanks to Christophe Lemaigre for the workflow editor and Francisco Montero for IdealXML.

REFERENCES

- [1] Brandl, A., Generating User Interfaces for Ad-hoc Workflows. Poster demo at IUI'01. Santa Fe, USA, 2001.
- [2] Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J. A Unifying Reference Framework for Multi-Target User Interfaces. *Interacting with Computers* 15, 3 (2003), pp. 289–308.
- [3] Casati, F. et al. *WIDE Workflow model and architecture*. Technical Report 96-19, Centre for Telematics and Information Technology (CTIT), University of Twente (1996).
- [4] Dumas, M. and ter Hofstede, A. UML Activity Diagrams as a Workflow Specification Language. In M. Gogolla and C. Kobryn (eds.), *Proc. of 4th International Conference on the Unified Modeling Language UML'2001*, pp. 76-90, Toronto, Canada, 2001.
- [5] Esposito, D. Getting Started with Microsoft Windows Workflow Foundation: A Developer Walkthrough. September 2005. <http://msdn.microsoft.com/winfx/reference/workflow>.
- [6] Guerrero, J., Vanderdonckt, J., and Gonzalez Calleros, J.M. Towards a Multi-User Interaction Meta-Model. Working paper 08/28 ULC, Louvain School of Management. December 2008.
- [7] Guerrero, J., Vanderdonckt, J., and Gonzalez Calleros, J.M. FlowiXML: a step towards designing workflow management systems. *Journal of Web Engineering* 4, 2 (2008), pp. 163-182.
- [8] Guerrero, J., Vanderdonckt, J., González Calleros, J.M., and Winckler, M. Towards a Library of Workflow User Interface Patterns. In *Proc. of 15th Int. Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'2008* (Kingston, July 16-18, 2008). Lecture Notes in Computer Sciences, Vol. 5136, Springer, Berlin, 2008, pp. 96-101.
- [9] Guerrero-García, J., González-Calleros, J.M., Vanderdonckt, J., and Muñoz-Arteaga, J. A Theoretical Survey of User Interface Description Languages: Preliminary Results. In *Proc. of Joint 4th Latin American Conference on Human-Computer Interaction-7th Latin American Web Congress LA-Web/CLIHIC'2009* (Merida, November 9-11, 2009). IEEE Computer Society Press, Los Alamitos, 2009, pp. 36-43.
- [10] Helms, J., Schaefer, R., Luyten, K., Vermeulen, J., Abrams, M., Coyette, A., and Vanderdonckt, J. *Human-Centered Engineering with the User Interface Markup Language*. In Seffah, A., Vanderdonckt, J., Desmarais, M. (eds.), “Human-Centered Software Engineering”, Chapter 7, HCI Series, Springer, 2009, pp. 141-173.
- [11] IBM WebSphere® MQ Workflow. (2006). Retrieved June 2006, from: <http://www-306.ibm.com/software/integration/wmqwf/>
- [12] Kristiansen, R. and Trætterberg, H. Model-Based User Interface Design in the Context of Workflow Models. In *Proc. of Tamodia'2007*. Springer, Berlin (2007) 227–239.
- [13] Limbourg, Q. and Vanderdonckt, J. UsiXML: A User Interface Description Language Supporting Multiple Levels of Independence. In Matera, M., Comai, S. (eds.), “Engineering Advanced Web Applications”, Rinton Press, 2004, pp. 325-338.
- [14] Luyten, K., Abrams, M., Vanderdonckt, J. and Limbourg, Q. *Developing User Interfaces with XML: Advances on User Interface Description Languages*, Proc. of Advanced Visual Interfaces 2004 Workshop UIXML'2004, Galipoli, 2004.
- [15] Luyten, K., Van den Bergh, J., Vandervelpen, Ch., and Coninx, K., Designing distributed user interfaces for ambient intelligent environments using models and simulations. *Computers & Graphics* 30, 5 (2006) pp. 702-713.
- [16] Marshak, R.T. Workflow: Applying Automation to Group Process. In Coleman, D. (ed.): *Groupware-Collaborative Strategies for Corporate LANs and Intranets*. Prentice Hall PTR, 1997, p.p. 143-181.
- [17] Montero, F., López-Jaquero, V. IdealXML: An Interaction Design Tool-A Task-Based Approach to User Interfaces Design. In *Proc. of 6th Int. Conf. on Computer-Aided Design of User Interfaces CADUI'2006* (Bucharest, 6-8 June 2006), Chapter 20, Springer-Verlag, Berlin, 2006, pp. 245-252.
- [18] Object Management Group: Business process modeling notation specification, final adopted specification dtc/06-02-02 (2006).
- [19] Russell, N., ter Hofstede, A.H.M., Edmond, D., & van der Aalst, W.M.P. (2004). *Workflow Data Patterns* (Tech. Rep. FIT-TR-2004-01). Brisbane, Australia: Queensland University of Technology.
- [20] Russell, N., van der Aalst, W.M.P., ter Hofstede, A.H.M., and Edmond, D. Workflow Resource Patterns. In O. Pastor and J. Falcao e Cunha (eds.), *Proc. of 17th Int. Conference on Advanced Information Systems Engineering CAiSE'05*, Vol. 3520 of Lecture Notes in Computer Science (pp. 216-232). Springer-Verlag, Berlin, 2005.
- [21] Stavness, N. Supporting Flexible Workflow Process with a Progression Model. Unpublished master's thesis. University of Saskatchewan, Saskatoon, 2005.
- [22] van der Aalst, W.M.P. The application of Petri Nets to Workflow Management. *Journal of Circuits, Systems, and Computers* 8,1 (1998) 21–66.

- [23] van der Aalst W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B. and Barros, A.P. Workflow Patterns. *Distributed and Parallel Databases 14*, 3 (July 2003), pp. 5-51.
- [24] Vanderdonckt, J. Model-Driven Engineering of User Interfaces: Promises, Successes, and Failures. In *Proc. of 5th Annual Romanian Conf. on Human-Computer Interaction ROCHI'2008* (Iasi, September 18-19, 2008), S. Buraga, I. Juvina (eds.). Matrix ROM, Bucurest, 2008, pp. 1–10.
- [25] van Welie M., van der Veer, G.C., and Eliëns, A An Ontology for Task World Models. In *Proc. of 5th International Eurographics Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'98* (Abingdon, 1998). Springer, Berlin, 1998, pp. 57-70.
- [26] Visual Paradigm International Ltd. Business Process Virtual ARCHITECT, User's guide. Retrieved January 2, 2007. From www.visual-paradigm.com/product/bpva/bpvadocuments.jsp
- [27] W3C State Chart XML (SCXML), State Machine Notation for Control Abstraction 1.0. W3C Working Draft (2005). <http://www.w3.org>
- [28] Workflow Management Coalition (1999). Workflow Management Coalition Terminology & Glossary. Document Number WFMC-TC-1011. Document Status – Issue 3.0

Prototyping Digital, Physical, and Mixed User Interfaces by Sketching

Adrien Coyette, Jean Vanderdonckt

Université catholique de Louvain, Louvain School of Management, Place des Doyens, 1
B-1348 Louvain-la-Neuve (Belgium)
{adrien.coyette, jean.vanderdonckt}@uclouvain.be

ABSTRACT

Sketching digital and physical user interfaces present many benefits such as naturalness, intuitiveness, ability to elicit user requirements, and ability to discover usability problems. These advantages are confirmed in MIXEDSKETCH, a software for prototyping any type of user interface by sketching: a digital interface such as a graphical user interface, a physical interface such as a tangible user interface, and, more uniquely, mixed user interfaces that combine elements from both digital and physical worlds, simultaneously or at different design stages. As the development process proceeds from early design to detailed development, MIXEDSKETCH ensures a smooth transition from a low-fidelity representation to a high-level representation of the UI being sketched through mid-fidelity. In the last stage, a precise presentation and a dialog can be sketched that automatically generate a description of the future interface for one or multiple toolkits. In addition, MIXEDSKETCH enables the designer to transform a digital, a physical, or a mixed user interface in a counterpart in another world, e.g. moving from digital to physical to mixed.

Author Keywords

Level of fidelity, shape recognition, sketching, user interface design, user interface prototyping.

General Terms

Design, Human Factors, Languages.

ACM Classification Keywords

D.2.2 [Software Engineering]: Design Tools and Techniques – *User interfaces*. H.5.2 [Information Interfaces and Presentation]: User Interfaces – *Graphical user interfaces*. I.3.6 [Computer Graphics]: Methodology and Techniques – *Interaction techniques*.

INTRODUCTION

Prototyping a digital user interface (UI), such as a Graphical User Interface (GUI), is typically conducted in a GUI builder by dragging interaction objects or widgets (such as radio buttons, push buttons, and sliders) from a palette and dropping them onto a working surface area. In order to fine tune the *presentation* (i.e., the UI static part), the designer specifies physical properties of these widgets such as location, dimensions, colors, and layout. When it comes to defining the *dialog* (i.e., the UI dynamic part), the drawing visual paradigm stops: the designer needs to open every widget of concern and program its behavior in the pro-

gramming, markup or scripting language supported by the Integrated Development Environment (IDE).

Prototyping a physical User Interface (UI), such as a Tangible User Interface (TUI) [4], is even more challenging: the objects are by definition physical, thus difficult to prototype unless a digital counterpart exists. Both their presentation and their dialog are complex to specify, existing IDEs offer little or no help, everything is often coded by hand, leaving little or no space for prototyping outside coding.

We hereby define a *Mixed User Interface* (MUI) as a UI that combines digital and physical objects to be designed and developed in a coordinated way, by ensuring consistency across both worlds and integrating them [10]. MUI prototyping is even more complex since there is a need to bridge the gap between softwired and hardwired worlds [25] and make a correspondence between them.

Sketching is universally recognized for its natural [11], unconstrained [27], and informal [9] virtues in multiple areas of human activity, such as layout design [12], multimedia design [1], and UI design [3,5,11,16,20]. As long as the sketching is not submitted to any recognition engine, the end user does not perceive any shortcoming apart from little or no reusability of the sketches for future steps in the design process [18]. When the sketches need to be recognized, e.g. for beautification [3] or interpretation [22], the end user may feel again constrained as she knows that every gesture should be performed correctly in order to be properly recognized, thus reducing the virtue of naturalness.

UI design by sketching has already demonstrated several advantages: UI sketching is preferred over traditional interface builders, especially by end users [11,16], it can be performed at different levels of fidelity without losing advantages [17,26], the amount of usability problems discovered through a sketched design is not inferior to those corresponding to a genuine UI [26], the expressive power of a sketched UI remains the same [27], a sketched UI provides quantitative and qualitative results that are comparable to traditional UI prototypes except that the cost is reduced [18], UI sketching encourages exploratory design and fosters communication between stakeholders more than any other prototypes [22], flexibility is superior to UI builders [27], authoring tools [1], and paper prototypes [27]. There-

fore, sketching is an interaction technique that could be effectively and efficiently used for prototyping UIs, but has never been considered extensively for designing GUIs, TUIs, and mixed UIs at the same time.

There are a number of problems that traditional sketching methods pose which are challenging for novice users and inefficient for expert users. The first problem is related to the meaningfulness of representations: what is the best object representation? Should multiple representations of the same object be offered? How should it be sketched? Should it be sketched in one stroke or several strokes? If a representation is not meaningful enough for an end user, the representation will be forgotten or badly drawn. The second problem is that the result of the chosen representation is often far from what is expected by the novice user and difficult to reproduce [16]. The third problem with traditional recognition engines is that the representations should be different enough [12] and sketched precisely enough to be efficiently recognized [3,9,12].

To address these problems, we developed MIXEDSKETCH, a multi-platform sketching tool that provides original functionalities with respect to state-of-the-art software (as described in the following section) and that supports designing GUIs, TUIs, and MUIs. These three UI types will be addressed respectively in the next sections, as well as the link between. A summary of the main contributions of this paper and a description of the planned follow-up work conclude the paper.

RELATED WORK

During the UI development life cycle, the design step is often characterized as a process that is intrinsically open (new designs may appear at any time that require further exploration), iterative (several cycles are performed to reach a solution), and incomplete (not all information is available at design time) [11]. The area of UI design by sketching has been extensively researched to identify appropriate techniques such as paper sketching, prototypes, mock-ups, diagrams [1,3,5,9,11,13,16,20]. Several software for UI design by sketching emerged from this research: DENIM [13], DEMAIS [1], FreeForms [16], InkKit [16], JavaSketchIt [5], Satin [9], Silk [11] to name the most representative ones.

Since the needs of rapid UI prototyping vary depending on the project and allocated resources, it makes sense to rely on the level of fidelity. The level of fidelity expresses the similarity between the final UI and the prototyped UI. The UI prototype fidelity is said to be *high* if the prototype representation is the closest possible to the final UI, or almost in the same representation (based on [16]). This means that the prototype should be of high-fidelity in terms of presentation (what layout, what are the UI elements used), of global navigation and dialog (how to navigate between information spaces), of local navigation (how to navigate within an information space). More precisely, McCurdy *et al.* [14] identified five independent dimensions along which the level of fidelity could be more rigorously defined: the

level of visual refinement, the breadth of functionality, the depth of functionality, the richness of interactivity, and the richness of the data model. In the remainder of this paper, the four first dimensions will be considered, the last one requiring a connection to a data model containing data.

Similarly to the above definition, the level of fidelity is said to be *low* if the prototype representation only partially evokes the final UI without representing it in full details. Between *high-fidelity* (Hi-Fi) and *low-fidelity* (Lo-Fi) [17], we can introduce *medium-fidelity* (Me-Fi). We usually observe that a UI prototype only involves one representation type, i.e. one fidelity level at a time. But due to the variety of stakeholders' inputs, several levels of fidelities could be combined together, thus leading to the concept of *mixed-fidelity*, where several different fidelities are mixed in the same UI design [13]. Beyond mixed-fidelity, we introduce *multi-fidelity* that is reached when a prototype simultaneously involves elements belonging to different levels of fidelity, but only one level of fidelity is acted upon at a time, thus assuming that a transition is always possible for an element from one level of fidelity to another.

Another system to support UI design on tangible surfaces is shown in The Designer's Outpost [10]. The system tracks multiple post-it notes on a surface and is used to support the designing of web sites. Several post-its are added, labeled and recognized by computer-vision. Electronic sketching is hereby used to add hierarchy to the post-its on the surface. Digital pen input is used to describe functional dependencies between the different post-its.

Similarly to The Designer's Outpost, the Pin&Play environment [3,23,25], the VoodooIO toolkit [25,26] support the arrangement of multiple physical TUI elements on a rigid or flexible surface area and connect them together so as to form a working interface, e.g. for music [28], for sound design [27] based on a TUI. However, instead of using passive post-its, the aforementioned systems use active elements like buttons, sliders, or dials.

Successful examples of prototyping environments for various UI types are demonstrated their feasibility and applicability: DEMAIS for multimedia applications [1], iStuff for ubiquitous computing [2], JavaSketchIt for Java applications [4], d.Tools for TUIs [8], The Designer's Outpost for web sites [10], PALETTE for presentations [14], PaperButtons for physical UIs [15], PaperPoint [19], and VoodooFlash for TUIs [21].

In the next sections, we will demonstrate that MIXEDSKETCH departs from the state-of-the-art, in particular the aforementioned works, in that it:

- Supports simultaneously the rapid prototyping of digital UIs (such as GUIs), physical UIs (such as TUIs), and mixed UIs (MUIs) that combine elements belonging to both worlds, digital and physical.
- Enables designers to sketch these UIs in multiple levels of fidelity, to transform each element from one level of

- fidelity to another, and to apply fusion of elements.
- Enables designers to smoothly switch from one world to another in order to prototype a behaviorally-equivalent UI in another world (for instance, a GUI as a MUI).
 - Exports a representation of such UIs that can be imported in a IDE, an appropriate toolkit, or rendering engine.

DIGITAL INTERFACE PROTOTYPING

This section describes how MIXEDSKETCH supports the prototyping life cycle of a GUI belonging to the digital world. The original features of the tool are progressively motivated, described, and discussed. This sketching tool today consists of about 122,000 lines of Java 1.5 code and can be freely downloaded from www.anonymized.org, both the executable software and its full source code. This tool enables designers to sketch a UI as easily as on paper, while keeping advantages of computer-based design as identified and confirmed in [1,16,26,27]. At any time, the designer may ask the tool to recognize the UI being sketched and generate a working UI from the sketches. At any time, it provides the facilities detailed in the following sub-sections.

World modeling. MIXEDSKETCH models the design space as a simplified object-oriented world, i.e. a set of *objects* linked by typed *relationships*. Each object can be described by its location (X,Y), its dimensions (length and height), and additional properties that can be specified through property sheets (e.g., design annotations, color). Each relationship is characterized by its type (e.g., spatial constraint, decomposition, navigation link), by its edges (i.e., how many source and target edges, what types of objects are accepted for each edge), and by additional properties (e.g., one-way or two-ways transitions). Therefore, both digital and physical design worlds will be represented in the same way with connections between.

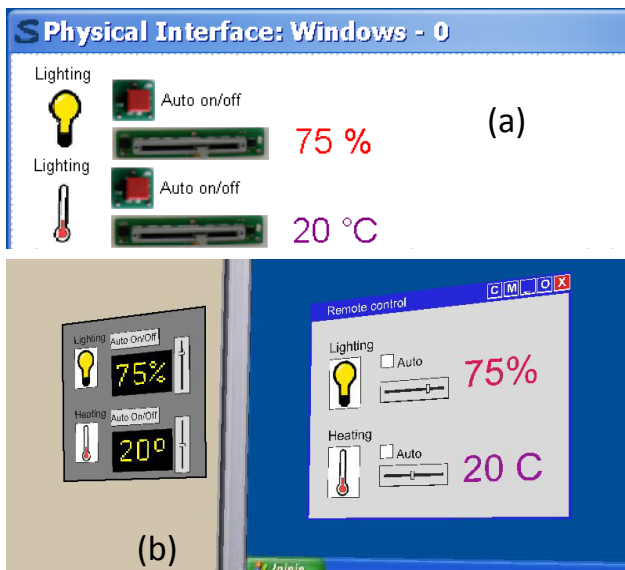


Figure 1. Three UIs for lighting and heating control: a TUI (a), a physical UI (b, left), a software GUI (b, right).

Fig. 1 depicts three different UIs for a simple home control of lighting and heating: a TUI sketched in terms of the Voodoo toolkit [4] in MIXEDSKETCH (a), a physical UI made of physical devices (b, left), and a GUI made of corresponding widgets in Windows environment (b, right) rendered in a 3D scene. Each time a UI is designed, its underlying model is automatically generated for the end user in an appropriate User Interface Description Language (UIDL).

Object recognition. An object recognition engine recognizes and interprets 32 different types of widgets (ranging from check boxes and spin button to search buttons, progress bar, calendar, and video input), 8 basic predefined shapes (i.e., triangle, rectangle, cross, line, wavy line, arrow, ellipse, and circle), and 6 basic commands (i.e., undo, redo, copy, paste, cut, new window). This amount of recognized objects is superior to what can be found in other software equipped with a recognition engine in the same domain [3,11]. Each object is rigorously defined in terms of constituent shapes (any of the 8 aforementioned basic shapes) and constraints between them.

Each constraint should belong to the set of the 31 constraints supported today: areParallel, cross, hasInside, hasInsideInLowerRightCorner, hasInsideInTheCenter, hasInsideInTop, hasInsideInUpperRightCorner, hasInsideOnTheLeft, hasInsideOnTheRight, hasPositiveSlope, intersect, isCrossedBy, isHorizontal, isInside, isInsideInBottom, isInsideInLowerRightCorner, isInsideInTheCenter, isInsideInTop, isInsideInUpperRightCorner, isInsideOnTheLeft, isInsideOnTheRight, isOnTheLeftOf, isOnTheRightOf, isOnUpperLeftCorner, isSmall, isSquare, isThin, isUnder, isVertical. Any object representation is expressed in a XML format stored in a graphical grammar [3] that is parsed and interpreted at run-time [11]. In this way, any custom object could be easily added by adding a new representation in the grammar. Each UI element can be sketched and recognized or not depending on its shape and the wish for the user to see it recognized or not. The object recognition is only on-demand. Those shapes which are not recognized are simply added and maintained throughout the process. Fig. 2 shows a design session where some UI objects have been sketched in Lo-Fi mode. In this mode, objects which have been correctly recognized are beautified and the name is added.

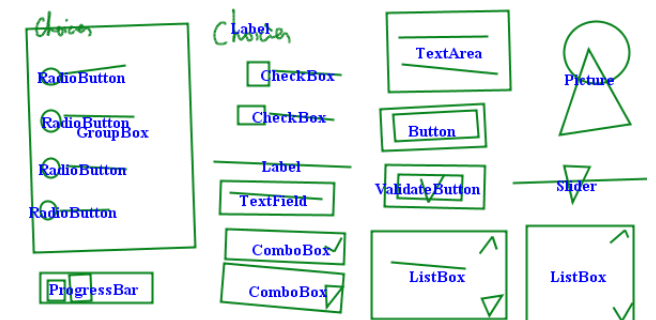


Figure 2. A typical GUI design session with sketched objects.

Multiple object representations. Existing software incorporating an object recognition engine typically support only one single representation per object, most frequently through a mono-directional single-stroke gesture [3]. Our tool accommodates several representations for a single object, without affecting significantly the system response time. Fig. 3 reproduces an excerpt of the radio button representation. In addition, thanks to this logical definition, each representation could be sketched in a multi-stroke manner that is independent of the direction. In this way, left-handed or right-handed persons are equally supported.

```

- <widget type="RadioButton">
- <representation id="0">
  <constraint id="0" shapel="Line_1"
  shape2="Circle_0" condition="isOnTheRightOf"
  />
  <constraint id="1" shapel="Line_1" shape2="-"
  condition="isHorizontal" />
  <shape id="Circle_0" type="Circle" />
  <shape id="Line_1" type="Line" />
</representation>
...
</widget>

```

Figure 3. A representation for the radio button.

Multi-fidelity representation. Thanks to the object recognition process, the designer can input any UI object in any level of fidelity and see the result in any other level as the interpretation is immediate. In the same way, any custom object could be drawn in Lo-Fi and a predefined widget could be added in Me-Fi or Hi-Fi. Therefore, four fidelity levels are supported as recommended by [14]: none (only the drawing is displayed), Lo-Fi (the drawing is displayed with recognized portions), Me-Fi (the drawing is beautified where portions are recognized, including for basic shapes), and Hi-Fi (a genuine UI is produced with widgets for those recognized portions). Fig. 4 exemplifies multi-fidelity representations for a subset of the 32 widgets supported. To our knowledge, no existing software today supports so many widgets in different levels of fidelity as we have here.

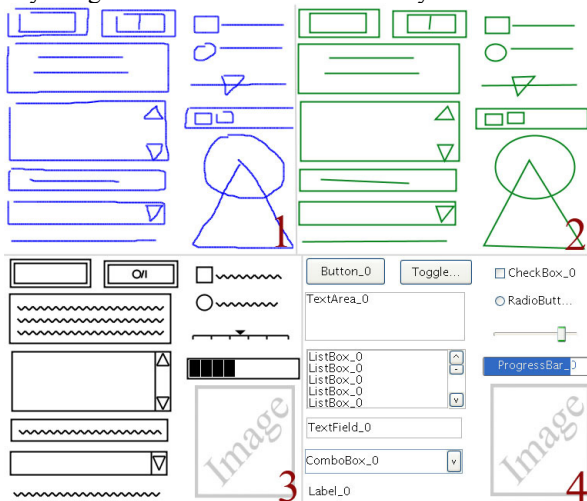


Figure 4. A set of widgets with the representations corresponding to the four levels of fidelities (none, low, medium, high).

Fidelity transition. A slider allows the designer to easily switch between the four levels of fidelity (Fig. 5). Fig. 6 shows a GUI prototype after the designer moved from No-Fi (a) to Lo-Fi (b), to Me-Fi (c), and Hi-Fi (d). Note that the representation of Fig. 6c is platform agnostic: it does not produce any representation that would suggest any particular window manager or UI builder. In Fig. 6d, the representation is made up of genuine widgets belonging to the widget set of the currently being used platform (here, Java). In order to identify the parts recognized by the engine, labels of recognized widgets can be turned on (as in Fig. 6b and d) or off (as in Fig. 6a and c). Different widget sets and look & feel could be used alternatively that mimic a Hi-Fi representation in other window managers and operating systems. If a UI element has not been recognized, it is simply kept as it is. For instance, if a histogram would have been sketched, it would not be altered so as to respect the naturalness of the design process as recommended in [13].

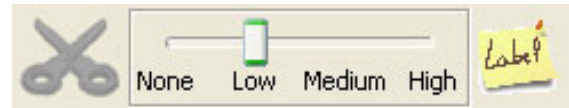
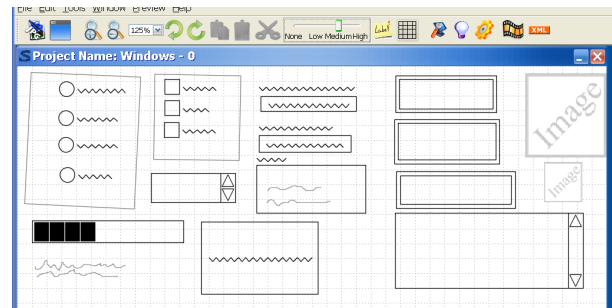
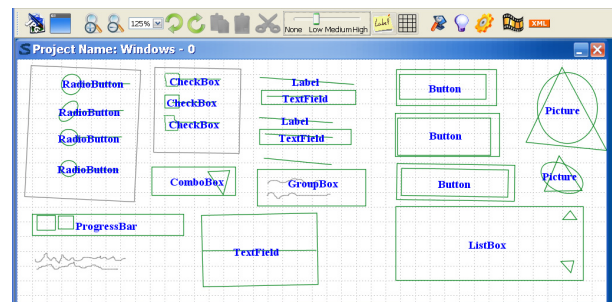
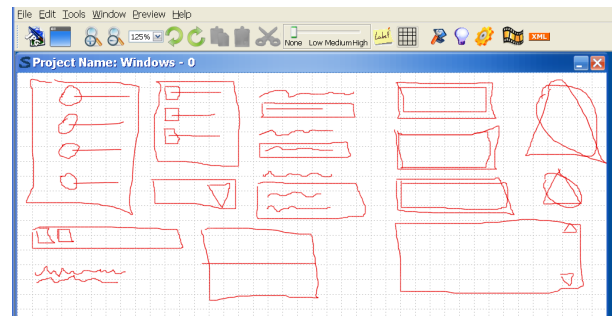


Figure 5. Slide to switch between levels of fidelity.



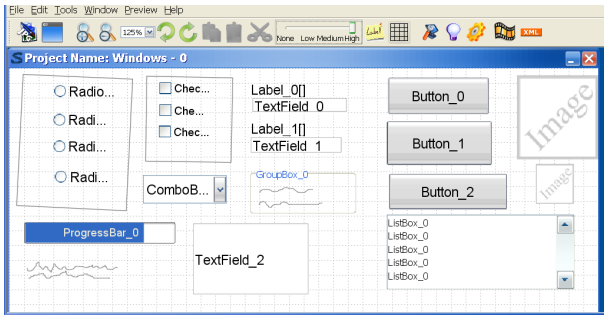


Figure 6. Transition between the four levels of fidelity.

Gesture recognition. Sketching tool users sometimes complained that they are forced to learn a graphical representation [12] for every widget, shape or command. In order to support this user flexibility, each such object could of course give rise to a new representation in the graphical grammar. Some user studies revealed the need for the user to interactively define personalized representations [12]. For this purpose, a gesture recognition system has been implemented based on hand gesture decomposition in order to customize the representation of all widgets, shapes, and commands according to each user's preferences (Fig. 7). One or several occurrences of a new gesture could be graphical defined that will then serve as a redundant input technique for every widget, shape, or command.

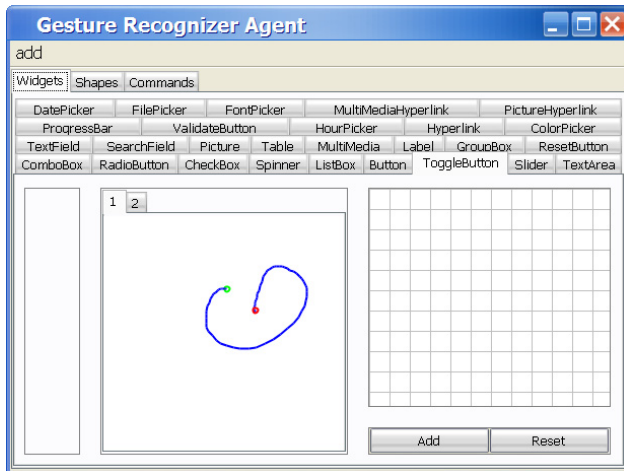


Figure 7. A graphical editor for a new object representation and a gesture recognition system where a new gesture replaces a predefined object (here, a gesture is drawn, added, and activated to represent a toggle button in a personalized way).

Multiple output formats. At any time, the tool produces UI specifications in terms of a User Interface Description Language (UIDL) instead of UI code, which is the prevalent approach of most tools [3,13,16,19]. As opposed to many tools where little or no portions of the sketch could be re-used, our tool always maintains up-to-date UI specifications, including the description of custom objects. It is also possible to define the navigation between these objects in the same way to address the second and third dimensions of

McCurdy [14] (Fig. 8). Whereas Fig. 5 graphically depicts the four levels of fidelity for expressing the presentation of a GUI, Fig. 7 shows how to define the corresponding dialog (or behavior) in terms of Event-Condition Action (ECA) systems. Predefined transitions exist, such as in DEMAIS [1], to express behaviors like: if the user clicks on this button, then this control is activated (or deactivated), if the value v is selected among the values of the widget w , then that information is displayed. Other behaviors include coupled widgets, like coupling a slider and the value representing a percentage (as in Fig. 1). These predefined transitions are specified in graphical direct manipulation. Other transitions are expressed in the ECA rule language inspired from the state chart mechanism in [8].

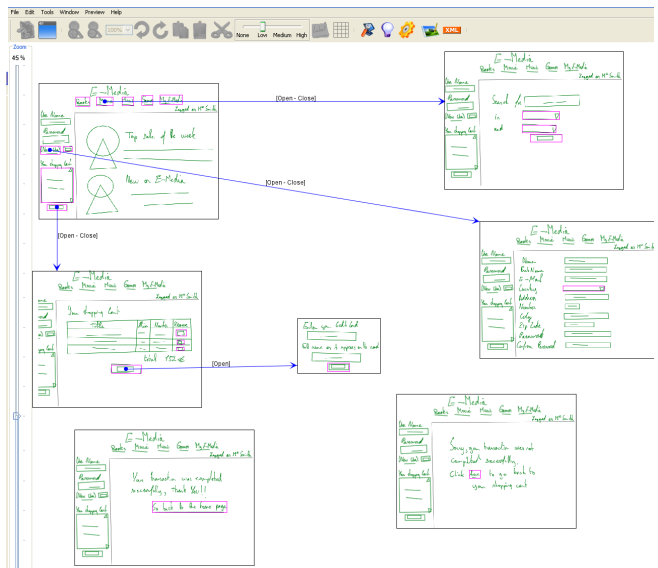


Figure 8. Definition of the dialog of UI objects.

Multi-platform UIs. By specifying project properties, the sketching tool enables designers to sketch UIs for a particular computing platform at a time (e.g., a desktop) or for several platforms in a coordinated way (e.g., a PDA, a laptop, and a desktop computer). It exports UI specifications in UIML (www.uiml.org), UsiXML (www.usixml.org – which is able to automatically generate code for XHTML, Java, and XUL). As opposed to some tools which are dedicated to a particular environment (e.g., Visual Basic in FreeForms [16], Java in JavaSketchIt2 [5]), MIXEDSKETCH is shipped with predefined profiles covering a wide range of different computing platforms, including but not limited to: mobile phone, smartphone, PocketPC, Tablet PC, laptop, and desktop. Each profile not only expresses constraints imposed by a particular platform (e.g., the screen resolution, a restricted widget set), but could also have a particular gesture data base for sketching those UI elements which are peculiar to this platform (e.g., a gesture associated to a histogram). Such specific UI elements are added by the gesture recognizer agent (Fig. 7). MIXEDSKETCH also renders a sketched UI for some particular Look & Feel of a computing platform (Fig. 9).

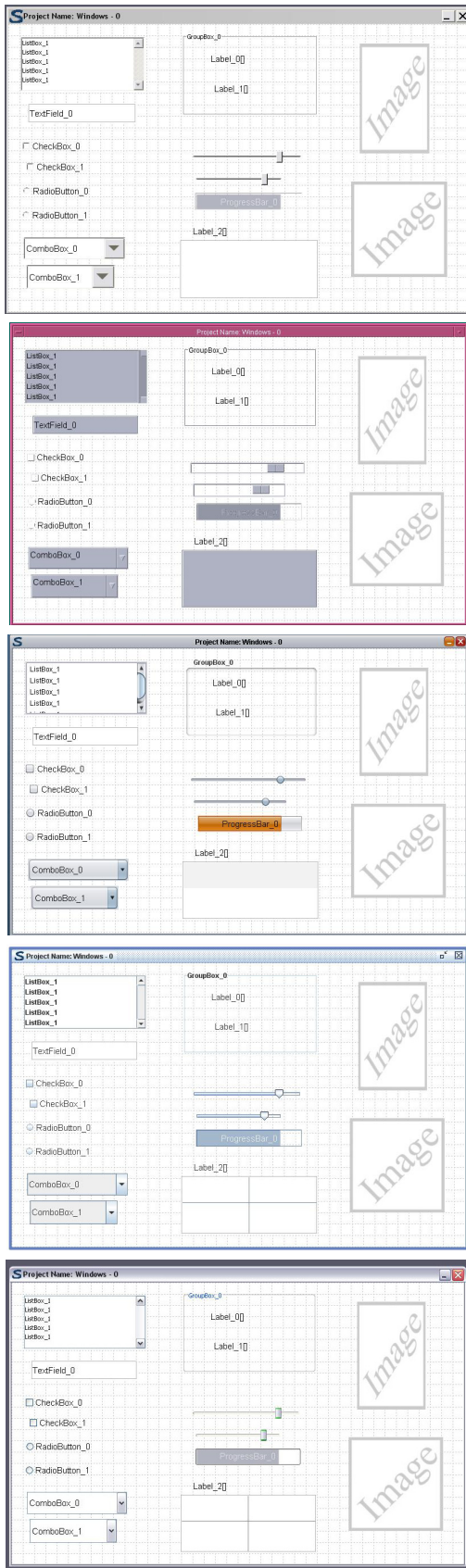


Figure 9. Rendering for different platforms.

PHYSICAL INTERFACE PROTOTYPING

This section describes how MIXEDSKETCH supports the prototyping life cycle of a TUI belonging to the physical world. In addition to the features presented in the previous section, other features are introduced to support prototyping TUIs belonging to the physical world.

Multiple input formats. At any time, a particular TUI object set can be specified in order to rely on the definitions available in the object set. If for instance, a TUI object set like Phidgets [7] offers predefined TUI objects, a representation can be defined for each such object that can be recognized either in the object recognition engine or in the gesture recognition engine. As long as the TUI objects of the set are defined according to the rules defined in the world modeling, they can be part of the sketching process whatever their purpose is. Examples are Pin & Play [6,23] and Voodoo [24]. Pin&Play [23] is a toolkit that allows developing physical interfaces by integrating software widgets and physical devices such as slider, toggle button, and potentiometer. Since the toggle button is not a standard element, it has been defined through a new custom gesture (Fig. 7), which could then be associated with the description of the genuine physical toggle button (such as a switch) belonging to the TUI object set. Fig. 10 respectively reproduces such a physical UI in Lo-Fi (a), Me-Fi (b), and Hi-Fi (c) with a smooth transition between these modes as previously described.

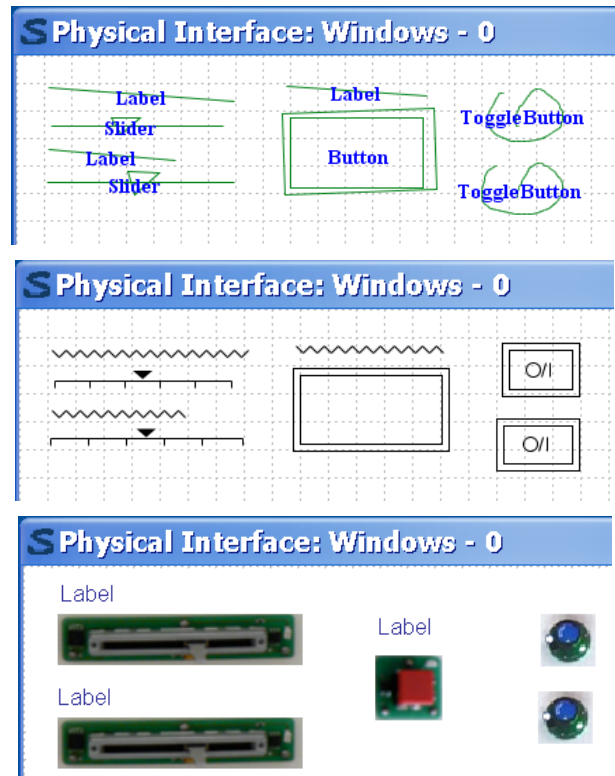


Figure 10. A simple TUI sketched in Lo-Fi mode (a), then transformed into Me-Fi mode (b), and represented in terms of TUI elements in Hi-Fi mode (c).

Multiple output formats. Once a particular TUI has been specified, its internal model is automatically generated and augmented depending on information provided during the design stage. A mapping system exists that links each internal model object to the corresponding representation of the target output, like a description in Voodoo or in Pin&Play. Other TUI object sets could be equally supported provided they can define their design space in terms of objects and relationships as explained before. For the moment, the model is expressed in terms of a UIDL such as UsiXML (User Interface eXtensible Markup Language – <http://www.usixml.org>).

MIXED USER INTERFACE PROTOTYPING

Multiple rendering. In the two previous sections, we have seen that each UI corresponding to the digital or physical worlds can be sketched in exactly the same way: start at a particular level of fidelity (there is no need to start from Lo-Fi), move to any other level of fidelity as appropriate, and refine in the last Hi-Fi level. When the prototype has been validated by the end user, its specifications can be exported to a rendering engine. Two kinds of rendering engines exist:

1. An *interpreter* reads the generated specifications and graphically renders the corresponding UI in the environment. Fig. 1b shows a rendering engine in virtual reality so as to prototype UIs for ambient intelligence. If the TUI of Fig. 1a has been drawn, then a physical wall UI could be rendered (Fig. 1b – left) or a software counterpart (Fig. 1b- right). This is achieved in a very small amount of time by reinterpreting the UI specifications in the desired engine. Fig. 11 shows a screen shot of the VRML rendering engine where the same UI is projected on a wall, thus allowing the designer to explore alternative designs.

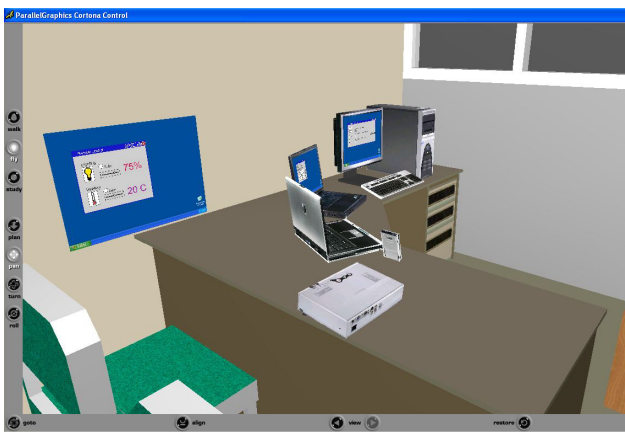


Figure 11. A projected GUI corresponding to the GUI (Fig. 1).

2. A *code compiler* reads the generated specifications and generate the code corresponding to the UI. Fig. 12 has transformed the TUI depicted in Fig. 10c into a GUI by relying on the mapping system. In this case, there is a mapping between GUI widgets and TUI objects. If such a mapping cannot be established, the initial object remains unchanged or is erased depending on preferences.

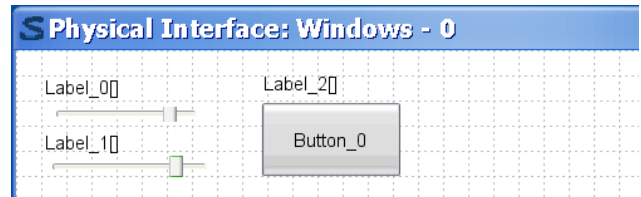


Figure 12. A simple GUI corresponding to the TUI of Fig. 10.

Therefore, not only it is possible to sketch a digital or physical UI at different levels of fidelity, but it is always possible to switch from one world to another and immediately see the corresponding UI in the other world. When no specific world is designated, a same UI being prototyped may involve GUI controls as well as TUI objects since their internal representation in the underlying model remains the same. In this way, a mixed UI could be similarly prototyped.

Transition from any world to another one. When a GUI is sketched in the digital world, MIXEDSKETCH is able to automatically generate a corresponding TUI (here a physical UI) in the physical world by searching the *closest behaviorally-equivalent element* in the target world. For instance, a slider widget (belonging to the digital world) can be mapped onto a physical slider (belonging to the physical world). The technique works also in the other direction: a TUI can be transformed into a GUI whose elements are found to be the closest behaviorally-equivalent to those found in the TUI.

In order to determine the closest behaviorally-equivalent element from one world to another, a UML class diagram has been realized that abstracts the most frequent elements into *abstract user interface objects* (Fig. 13). These objects are the core elements of a *abstract user interface model*, a model that describes canonically a user interface in terms of abstract interactors, containers and relationships in a way that is independent from the concrete interactors available on the target platforms. In practical terms this means that an abstract UI model is independent of any world, digital or physical. An *Abstract User Interface Object* is the root of the hierarchy of User Interface object. *AuiObjects* are the elements populating the AUI Model, they may be of two types:

AuiContainer is a container for grouping elements in order to define group of tasks that have to be presented together, in the same window or panel, for example. The container could contains *AuiInteractor* elements or another *AuiContainer* element. The relationship between *AuiContainer* and *AuiInteractor* is declared with a *AuiRelationship* object.

AuiInteractor is any individual element that populates an abstract container. It could be either a *DataInteractor* (i.e., an aggregation of the different types of elements that interacts with the user to present or obtain data which could be (input or output) and (PredefinedSet or UndefinedSet) or a *TriggerInteractor* (i.e., an aggregation of the different types of elements that triggers actions).

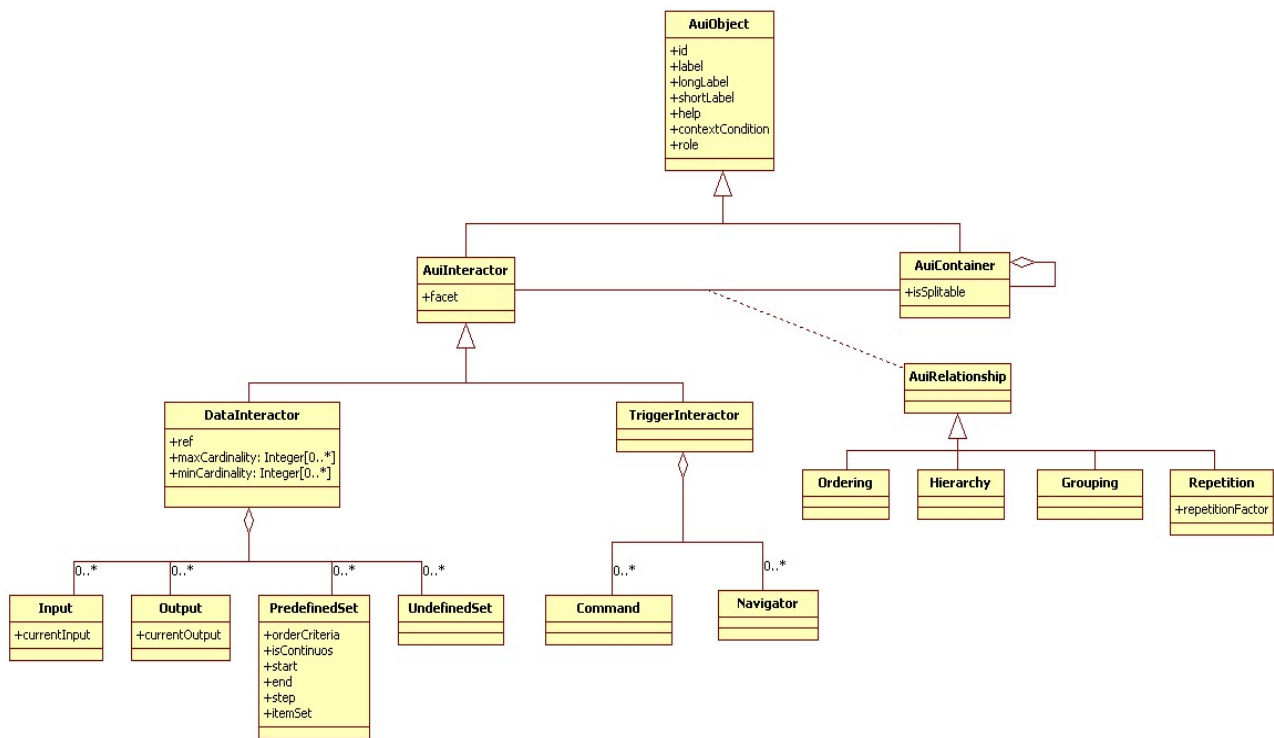


Figure 13. The meta-model used for the abstract user interface characterization.

An *AuiInteractor* cannot be simultaneously a *DataInteractor* and a *TriggerInteractor*. A *DataInteractor* could of course be attached to a particular behaviour depending on the data manipulated (i.e., entered, deleted, or modified).

Input is an element used for obtaining values from the user, *Output* is an element used for presenting values to the user, *PredefinedSet* describes the set of values valid for the element, *UndefinedSet* describes a set of open values (without predefined values). In this way, each *AuiInteractor* may hold one or several facets that are represented by the sub-classes. The UML class diagram of Fig. 13 does not represent the constraint that each *DataInteractor* should have at least one facet represented by a sub-class.

Of course, an *AuiInteractor* may have until four facets: *Input*, *Output*, *PredefinedSet*, and *UndefinedSet*. Indeed, an interactor can serve for input only (no feedback), output only (only display), input/output (input that gives some system feedback) and still have a mixed domain of values: one part that is known at design-time (*PredefinedSet*) and another part that is unknown at run-time (*UndefinedSet*).

TriggerInteractor is an aggregation of the different types of elements that triggers actions, while *CommandInteractor* is an element used to trigger a command. *NavigatorInteractor* is a trigger used for indicating a navigation to manage. Other classes and relationships are detailed at <http://www.-anonymous.org>. Here, we only detail the classes useful for searching the closest element.

Basically, any element, whether it is a GUI element belonging to the digital world or a TUI element belonging to the physical world, is linked to an abstraction expressed in the

terms outlined in Fig. 13. For instance, a slider widget is characterized as a *DataInteractor* that is applicable for both *Input* and *Output*, based on a *PredefinedSet*. A physical slider is characterized exactly in the same way.

By definition, the closest behaviorally-equivalent element is any element that maximizes the matching in terms of the *AuiModel*. For this purpose, the system checks all the facets of existing elements in order to find any other element matching the same facets with any sub-set or super-set of values for them. The facets are examined first one by one (i.e., *Input*, *Output*, *PredefinedSet*, and *UndefinedSet*), then in combination of 2 (i.e., *Input* and *Output*, *Input* and *PredefinedSet*, *Input* and *UndefinedSet*), 3, or 4, if possible.

In the case of our slider, the answer is obvious since a perfect match exists. If this is not the case, the system searches the graph for the element that matches the best the input element by checking all facets and associated constraints. Therefore, one initial UI element of a source world can be mapped onto zero, one, or many UI element of the target world. When zero matching exists, the initial UI element is only replicated or erased depending on options set by the designer. When only one match exists, thus meaning a bijection, the mapping is straightforward: the corresponding element is selected and added in the UI model corresponding to the new UI of the new world. When many matches exist, they are ranked by level of similarity of behavior and the designer is then allowed to choose any of them. This metric consists of the sum of similarities for all properties hold in each facet of each element. The designer can specify a threshold beyond which this similarity is considered

reached since, in general, no perfect match exist for all values for all properties. Fig. 14 shows some mappings between GUI elements and TUI elements maintained in ontology of MIXEDSKETCH.

This ontology has been created by exporting the UML class diagram of Fig. 13 in XML, along with the mappings to respective elements. Although this file can be edited manually, it is the responsibility of the designer to maintain consistency between the mappings in order to avoid any mistake. MIXEDSKETCH then parses this ontology at run-time in order to browse the relationships between *AuiInteractors*. Actually, each UI element used in one world is abstracted into a corresponding *AuiInteractor*. Relationships between *AuiInteractors* are then exploited to find out the closest one.

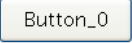
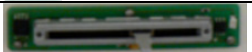
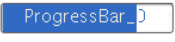
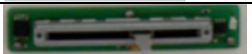
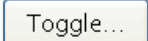
AuiInteractor	Digital world	Physical world
PushButton		
Slider		
CheckBox	☐ CheckBox_0	
RadioButton	☐ RadioButt... ☐ RadioButt... ☐ RadioButt...	  
TextArea		
ProgressBar		
ToggleButton		
ListBox	 	
Rotator		

Figure 14. Some mappings between GUI and TUI elements.

Figure 15 shows the two families of input devices supported by MIXEDSKETCH: tactile screens (fig. 15a) and graphic tablets (fig. 15b). Graphic tablets impose less physical arm fatigue than tactile screens, but are intrinsically indirect input devices: the end user only sees the results on the main screen, not on the tablet.

CONCLUSION AND FUTURE WORK

Regarding to the MIXEDSKETCH tool, we demonstrated that it provides a user interface that is specifically de-signed to support hand gestures on a graphics tablet or any other interactive surface, such as electronic whiteboards. Whiteboards usually fall into two categories: Integrated back-projected electronic whiteboards and Modular front-projected electronic whiteboards. Gestures are pen strokes that are drawn by the user with a pen or mouse, and which are then interpreted by the program and replaced by a corresponding symbol (e.g., a widget representation, a diagram symbol).



Figure 15. Typical devices supported by MIXEDSKETCH.

For instance, if one sketches the outline of a double box in MIXEDSKETCH, the tool will immediately interpret this as the gesture for a push button and replace this stroke with the representation corresponding to this widget. This representation may vary depending on the level of fidelity: none, low, medium, or high. The designer can then explore alternative designs by switching from one world to another. This switching is supported thanks to an ontology of abstract interactors, along with their relationships, that is maintained and exploited by MIXEDSKETCH. Currently, there are only two worlds (i.e., digital and physical).

Thanks to this ontology, one can imagine any new representation world that would introduce a new dimension with respect to the other ones. This new world should then be introduced in the ontology in terms of new element connected to an *AuiInteractor* along with its facets, its relationships, its representation in Hi-Fi (multiple representations of the same interactor are possible depending on some conditions to be stated), and the name of a corresponding element for export (e.g., the name of the objet in Voodoo), and its basic properties (e.g., dimensions, location, format).

In this way, we demonstrated that any person who is interested in sketching can actually draw a sketch of objects, regardless the world they are input (object recognition on multiple inputs), see them refined (multi-fidelity) and rendered (multi-rendering). Any object can be represented in the worlds it is supported (multi-world representation). Any object input in any world can initiate an object in any other

corresponding world, provided that such a transition via the ontology exists (transition between worlds), and then be exported in a XML-compliant format for being recuperated in a toolkit. This transition ensures consistency between designs across worlds since the software naturally maintains this property thanks to the notion of behavior equivalence. When this property cannot be established, the closest one is selected instead. When objects belong to the digital world, the prototype covers main aspects of GUI prototyping, but for multi computing platforms with the ability to switch from one fidelity (e.g., the level at which the GUI was drawn) to another (e.g., Mi-Fi, Hi-Fi). When all objects belong to the physical world, the prototype covers main aspects of UI prototyping for ambient intelligence, home automation, virtual reality, etc. When both types of objects are present, the UI being prototyped is said to be mixed since it combines objects from both worlds.

REFERENCES

1. Bailey, B.P. and Konstan, J.A. Are informal tools better? Comparing DEMAIS, pencil and paper, and Authorware for early multimedia design. In *Proc. of CHI'03*, ACM Press, New York (2003), 313-320.
2. Ballagas, R., Ringer, M. Stone, M., and Borchers, J. iStuff: a physical user interface toolkit for ubiquitous computing environments. In *Proc. of CHI'2003*, ACM Press, New York (2003) 537-544.
3. Caetano, A., Goulart, N., Fonseca, M., and Jorge, J. JavaSketchIt: Issues in Sketching the Look of User Interfaces. In *Proc. of AAAI'02 Spring Symp. on Sketch Understanding*, AAAI Press, Menlo Park, 9-14.
4. Fitzmaurice, G., Ishii, H. and Buxton, W. Bricks: Laying the Foundations for Graspable User Interfaces. In *Proc. of CHI'95*, ACM Press, NY (1995) 442-449.
5. Fonseca, M., Jorge, J., and Garcia, F.M. Visual Syntax Analysis for Calligraphic Interfaces. In *Proc. of 13th Encontro Português de Computação Gráfica*, Universidade Trás-Os-Montes e Alto Douro (Vila Real, Oct. 2005).
6. Gellersen, H. Smart-Its: computers for artifacts in the physical world. *Com. of the ACM* 48(3), 2005, 66.
7. Greenberg, S and Fitchett, C. Phidgets: easy development of physical interfaces through physical widgets. In *Proc. of UIST'01*, ACM Press (2001), 209-218.
8. Hartmann, B., Klemmer, S. R., Bernstein, M., and Mehta, N. d.tools: Visually Prototyping Physical UIs through Statecharts. In *EA of UIST'2005*, ACM Press (2005).
9. Hong, J.I. and Landay, J.A. Satin: a toolkit for informal ink-based applications. In *Proc. of UIST'00*, 63-72.
10. Klemmer, S.R., Integrating Physical and Digital Interactions. *IEEE Computer* (2005) 111-113.
11. Landay, J.A. and Myers, B.A. Interactive Sketching for the Early Stages of User Interface Design. In *Proc. of CHI'95*, ACM Press, New York (1995), 43-50.
12. Lin, J., Thomsen, M., and Landay, J.A. A visual language for sketching large and complex interactive designs. In *Proc. of CHI'02*, ACM Press, 307-314.
13. Lin, J. and Landay, J.A. Employing patterns and layers for early-stage design and prototyping of cross-device user interfaces. In *Proc. of CHI'2008*, pp. 1313-1322.
14. McCurdy, M., Connors, C., Pyrzak, G., Kanefsky, B., and Vera, A. Breaking the Fidelity Barrier: An Examination of our Current Characterization of Prototypes and an Example of a Mixed-Fidelity Success. In *Proc. of CHI'06*, ACM Press, New York (2006), 1233-1242.
15. Pedersen, E. Sokoler, T. Nelson L. PaperButtons: expanding a tangible user interface. In *Proc. of DIS'00*, ACM Press, New York (2000) 216-223.
16. Plimmer, B.E. and Apperley, M. Interacting with Sketched Interface Designs: an Evaluation Study. In *Extended Proc. of CHI'04*, ACM Press, pp. 1337-1340.
17. Rudd, J., Stern, K., and Isensee, S. Low vs. high-fidelity prototyping debate. *Interactions* 3(1), 1996, 76-85.
18. Sefelin, R., Tscheligi, M., and Giller, V. Paper Prototyping – What is it Good for? A Comparison of Paper-and Computer-based Prototyping. In *Proc. of CHI'03*, ACM Press, New York (2003), 778-779.
19. Signer, B. and Norrie, M.C. PaperPoint: A Paper-Based Presentation and Interactive Paper Prototyping Tool. In *Proc. of TEI'07*, ACM Press, New York (2007) 57-64.
20. Snyder, C. *Paper Prototyping: The Fast and Easy Way to Design and Refine User Interfaces*. Series in Interactive Technologies, Morgan Kaufmann, 2002.
21. Spiessl, W., Villar, N., Gellersen, H., and Schmidt, A. VoodooFlash: authoring across physical and digital form. In *Proc. of TEI'2007*, ACM Press (2007), 97-100.
22. Tohidi, M., Buxton, W., Baecker, R., and Sellen, A. User Sketches: a Quick, Inexpensive, and Effective Way to Elicit more Reflective User Feedback. In *Proc. of NordiCHI'06*, ACM Press, New York, 105-114.
23. Van Laerhoven, K., Villar, N., Schmidt, A., Gellersen, H., Håkansson, M., and Holmquist L.E. Pin&Play: The Surface as Network Medium. *IEEE Communications Magazine* 41(4), April 2003, 90-96.
24. Villar, N., Gilleade, K., Raymundo-Ellis, D., and Gellersen, H. The VoodooIO Gaming Kit: A Real-Time Adaptable Gaming Controller. In *Proc. of ACE'06*, ACM Press, New York (2006).
25. Villar, N. and Gellersen, H. A Malleable Control Structure for Softwired User Interfaces. In *Proc. of TEI'07*, ACM Press, New York (2007) 49-56.
26. Virzi, R.A., Sokolov, J.L., and Karis, D. Usability problem identification using both low- and high-fidelity prototypes. In *Proc. of CHI'96*, ACM Press, 236-243.
27. Walker, M., Takayama, L., and Landay, J. High-fidelity or Low-fidelity, Paper or Computer medium? In *Proc. of HFES'02*, HFES, Santa Monica (2002), 661-665.

GUILayout++: Supporting Prototype Creation and Quality Evaluation for Abstract User Interface Generation

Francisco Montero and Víctor Manuel López Jaquero

Laboratory of User Interaction & Software Engineering (LoUISE)

Instituto de Investigación en Informática (I3A)

University of Castilla-La Mancha, 02071 – Albacete (SPAIN)

{ fmontero | victor }@dsi.uclm.es

ABSTRACT

User interface development is not an easy task. One of the key elements is tackling user interface design by using user-centered development (UCD) techniques, more specifically user interface prototyping and iterative design based on evaluation. In this paper, some facilities are introduced to support the aforementioned activities. These facilities have been included in a tool called GUILayout. The new tool created has been called GUILayout++. By using the extra functionality it is possible creating high or low-fi user interface prototypes, analyze the structure and the look of the user interface according to the different areas that it comprises, and compute the Layout uniformity metric proposed by Constantine & Lockwood. Furthermore, starting with the prototypes created it is possible to automatically generate an abstract user interface compliant with UsiXML (<http://www.usixml.org> [23]) user interface description language.

Keywords

User interfaces, prototyping, UI evaluation, UI design.

INTRODUCTION

The user interface is a set of commands or menus the user uses to communicate with a program [15]. The main idea behind user interface concept is the mediation between the human and the machine. In a broad sense, the user interface is the link between the user and the software, that it, is the mean for the user to communicate with the machine.

This paper elaborate on the idea that in order to develop any software application with quality, the user must be considered from the very beginning of development, to avoid errors or flaws in the product created. This design philosophy is known as user centered design [18], and remarks the fact that interaction must be seen from the user's point of view [6].

A paramount ingredient to carry out this design philosophy has been using prototypes, that is, partial or total representations of the final product (ranging from simple mockups to more elaborated designs created by using software tools) that support, along the full process, the implication of the end-user in the design activities.

Using prototypes will support the designers in communicating with the end-users in a more effective way. Thus, from the very beginning prototypes will be created (more

elaborated each time) that will be presented to the users. Users will interact with these prototypes as they would with a final product.

In this paper, the need to jointly consider prototyping and evaluation in the development of user interfaces is exposed. To do so, analysis and evaluation criteria for different kinds of prototypes are identified. To support considering jointly prototyping and evaluation activities in user interface design a tool has been created, namely GUILayout++.

This paper is organized as follows. First, different tools and methodologies supporting prototyping are discussed. Next, those techniques and evaluation and analysis metrics that have been proposed are gathered. GUILayout++ tool is described next. Lastly, some conclusions and future work are provided.

PROTOTYPING TOOLS AND METHODOLOGIES

There are several taxonomies to describe the different kinds of prototyping [19]. Regarding their fidelity degree prototypes can be either high-fidelity or low-fidelity. The look of low-fidelity prototypes (see Fig. 1) is far from being closed to the look the final application will exhibit. Nevertheless, high-fidelity prototypes (see Fig. 2) look more alike the final product.

There are some other terms related to prototyping methods. For instance, there is quick prototyping and modular prototyping methods. Quick prototyping consist on developing quickly new designs, evaluate them and discard the current prototype to create a new prototype for the next design iteration. Modular prototyping seeks reusing prototypes by modifying an existing prototype as the design cycle goes on. For this reason, this prototyping method is also known as incremental prototyping.



Figure 1. Example of low-fidelity prototype.



Figure 2. Example of high-fidelity prototype.

The importance of prototyping activities for user interface design is evidenced by its inclusion in many user interface development methodologies. These methodologies are supported by different tools and software products coming from both academia software companies. Next, an overview of some methodologies and tools where prototyping and evaluation are considered is provided.

Methodologies

Usability engineering methodologies are software development methodologies with a special emphasis on optimizing the usability of the system that is to be developed. User interface prototyping activity with different degrees of detail is often considered in many methodological proposal included in Usability Engineering. Examples include the Star Lifecycle [6] Usability Engineering [17], LUCID (Logical User-Centered Interface Design Method) [21], the Usability Engineering Lifecycle [13], Usage-Centered Design [4], MUSE (Method for Usability Engineering) [10] and ACUE approach [19].

Examples of usability engineering techniques include affinity diagramming and Canonical Abstract Prototypes [4], evaluation activities and usability inspections [17], heuristic evaluation [17], and task analysis and different kinds of prototypes [13]. In these methodologies, prototyping is a recurrent activity for user interface development.

Tools

Prototyping has been used by many developers [22], and it is supported by several tools. Some meaningful examples from academia are: GUILayout [2], DENIM [11], CanonSketch [3], SketchiXML [5]. There are also some other commercial tools such as Pencil (<http://www.evolus.vn/pencil/>) or SketchFlow (<http://www.microsoft.com>).

In some of the aforementioned tools, for instance in DENIM, experience has been integrated into prototyping, but no tool supports assessing or analyzing the prototypes created by using these tools.

In this paper the joint consideration of prototyping and analysis is proposed. Nevertheless, to support the evaluation of the prototypes the identification of some quality criteria is required. In the next section, some quality criteria for prototypes are discussed.

QUALITY METRICS IN PROTOYIPING

In user interface development methodologies considering usability there are plenty of quality principles and criteria than can be considered. Shneiderman [2], Nielsen [16, 17], Norman [18], Mayhew [12], Constantine and Lockwood[4], Tognazinni (www.asktog.com) and Bastien and Scapin [1], are probably the most recurrently cited in the literature.

The problem is that those criteria are hard to apply and evaluate automatically, and designer experience plays an important role when putting them in practice and assessing whether they are achieved or not.

We have identified some user interface analyzing activities that can be automated to contribute to the evaluation of user interfaces, and that can be used for prototyping assessment. These analyzing activities will be described in depth in the next section, and they were proposed in [16] and [4].

Zone Analysis by Nielsen and Tahir

In the book “Homepage Usability. 50 Websites Deconstructed” [16], the authors describe several heuristic usability evaluations of different websites. For this evaluation the authors consider some style guidelines (113) defined at the homepage level of websites. In the book, they analyzed also the use and availability of the different zones of the homepage of each website. Thus, the authors analyzed the purpose each zone was devoted to in the homepage.

Color	Purpose	Ideal value
Blue	Content of interest	6%
Green	Advertising and sponsorship	25%
Cyan	Self promotional	25%
Orange	Filler	6%
White	Unused	2%
Yellow	Welcome and site identity	11%
Pink	Navigation	25%

Table 2. Breakdown of screen real state.

The purpose for the different zones in a website for real state is shown in Table 1. The purpose of each zone can be specified when prototyping the user interface in GUI-Layout++.

An Structural metric by Constantine and Lockwood

Constantine and Lockwood [4] identified several flavors of design metrics for user interfaces: structural metrics (which are based on surface properties), semantic metrics (which are content sensitive) and procedural metrics (which are task sensitive).

The simplest to compute are those measures called structural metrics, which are based on those surface properties of the configuration and layout of user interface architectures that can simply be counted or toted. In this sense, not every designer who ends up responsible for user interface design necessarily has a graphic designer’s eye for layout.

Layout uniformity is a structural metric that gives a quick handle on one important aspect of visual layout.

Layout Uniformity (LU) measures only selected aspects of the spatial arrangement of interface components without taking into account what those components are or how they are used. LU is based on the rationale that usability is hindered by highly disordered or visually chaotic arrangements.

$$LU = 100 \left(1 - \frac{(N_h + N_w - N_e + N_t + N_b + N_r) - M}{6N_c - M} \right)$$

where N_c = total number of visual components on screen, dialog box or other interface composite and N_h, N_w, N_e, N_t, N_b and N_r are, respectively, the number of different heights, widths, top-edge alignments, left-edge alignments, bottom-edge alignments, and right-edge alignments of visual components. M is an adjustment of the minimum number of possible alignments and sizes needed to make the value of LU range from 0 to 100. $M = 2 + 2\lfloor 2\sqrt{N_{components}} \rfloor$.

As a structural metric concerned only with appearance, Layout Uniformity should not be given undue weight in evaluating designs. It can, however, be useful to the designer who lacks an eye for layout to know when a visual arrangement might be improved.

This metric, Layout Uniformity, was implemented and included in GuiLayout++.

GUI LAYOUT++: PUTTING TOGETHER PROTOTYPING AND EVALUATION OF USER INTERFACES

GuiLayout++ improves GuiLayout tool. This tool was developed by Kai Blankenhorn [2] for his master thesis. This work identified the limitations that Unified Modeling Language (UML) provides for the specification of user interface. To overcome the identified limitations he created a UML profile for user interface development and for web site prototyping.

GuiLayout is a profile that provides an easily comprehensible abstract representation of an actual screen based on designers' sketches. A GUI Layout Diagram consists of a Screen, which contains multiple *ScreenAreas*. Each may be decorated with one or more Stereotypes, representing performed functionalities like text, image or link. By nesting and arranging properly stereotyped ScreenAreas within each other, the developer is able to create an abstract version of a user interface (see Fig. 3). The Navigational Diagram provides UML-based support for common design artifacts like storyboards and sitemaps.

GuiLayout represents an interesting contribution in the scope of user interface prototyping, and a complement for the different diagrams defined for UML. Nevertheless, this tool can be improved with evaluation and analysis facilities for software products that assist designers in the development of the user interface, which is always a hard task. Furthermore, as aforementioned, some analysis and metrics

can be automatically computed to help in the different iterations of the design process.

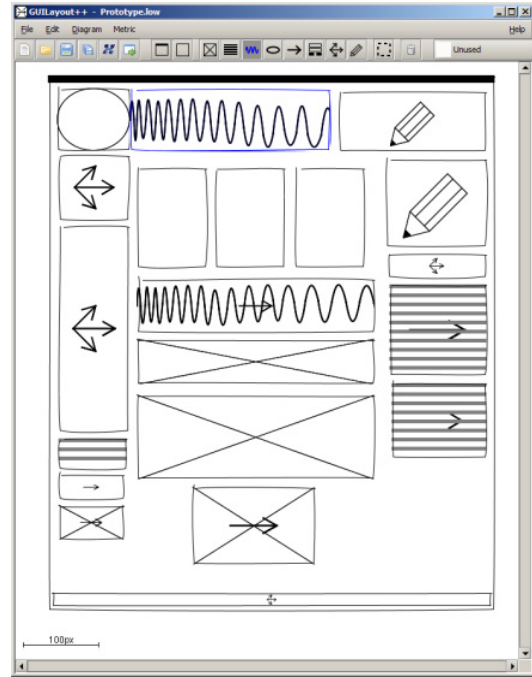


Figure 3. GuiLayout tool and low-fidelity prototyping.

Using GuiLayout++

One of the improvements GuiLayout++ provides with respect to GuiLayout is supporting low-fidelity prototyping while visually specifying, by using color codes, the purpose each area or zone is devoted to (see Fig. 4).

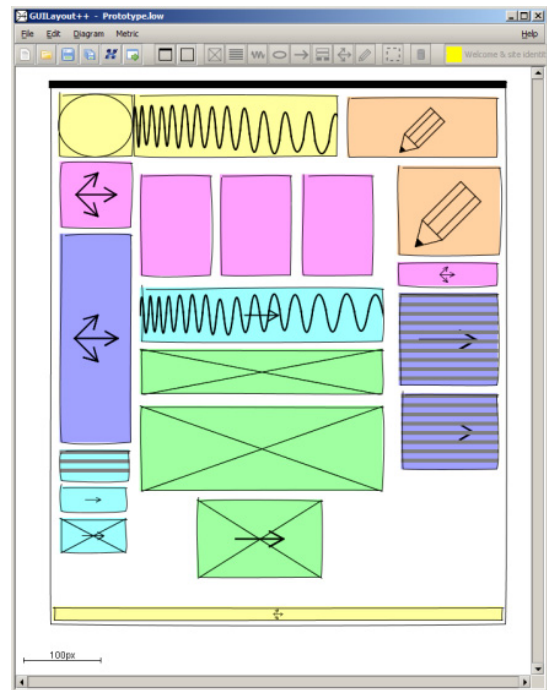


Figure 4. GuiLayout++ and low-fidelity prototyping.

By using the color codes, designers can identify visually and numerically the space devoted to the different concepts described by Nielsen and Tahir [16] (see Table 1), but working with a low-fidelity prototype. The resulting values can be compared with the ideal values for each application domain. Depending on how the resulting values match the ideal values they are tagged as: 😊 *ok*, 😟 *warning* or 😡 *error* (see Fig. 5). The ideal values used in the tool can be changed to match the application domain of the user interface designed.

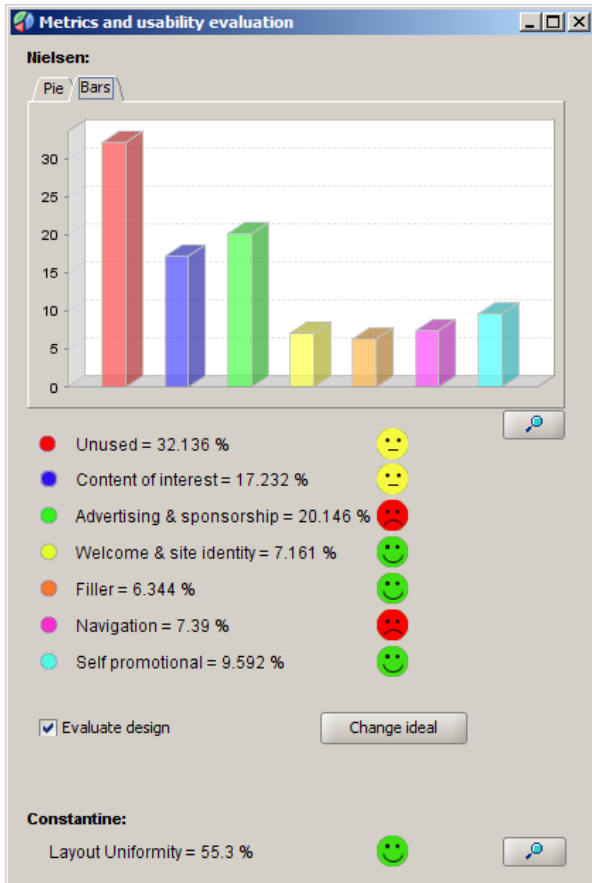


Figure 5. GUILayout++ and high-fidelity prototyping and metrics

Another facility provided by GUILayout++ is computing metrics related to structural design features found in the prototype. Currently GUILayout++ support computing *Layout Uniformity* metric, as proposed by Constantine and Lockwood [4] (see Fig. 6). Additional metrics could be considered in GUILayout++, for instance, [24]. Another UI metrics, for instance Ivory [7, 8], are not useful in this version of GUILayout++. GUILayout++ cannot only work on low-fidelity prototypes, but also on high-fidelity prototypes. By using screenshots the designers can use high-fidelity prototypes to carry out the analysis previously described. This other way of working with GUILayout++ is illustrated in Fig. 7. In this figure, a screenshot of the web for tourism of Wallone region in Belgium

(<http://www.wallonie.be/fr/index.html>) has been used. Then the designer has marked the different zones in the website, specifying the purpose of each zone.

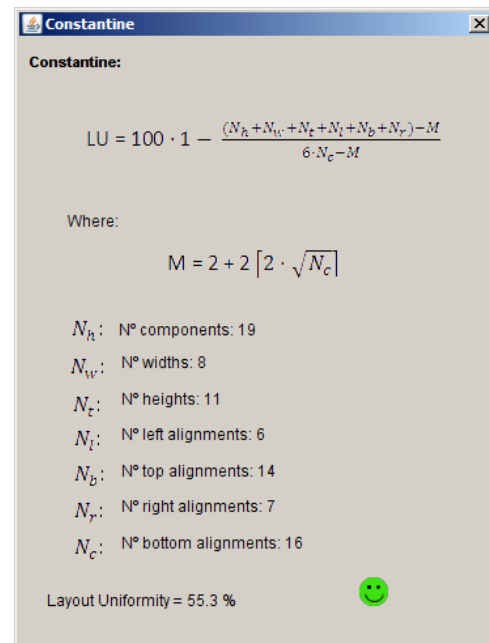


Figure 6. Layout Uniformity metric in GUILayout++.



Figure 7. GUILayout++ and high-fidelity prototyping.

Generating usiXML Abstract User Interface

Creating prototypes contributes to the subsequent user interface development, or to its refinement. In GUILayout++ the designer can use the prototype created in the user interface development process by storing the prototype using the abstract user interface components proposed for UsiXML abstract user interface model. These components are represented visually by using the graphical notation for the abstract user interface model proposed in IdealXML [14] (see Table 2). The abstract user interface for the high-fidelity prototype in Fig. 7 is shown in Fig. 8.

Image	Element
	Container
	Component
	Input
	Output
	Control
	Navigation

Table 3. Abstract User interface elements defined in UsiXML.

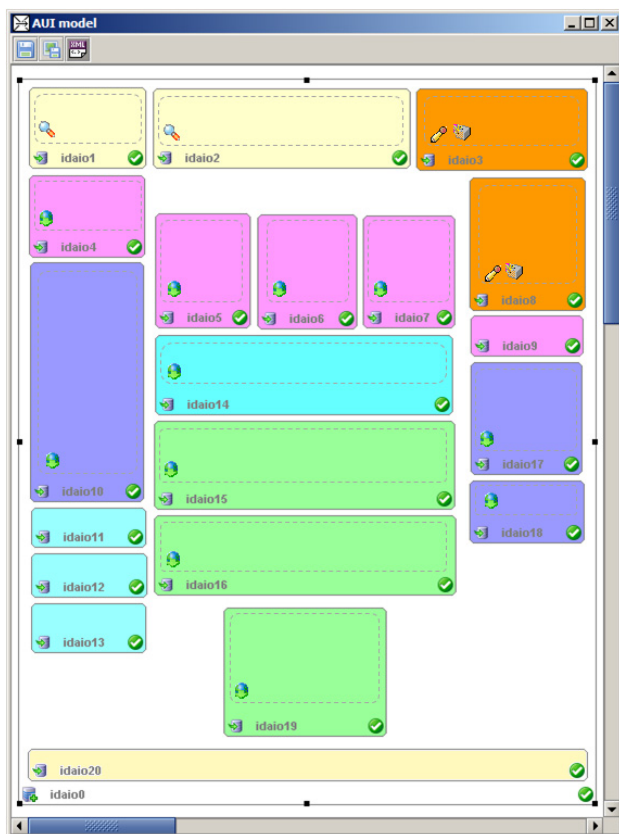


Figure 8. Abstract user interface specification by using UsiXML.

Starting from the abstract user interface generated by GUI-Layout++ the designer can follow different development paths for the user interface design, as described in [9]. For

instance, the designer can derive the task model by using abstraction transformations, or he can go on refining the abstract user interface to generate a concrete user interface, that latter will be transformed into the final user interface the user will interact with.

CONCLUSION AND FUTURE WORK

In this paper the need to combine two activities, prototyping and evaluation, when tackling user interface design is exposed. In this sense, we have identified that those activities are usually included in many user centered or Usability Engineering methodologies available in the literature. In the scope of user interface evaluation several metrics haven identified that can be considered in multiple prototyping tools. In this sense, some of them have been included in the prototyping tool GUILayout, to create GUILayout++. Currently, we are including additional metrics in the prototyping tool so behavioral and navigational aspects are considered also in GUILayout++.

ACKNOWLEDGMENTS

This work is partly supported by the Spanish PEII09-0054-9581 and CICYT TIN2008-06596-C02-01 grants. Also, we gratefully acknowledge the support of the ITEA 2 Call 3 UsiXML European project under reference #2008026.

REFERENCES

1. Bastien, J., Scapin, D. Preliminary findings on the effectiveness of ergonomic criteria for the evaluation of human-computer interfaces. In *Proc. of INTERCHI Adjunct Proceedings 1993*: 187-188
2. Blankenhorn, K.: A UML Profile for GUI Layout, Master's Thesis, University of Applied Sciences Furtwangen, Department of Digital Media (2004)
3. Campos, P. Nunes, N. J. CanonSketch: a User-Centered Tool for Canonical Abstract Prototyping, In *Proceedings of EHCI-DSV-IS'2004*, Hamburg, Germany, 2004.
4. Constantine, L., Lockwood, L. *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, 1999.
5. Coyette, A., Vanderdonckt, J., and Limbourg, Q. SketchiXML: A Design Tool for Informal User Interface Rapid Prototyping. In *Proc. of Int. Workshop on Rapid Integration of Software Engineering techniques RISE'2006* (Geneva, 13-15 September 2006). Lecture Notes in Computer Science, Vol. 4401, Springer-Verlag, Berlin, 2007, pp. 160-176.
6. Hix, D., Hartson, H.R. *Developing user interfaces: ensuring usability through product & process*. Wiley, 1993.
7. Ivory, M., Sinha, R., Hearst, M.: Empirically validated web page design metrics. In *Proc. of CHI 2001*: 53-60.
8. Ivory, M., Sinha, R., Hearst, M.: Statistical profiles of highly-rated web sites. In *Proc. of CHI 2002*: 367-374.

9. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., and Lopez-Jaquero, V. UsiXML: a Language Supporting Multi-Path Development of User Interfaces. In *Proc. of 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems EHCI-DSVIS'2004* (Hamburg, July 11-13, 2004). Lecture Notes in Computer Science, Vol. 3425, Springer-Verlag, Berlin, 2005, pp. 200-220.
10. Lim, K.Y, Long, J. The MUSE method for usability engineering. Cambridge ; New York : Cambridge University Press, 1994.
11. Lin, J., Newman, M. W., Hong, J. I., Landay, J. A. DENIM: finding a tighter fit between tools and practice for Web site design. In *Proc. of CHI 2000*: 510-517
12. Mayhew, D. J. (1999). The usability engineering lifecycle: A practitioner's handbook for user interface design. San Francisco, California
13. Mayhew, D. J., Principles and Guidelines in Software User Interface Design, Prentice-Hall, 1992
14. Montero, F., López-Jaquero, V. IdealXML: An Interaction Design Tool. In *Proc. of CADUI 2006*: 245-252
15. Myers, B. and Kosbie, D. Reusable Hierarchical Command Objects. In *Proceedings of CHI'96: Human Factors in Computing Systems*, Vancouver, BC, Canada, April 14-18, 1996.
16. Nielsen, J. and Tahir, M.: Homepage Usability: 50 Websites Deconstructed. New Riders Press. Nov. 15, 2001.
17. Nielsen, J. Usability engineering. Boston, MA: AP Professional, 1993.
18. Norman, D. A. (1988). The psychology of everyday things. New York: Basic Books.
19. Seffah A., and Metzker, E. Usability Engineering Methods Plethora. Adoption-centric Usability Engineering Systematic Deployment, Assessment and Improvement of Usability Methods in Software Engineering. Springer.
20. Shneiderman, B. Designing the User Interface: Strategies for Effective Human-Computer Interaction, 1st edition. Addison-Wesley. 1987
21. Smith, A., Dunckley, L. User Centred Design: The Application of the LUCID Interface Design Method. HCI (2) 1997: 563-568
22. Szekely, P. User Interface Prototyping: Tools and Techniques. Technical report, Intelligent Systems Division, University of Southern California. 1994
23. Usixml. <http://www.usixml.org>.
24. Vanderdonckt, J., Gillo, X., Visual Techniques for Traditional and Multimedia Layouts, in *Proc. of 2nd ACM Workshop on Advanced Visual Interfaces AVI'94* (Bari, 1-4 June 1994), T. Catarci, M.F. Costabile, S. Levialdi, G. Santucci (eds.), ACM Press, New York, 1994, pp. 95-104.

A Framework for Accessibility Testing of Virtual Environments based on UsiXML

Nikolaos Kaklanis^{1,2}, Konstantinos Moustakas², Konstantinos Votis² and Dimitrios Tzovaras²

¹ Department of Computing, University of Surrey
Guildford, United Kingdom

² Informatics and Telematics Institute, Centre for Research and Technology Hellas
6th Km Charilaou-Thermi Road, 57001 (PO Box 60361)
Thermi-Thessaloniki, Greece
{nkak, moustak, kvotis, Dimitrios.Tzovaras}@iti.gr

ABSTRACT

The present paper introduces a framework that supports accessibility testing at all the stages of the development of mainstream ICT and non-ICT technologies. The main innovation of the proposed framework lies in the fact that, the whole framework is based on a new virtual user modelling technique including physical, cognitive and behavioral/ psychological aspects and parameters of the user with disability. The proposed technique will not be constrained in static modelling of the virtual user but will generate a dynamic and parameterizable user model including interactions that can be used directly in simulation frameworks, adaptive interfaces and optimal design evaluation frameworks.

Keywords

User Modeling, virtual user model, accessibility testing, UsiXML

INTRODUCTION

It is important to realise that people with disabilities are not just a tiny minority of the population of the European Union. The lowest estimate, based on the currently defined disablement categories, estimates their total number at around 74 Million persons. However, other estimates that take into account a) people with cognitive difficulties, and b) those people in the so-called hinterland between fully able bodied and the classically termed people with disabilities, should considerably raise those numbers, as highlighted in table 1.

With increasing life expectancy, visual and hearing impairments also increase, as well as neurological disorders such as Alzheimer's disease and dementia. In addition, we need to consider that the composition of the population is changing. Many people are surviving permanently disabling accidents and illness and even more are living longer. It would seem logical that the spaces built to accommodate this population must, by necessity, change also. However, future trends in age-specific risks of becoming hampered will be a key factor in the number of older people that will be in need of assistance and care. As a result of this demographic trend EU Countries are expected to experience significant social and economical impacts, with enormous effects on welfare expenditures and in particular on employ-

ment and labour markets, in pension systems and in healthcare systems.

In the EU 27 countries about 16% of the population are over 65, a number that is estimated to rise rapidly in the coming years⁸.

Up to 15% of the population across the European Union has a disability, such as a visual, hearing, speech, cognitive, or motor impairment⁹.

Around 20% of people over 50 experience severe physical disabilities

Spending on pensions, health and long-term care will rise sharply over the next 20 years

Table 4. Statistics about the elderly and disabled

Despite the rapid evolution of ICT over the last years and the increasing acknowledgment of the importance of accessibility, the developers of mainstream ICT-based products still act and struggle under total absence of structured guidance and support for adjusting their envisaged products and services with their user's real-time accessibility needs. As a result, a critical mass market, including that of older people and people with disabilities-friendly ICT-based products and services targeting older people and people with disabilities, remains highly locked.

A similar situation is observed in the development of non-ICT products and services, where developers toil to test and evolve their prototypes in terms of their functionality, without however being able to systematically test their developments in terms of their accessibility.

Thus, it is a technological challenge to provide senior citizens with systems that could foster the different facets in the perception of quality of life. These systems should improve the level of independence, promote the social relationships, leverage the immersion in the environments and encourage the psychological and physical state of the person.

⁸ Eurostat yearbook 2008.

⁹ Report of the Inclusive Communications (INCOM) subgroup of the Communications Committee (COCOM) COCOM04-08.

The European Disability Action Plan priorities for 2008-2009 include a focus on accessibility and emphasise that 'Accessible goods, services and infrastructures are essential to sustain non-discriminatory and inclusive forms of participation in many aspects of everyday life', and that 'Achieving accessibility requires the avoidance and removal of barriers that prevent people with disabilities from exercising their capabilities and participating fully and on equal terms'. An estimation regarding the user demand for accessible ICT products, services and assistive technologies among the EU (50+ population) has been examined by the European study "The Demographic Change – Impacts of New Technologies and Information Society" (following table 2). As depicted in this table, designing and developing for people with disabilities is becoming an increasingly important topic for a variety of reasons.

Indicator of potential market size		Demand potential in Mio		
Indicator for need	Degree of impairment	2010	2020	2050
Vision problems	slight/moderate	46.3	53.1	59.1
	severe	20.5	23.5	26.2
Hearing problems	slight/moderate	44.4	51.0	56.7
	severe	8.5	9.8	10.9
Dexterity problems	slight/moderate	32.5	37.2	41.4
	severe	17.1	19.6	21.8
More than one of these	slight/moderate	73.5	84.3	93.7
	severe	35.9	41.2	45.8

Source: Own calculation demographic data available from SEN-IORWATCH 2002 and demographic projections from Eurostat 2005

Table 5 User demand for accessible ICT products, services and assistive technologies among the EU 50+ population [From Study "The Demographic Change – Impacts of New Technologies and Information Society" http://ec.europa.eu/employment_social/social_situation/studies_en.htm].

The lack of non accessible products can cause large productivity losses, with many people being unable to fully participate at work, in education, or in a wide range of economic and social activities. People's choice of leisure activities may be narrower than it otherwise could be. The cost of making products and services more inclusive need not be very high. The lack of progress on accessibility reflects the current fragmented approaches to producing accessible

products and services, which rather limit their economic potential and create a barrier to a thriving single market for them in Europe.

The core concept of the proposed framework is to support accessibility testing at all the stages of the development of mainstream ICT and non-ICT technologies. The main innovation of the proposed framework lies in the fact that, the whole framework is based on a new virtual user modelling technique including physical, cognitive and behavioral/ psychological aspects and parameters of the user with disability. The proposed technique will not be constrained in static modelling of the virtual user but will generate a dynamic and parameterizable user model including interactions that can be used directly in simulation frameworks, adaptive interfaces and optimal design evaluation frameworks.

RELATED WORK

User Modelling

User models are represented in the literature using different syntaxes and implementations, varying from flat file structures and relational databases to full-fledged RDF with bindings in XML.

The notion of ontology-based user models was first developed by Razmerita et al. in 2003 [19] that presented a generic ontology-based user modelling architecture called OntobUM. OntobUM integrated three ontologies: a user ontology characterizing the users, a domain ontology defining the relationships between the personalization applications, and a log ontology defining the semantics of user-application interaction. A similar, but way more extensive approach for ontology-based representation of the user models was presented by [9]. In [8] GUMO is proposed, which seems to be the most comprehensive publicly available user modelling ontology to date. The need for a commonly accepted ontology for user models is also justified. These works are natural extensions of earlier works on general user modelling systems [10], [13], [14]. Such a general user model ontology may be represented in a modern semantic web language like OWL, and thus be available for all user-adaptive systems.

XML-based languages for user modelling have also been proposed. UserML has been introduced in [7] as user model exchange language. A central conceptual idea in UserML's approach is the division of user model dimensions into the three parts *auxiliary*, *predicate* and *range*. For example, if one wants to say *something about the user's interest in football*, one could divide this so-called *user model dimension* into the auxiliary part *has interest*, the predicate part *football* and the range part *low-medium-high*. Additionally, further important meta attributes have been identified for the user modeling domain, like the *situation* (like start, end, durability, location and position), *privacy* (like key, owner, access, purpose, retention) and *explanation* (like creator, method, evidence, confidence).

Simulation tools

In the field of virtual reality and augmented reality (AR) applications with a focus on accessibility and ergonomics, the work that has been carried out in recent years although not prolific has been important nonetheless. A state of the art study in 2002 [2] compares the two main methods of accessibility evaluation through the use of VR, namely immersing the designer in the virtual world of the mechanical system and semi-automatically control mannequins through a predefined kinematical and ergonomic behaviour.

To that end, the HabiTest software tool developed by the Laboratory for Innovations in Rehabilitation Technology, at the University of Haifa [18] uses the first approach of user immersion. The concept of simulating working environments has drawn the interest of researchers a decade ago [15]. One of the first tools presented in the literature is the Deneb/ERGO software tool [17]. In [3] the authors make a step further and present a prototype for adapting workplaces for people with physical disabilities. The general usability of virtual reality technologies to workplace ergonomics is also studied in [11] and [5].

ACCESSIBILITY TESTING FRAMEWORK BASED ON USIXML

The main goal of the proposed framework is to support comprehensively virtual user modelling, simulation and accessibility testing at all development stages of novel products and applications.

	GOMS	GTA	TOOD	UsiXML	Relevance
Decomposition	Hierarchy	Hierarchy	Hierarchy	Hierarchy	Must
Sequence	Sequence	Seq	Sequence	Enabling, enabling with information passing	Must
Iteration	+ Loop (If, then, else)	X	X	Iteration, finite iteration	Should
Choice	+ Or (If, then, else)	Or	Choice	Deterministic choice, undeterministic choice, inclusive choice	Should
Optionality	Optional (If, then, else)	Start condition	X	Optional	Should

Interruption	Interruption (If, then, else)	Stop condition	Interruption	Suspend-resume, disabling, disabling with information passing	Should
Concurrency	Concurrency (If, then, else)	X	Concurrency	Independent concurrency, concurrency with information passing, order independence	Should
Cooperation	X	Cooperation	Collaboration	Cooperation	Should
Parallel	X	And	Simultaneity	parallelSplit (process model)	Should

Table 6. Task Models comparison [6].

The framework is based on UsiXML due to the following reasons:

- UsiXML can describe the tasks sufficiently and as presented in table 3, it can offer a much more complete description of tasks compared with other task modelling techniques like GOMS [1], GTA [4], TOOD [16], etc.
- As there is no widely accepted user modeling language that can adequately describe the elderly and disabled users, there could be an extension of UsiXML (that has already some primary support for the description of the user), in order to achieve the detailed description of the possible disabilities of the user as well as the affected by the disabilities tasks.

Figure 1 presents the architecture of the proposed framework consisted by the following components:

- *Abstract User Models*

Describe a large set of disabilities. Ontologies are used to provide a powerful and growing specification of the abstract user models. An abstract user model stored in the ontology includes the type of user disability, user capabilities, user needs, characteristics from cognitive user models, physical user models, behavioral & psychological user models, guidelines and standards.

- *Task Models*

Describe the tasks of the user using the *taskModel* element of UsiXML. A task model refers to the interaction and task completion capabilities of the virtual user, including several parameters for their detailed description.

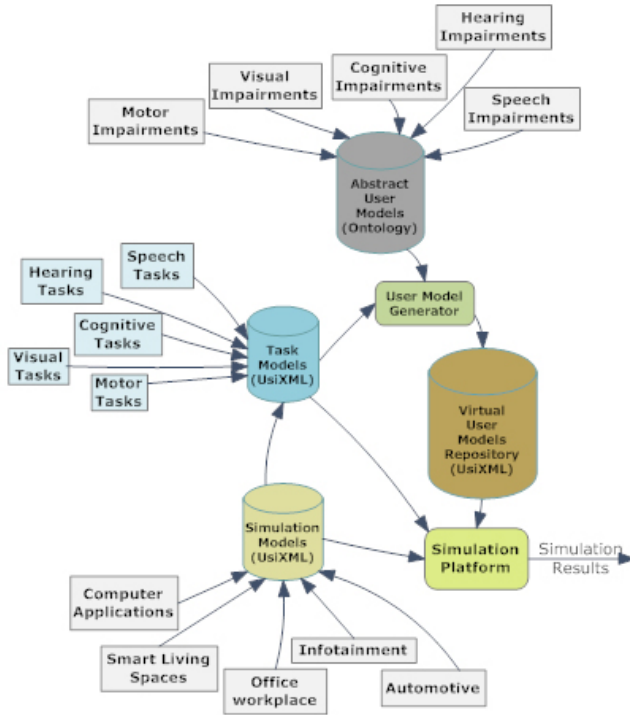


Figure 2. Framework's architecture.

- *Virtual User Models*

A Virtual User Model describes the disabilities of a specific user including special parameters for each disability, the affected tasks as well as the failure level of each affected task due to the disabilities. UsiXML is also used for the description of the Virtual User Models.

- *User Model Generator*

A software tool that enables the generation of a virtual user model in UsiXML format taking into account specific disability related parameters.

- *Simulation Models*

The simulation models refer to a sequence of primitive actions thus forming more complex actions that are defined by the designers/developers according to the functionality of the prototypes to be tested in terms of their accessibility. The *task-Model* element of UsiXML is used for the definition of the simulation models.

- *Simulation Platform*

The core module of the framework that performs the accessibility testing of a virtual environment for a specific virtual user. During the simulation phase, the virtual user models will aim to follow the simulation models (simulation scenario), so as to complete specific tasks defined by the design-

er/developer. The efficiency or capability of the virtual user to complete the actions defined in the simulation models will also depend on the content and parameters of the virtual user's problematic tasks due to specific disabilities.

A more detailed description for each of the above components follows in the next sections.

Abstract user models

The abstract user models describe a large set of disabilities including the type of the disability, user capabilities, user needs, characteristics from cognitive user models, physical user models, behavioural & psychological user models, guidelines and standards. The population of the abstract user models is based on:

- Analysis of existing physical user models with disabilities, cognitive disabled user models, behavioral and psychological user models with disabilities.
- Analysis of accessibility guidelines, methodologies and existing practices such as Human Factors (HF); Guidelines for ICT products and services; "Design for All" methodologies, etc. More specifically, appropriate principles, guidelines and standards for accessibility and universal design for various types of applications, services, goods and infrastructures, which are available from standardization organizations such as Mandate 376 "Accessibility requirements for public procurement of products and services in the ICT domain", the Mandate 420 "Accessibility of the Built Environment", ETSI EG 202 116 V1.2.1 (2002-09) ETSI Guide Human Factors (HF), "Guidelines for ICT products and services - Design for All", are taken into account.

Ontologies are being used to store the abstract user models. The advantage of using ontology for specification purposes gives a great potential to the models. Ontology can provide a common basis for communication and collaboration between heterogeneous artefacts and Aml environments [20].

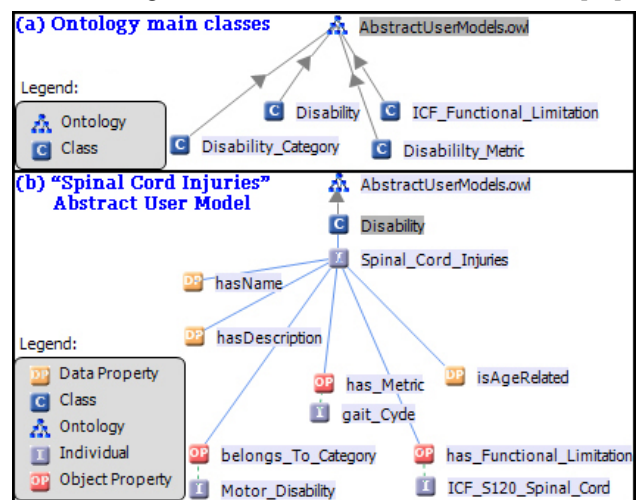


Figure 3. Abstract User Models - Ontology schema example.

The ontology can describe the basic conceptual terms, the semantics of these terms, and define the relationships among them. The use of ontologies to describe user models and their interrelationship also ensures the openness and the accuracy of the models specified. In case new abstract user models should be added, designers can extend existing abstract user models inheriting the properties of similar ones.

Figure 2 presents an example of the ontology schema of the abstract user models.

- An Abstract User Model contains:
 - The category of the disability
 - e.g. motor, visual, etc.
 - The name of the disability
 - e.g. hemiplegia, cerebral palsy, myopia, etc.
 - A short description of the disability
 - Quantitative disability metrics found in the literature
 - e.g. for “spinal cord injuries”, the “gait cycle” ranges between 1.12 sec and 3.22 sec
 - The corresponding functional limitations, according to the ICF classification
 - The relationship between the disability and the age
 - e.g. the disability appears in elderly only/not only, the disability is being affected while the user is growing older, etc.

Task models

The task models describe the interaction and task completion capabilities of the virtual user, including several parameters for their detailed description. The population of the task models is based on the analysis of:

- User Actions/Interactions: motor, cognitive and sensory abilities, such as the user’s ability to handle a specific task (e.g. to overcome physical obstacles such as narrow passageways in the construction area), to perceive visually presented information to select objects on a specific user interface, etc.
- Domain knowledge and relevant attributes with respect to the contents of application scenarios such as automotive, smart living spaces and building, domotics, infotainment, health.

The tasks are divided into *complex* and *primitive*. An example of a complex task could be the task of driving while a primitive task could be the opening of the fingers of the hand. For each complex task in the proposed framework a task model is being developed, in order to describe how the

complex task can be divided into primitive tasks (as they have been defined by the designers/developers according to the functionality of the prototypes to be tested in terms of accessibility). For the development of the task models of the proposed framework the *taskModel* element of UsiXML is being used.

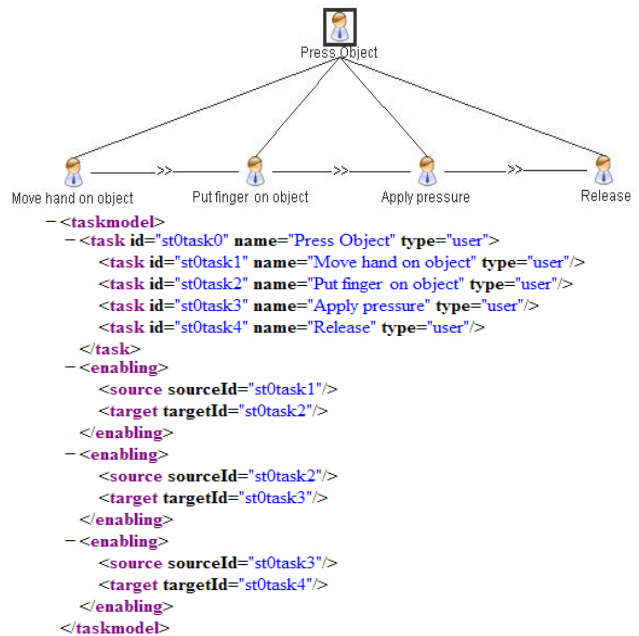


Figure 4. Task model example.

Figure 3 presents how the “Press Object” complex task can be divided into primitive tasks and how this can be described with UsiXML.

Virtual user models

A virtual user model describes a specific user including the disabilities and the problematic tasks due to the disabilities as well as the failure level for each task. A user model generator has been developed (Figure 4) in order to easily extract a virtual user model in UsiXML format.

Figure 5. Virtual user model generator.

As the main goal of UsiXML is to describe multimodal interfaces, it cannot sufficiently describe the user, including disabilities, special characteristics, etc.

UsiXML element/attribute	Original description	Meaning inside the virtual user model
<i>contextModel</i> element	A model describing the three aspects of a context of use in which a end user is carrying out an interactive task with a specific computing platform in a given surrounding environment. Consequently, a context model consists of a user model, a platform model, and an environment model.	A container for the disabilities.
<i>context</i> element	An instance of the context model.	Describes a disability.
“ <i>id</i> ” attribute of <i>context</i> element	The identifier of an instance of the context model.	The category of impairments where the disability belongs.
“ <i>name</i> ” attribute of <i>context</i> element	The name of an instance of the context model.	The name of the disability.
“ <i>id</i> ” attribute of <i>userStereotype</i> element	The identifier of a <i>userStereotype</i> , <i>UserStereotype</i> describes a set of users with similar characteristics.	Describes the special parameter(s) of the disability. [optional]
<i>taskModel</i> element	A model describing the interactive task as viewed by the end user interacting with the system. A task model represents a decomposition of tasks into sub-tasks linked with task relationships.	A container for the affected by the disabilities tasks.
<i>task</i> element	A task belonging to the task model.	Describes an affected by the disabilities task.
“ <i>name</i> ” attribute of <i>task</i> element	The name of a task.	The name of a task that is being affected by the user’s disabilities.
“ <i>complexityLevel</i> ” attribute of <i>task</i> element	Indicates an absolute complexity evaluation of a task. Task complexity is evaluated on a scale from 1 to 5. A value of 1 means that a task is trivial, 5 means that a task is very complex.	The failure level of the task due to the disabilities. [Accepted Values: 1 to 5 (1 means 20% failure, 2 means 40% failure, 3 means 60% failure, 4 means 80% failure and 5 means total failure)]

Table 7. The meaning of some UsiXML elements/attributes inside the virtual user model

Thus, we propose a UsiXML structure (Figure 5) for the virtual user model, in order to be able to describe user’s disabilities and the affected tasks, using some tags/attributes of UsiXML that have not been explicitly defined for that purpose. Table 4 presents the difference between the original description of some elements/attributes of UsiXML and their meaning in our proposed virtual user model. Figure 5 presents a virtual user model in UsiXML format describing a user with color blindness in red and Parkinson’s disease that results to 20% failure in reading a map and 60% failure in keyboard use.



Figure 6 Virtual user model example.

Simulation models

A simulation model refers to a sequence of primitive actions thus forming more complex actions defined by the designers/developers according to the functionality of the virtual environment to be tested in terms of its accessibility. For a better understanding, assume that there could be a simulation model describing the mobile phone use, including the following primitive actions (in a very abstract way for an easier understanding): grab mobile phone, type a telephone number, move mobile phone on ear, wait until someone answers, start conversation, etc..

The *taskModel* element of UsiXML is used for the definition of the simulation models. Figure 6 presents the simulation model of the “3D HapticWebBrowser”. The “3D HapticWebBrowser” is an application that enables haptic exploration on the web for the visually impaired users [12]. As depicted in figure 6, there are some *abstract tasks* (ex. “Explore current page”, “Go to another page”, “Use Options”, etc.), *application tasks* (ex. “View Options”, etc.) and *interaction tasks* (ex.: “Move Up (press UP-arrow)”, “Examine haptic’s surface using the Phantom haptic device”, etc.).

During the simulation phase the virtual user models aims to follow the simulation models (simulation scenario) so as to complete specific tasks defined in the simulation model. Thus, the efficiency or capability of the virtual user to complete the actions defined in the simulation model will also depend on the content and parameters of the virtual user's task model.

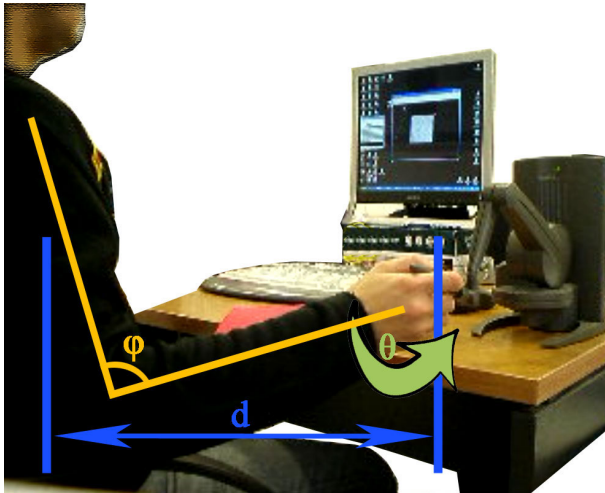


Figure 10 Simulation example (use the “Phantom Desktop” haptic device)

Currently, the simulation platform examines if low-level constraints are satisfied, in order to test the accessibility of a virtual environment for a virtual user. Figure 9 presents a simple example of low-level constraints that have to be satisfied for a user, in order to be able to use the “Phantom Desktop” haptic device. In that case, the environment to be tested is an office including a desk with the “Phantom Desktop” on it and the user being sat on a chair. In order to use the haptic device, the user should satisfy the following constraints:

- is able to extend the elbow: $angle \geq \varphi$, and
- is able to rotate the wrist: $angle \geq \theta$, and
- is able to reach an object at position X: $arm_length \geq d$

If the before mentioned constraints are satisfied, the result of the simulation process will be SUCCESS. Otherwise, the result will be FAILURE.

CONCLUSION AND FUTURE WORK

The present paper, a framework that performs accessibility testing in virtual environments was presented. A new user modeling technique using UsiXML that enables the description of user's disabilities as well as the affected tasks was introduced. The great opportunity derived by such a framework is the automatic accessibility testing of any environment for any user by testing its equivalent virtual environment for a virtual user.

Future work will contain the creation of a large set of Abstract User Models describing as many disabilities as pos-

sible based on the literature. Furthermore, an extension of UsiXML that will enable a more detailed description of the user will be proposed to the UsiXML consortium. Additionally, the simulation platform will be extended with visualization features and the simulation process will be extended in order to be able to test dynamic environments (including moving objects that will interact with the virtual user, etc).

ACKNOWLEDGMENTS

This work is supported by the EU funded project VERITAS (FP7 - 247765).

REFERENCES

1. Beard, D. V., Smith, D. K., & Denelsbeck, K. M. (1996). QGOMS: A direct-manipulation tool for simple GOMS models. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI '96)* (Vol. 2; pp. 25–26). New York: ACM Press.
2. Chedmail P.; Maille B.; Ramstein E. “Accessibility and ergonomics study with virtual reality, a state of the art”, *Mecanique & Industries*, Volume 3, Number 2, 2002 , pp. 147-152
3. Eriksson, J. Johansson G.I., Akselsson, K.R., “A planning tool prototype for environment adaptations”, *IEEE Transactions on rehabilitation engineering*, pp. 283 – 287, 1995.
4. Gerrit, C. van der Veer, Lenting, B.F., Bergevoet, B.A.J., *GTA: Groupware Task Analysis – Modeling Complexity*. *Acta Psychologica* 91 (1996) 297–322
5. Gill, S.A., Ruddle, R.A., “Using virtual humans to solve real ergonomic design problems”, *International Conference on Simulation*, pp. 223-229, 1998.
6. Guerrero Garcia, J., Vanderdonckt, J., and Gonzalez Calleros, J.M. Towards a Multi-User Interaction Meta-Model. IAG Working Paper 08/28, Université catholique de Louvain, Louvain-la-Neuve, 2008.
7. Heckmann, D.: Introducing situational statements as an integrating data structure for user modeling, context-awareness and resource-adaptive computing. In: *ABIS2003*, Karlsruhe, Germany (2003) 283–286
8. Heckmann, D., *Ubiquitous User Modelling*, Akademische Verlagsgesellschaft Aka GmbH, Berlin, ISBN ISBN 3-89838-297-4 and ISBN 1-58603-608-4, 2006.
9. Heckmann, D., Schwartz, T., Brandherm, B., Schmitz, M., and von Wilamowitz-Moellendorff, M.: GUMO – The General User Model Ontology. In *Proceedings of the 10th International Conference on User Modelling*, Edinburgh, UK, LNAI 3538: Springer, Berlin Heidelberg (2005) 428–432
10. Jameson, A. Modelling both the context and the user, *Personal Technologies*, 5(1):29–33 (2001)

11. John, R. Wilson, "Virtual environments applications and applied ergonomics", *Applied Ergonomics*, pp. 3-9, 1999.
12. Kaklanis, N., Tzovaras, D. and Moustakas, K., Haptic Navigation in the World Wide Web", *Proc. of HCI International*, LNCS 5616, pp. 707-715, July 2009, San Diego
13. Kay, J. The UM toolkit for reusable, long term user models. *User Modelling and User-Adapted Interaction*, 4(3), p.149-196, 1995
14. Kobsa, A. Generic user modelling systems. *User Modelling and User-Adapted Interaction Journal*, 11(1-2):49-63., 2001.
15. Laughery, R., "Computer simulation as tool for studying human-centered systems", In *Proceedings of the 30th Conference on Winter Simulation*, pp. 61-66, 1998.
16. Mahfoudhi, A., Abed, M., & Tabary, D. (2001). From the formal specifications of user tasks to the automatic generation of the HCI specifications. In A. Blandford, J. Vanderdonckt, & P. Gray, (Eds.), *People and computers XV* (pp. 331-347). London: Springer.
17. Nayar, N., "Deneb/ERGO-a simulation based human factors tool", In *Proceeding of the Winter Simulation Conference*, pp. 427-431, 1995.
18. Palmon, O., Oxman, R., Shahar, M. and Weiss, P. L., "Virtual environments as an aid to the design and evaluation of home and work settings for people with physical disabilities" *Proc. 5th Intl Conf. Disability, Virtual Reality & Assoc. Tech.*, Oxford, UK, 2004
19. Razmerita, L., Angehrn A., Maedche, A., Ontology-based user modeling for Knowledge Management Systems, in *Proceedings of "UM2003 User Modeling: Proceedings of the Ninth International Conference"*, USA, 2003
20. Weiser, M. (1991) The Computer for the 21st Century. *Scientific American*. 265, pp. 94-10.

3D User Interfaces for Information Systems Based on UsiXML

Juan Manuel González-Calleros, Jean Vanderdonckt

Université catholique de Louvain

Place des Doyens 1, 1348

Louvain-la-Neuve, BELGIUM

+32 (0)1 047 8349

{juan.m.gonzalez, jean.vanderdonckt}@uclouvain.be

ABSTRACT

Since many years, 3D interactive systems have demonstrated some benefits in reproducing adequately the reality, in improving it, and even in augmenting it by providing the user with unprecedented actions. 3D User Interfaces are becoming the primary subject of interest of a growing community of researchers and developers adopting different approaches for specifying and creating 3DUIs. Providing development methods and software support for 3DUIs is a complex problem. In this paper, we argue that developing 3DUIs for Information Systems is an activity that would benefit from the application of a model-driven development methodology composed of: a set of models defined according to an ontology, a language that expresses these models, and a structured method manipulating these models.

CATEGORIES AND SUBJECT DESCRIPTORS

D2.2 [Software Engineering]: Design Tools and Techniques – Modules and interfaces; user interfaces. D2.m [Software Engineering]: Miscellaneous – Rapid Prototyping; reusable software. H.1.2 [Information Systems]: Models and Principles – User/Machine Systems. H5.2 [Information interfaces and presentation]: User Interfaces – Prototyping; user-centered design; user interface management systems (UIMS).

General Terms

Design, Experimentation, Human Factors, Verification.

Author Keywords

User Interface, 3D User Interface, 3DUI, Model-Based User Interface Development, Information Systems design.

INTRODUCTION

In HCI, a *Graphical User Interface* (GUI) is often understood as a User Interface (UI) that involves graphical widgets that are displayed as planar regions in xy planes according to their abscissa and ordinates. As such, they are considered as *two-dimensional UIs* (2DUIs). Many desktop environments support overlapping of widgets, pseudo-relief effects (such as shadows) and depth effects, thus raising the level up to *two-dimensional and half user interfaces* (2D½UIs). In contrast, *three-dimensional user interfaces* (3DUIs) involve graphical widgets that are rendered as volumes in xyz spaces according to their 3D coordinates.

2DUIs, as well as 3DUIs, could be used to support the user to achieve a 2D or a 3D interactive task. A 2D task in na-

ture, e.g., navigating on a map, can be supported by a 2DUI and does not necessarily require a 3DUI. A 3D task in nature, e.g., controlling a satellite in space, can be supported by both 2DUI and 3DUI, but not with the same quality. Table 1 categorizes UI types according to two axes: the nature of the task (2D vs. 3D) and the associated front-end (2D vs. 3D). For instance, navigating on a map, a 2D task in nature, can be rendered as a GUI on a 2D desktop, but also as a flat object in a 3D environment, although this is not particularly interesting. Organising tasks in windows, a 2D task in nature, is typically achieved in a 2DUI, but also benefit from a 3DUI (Figure 1). Controlling a satellite in space, a 3D task in nature, would really benefit from a 3DUI, although it could be projected into 2D planes as a GUI in 2D. In most situations, 3DUIs have shown some benefits with respect to 2DUIs, but also some shortcomings. 3DUIs are not automatically superior or inferior to 2DUIs. Moreover, some transitions may become desirable from 2D to 3D and vice versa in order to ensure appropriate representation of a change of context and reasons exist why maintaining 2D contents in 3D [10].

Front-end	Nature of the task	
	2D	3D
2D	2DUI, such as a GUI	GUI in 2D
3D	GUI in 3D	3DUI

Table 1. Nature of the task and its front-end.

Some reasons to care about 3DUIs are the following: some users, not all, prefer the use of 3DUIs [6] although they are helpful for specific tasks, but not all [40]; human perceptual mechanisms to analyze the world into structures of 3D primitives are better compared to 2D representations [5]; human visual bandwidth is much larger in 3DUIs than in 2DUIs [39]; users tend to remember better objects shapes and location in 3DUIs than in 2DUIs [38]; companies venturing into virtual worlds virtual worlds, such as IBM for hosting virtual meetings, has been rewarded with an increment in collaboration [28].

Thus, 3DUIs are becoming the primary subject of interest of a growing community of researchers and developers [3,27] adopting different approaches for specifying and creating 3DUIs. Providing development methods and software support for 3DUIs is a complex problem. Researchers are at a stage where they are developing new interaction

techniques, gestures and metaphors for 3DUIs [33]. Most research and development is focusing on technological issues, as reported in a survey of major publications on 3DUIs [43]. The research is mainly focusing on how to overcome hardware and software issues [11]. Little or no attention is devoted to the design knowledge that should drive the development life cycle of 3DUIs.



Figure 1. TaskGallery, an example of a 2D task in 3D [38].

In this paper we argue that developing 3DUIs for Information Systems is an activity that would benefit from the application of a development method which is typically composed of: (1) a set of models defined according to an ontology, (2) a language that expresses these models, and (3) a principle-based approach manipulating these models based on principles.

STATE OF THE ART

3DUIs can be the result of different development approaches: programmatic, toolkit based, method.

In the *programmatic approach*, the 3DUI is obtained by directly coding in its target computer language, e.g., C++. The programmatic approach is frequently preferred for applications where performance is a priority such as in games. Most games are written in C++ although, some may use C to try to get even more speed (at the cost of not having built in Object Oriented support). There are also some other application programming interface (API) that can be used in most common programming languages, for instance, Google O3D, Microsoft XNA.

A 3DUI can be also coded as a XML-based language, examples are: X3D, VRXML [7,9]. Programming and maintaining 3DUIs without any method could be not as simple as it gives no guarantee for regularity. There are no evaluation criteria to consider unless the final result is achieved. The tendency is on the “rush to code” approach without any structure favours a “trial and error” method.

A *toolkit approach* allows a straightforward implementation of a final interface once modelled in a tool. Using a predefined set of objects that help developers in their pro-

gramming task, the toolkit approach offers the best solution to draw 3DUIs elements with an immediate feedback. A lot of toolkits exists for 3D modelling, open source (Google SketchUp, Vivaty Studio), and commercial (Autodesk, Anark). While they are very similar in their basic capabilities, there are some differences more related to the interoperability with other technologies. For instance Autodesk Maya or 3ds Max can import 2D diagrams from AutoCAD and making the process of creating 3D content very simple. A toolkit is always useful and it was for us for designing our 3D objects. However does not provide design knowledge for developing 3DUIs.

The *methodological approach* relies on the step-wise frameworks for developing 3DUIs. There is a plethora of methods VR-Wise [35], CoGenIVE [9], InTML [13], Contigra [8], Tres-D [31], Desktop 3DUIs [25,26,42], Participative [34], Task analysis [14,37] to develop 3DUIs, we might have skipped some significant work in this area but we are not trying to be exhaustive but just reviewed some significant exiting work. Although these different methods share some similarities on their steps, several conceptual dissimilarities differentiate them.

Consequently, it is challenging to transfer one abstraction from one method to another in order to have one consistent framework supporting the development lifecycle of 3DUIs. They decompose the software life cycle into steps and sub-steps, but these methods rarely provide the design knowledge that should be typically used for achieving each step. Most approaches cover aspects such as task modelling, dialog modelling and implementation aspects. While some knowledge in explicit of existing method most of the time it is hard to find the abstracted models, the transformational approach along with its rules.

To identify shortcomings on existing work, a series of comparative analyses were conducted using the three axes recommended in [1]: descriptive part: a common ground is needed to describe every piece of work (for this purpose we use the Cameleon reference framework); comparative part: a set of criteria were defined to compare the different works that we described using a common syntax (next paragraph describe such criteria); generative part: new work emerges from the comparative analysis as a result of the identification of limitations or potentiality of the literature review (the proposed methodology is the result of this process).

The properties analyzed in the comparison were: models manipulated by the toolkit, all of them compliant with UsiXML models with the following notation: Di = dialog, AUI=abstract presentation, CUI=concrete user interface, FUI = Final User Interface, U = user, C = context. Along with this models to support a transformational approach then a series of inter model transformation is needed. When such transformational approach exists then it is explicitly denoted using the following notation: **(A, ..., B)** indicates that A, ..., B are grouped models that are done at the same

level; $A \leftrightarrow B$ indicates that A derives B and B is reengineered from A; $A \rightarrow B$ indicates that A derives B; $A \approx \leftrightarrow B$ indicates that model A concepts could be manually linked to model B concepts and that B can be manually reengineered to A; $A \approx \rightarrow B$ indicates that model A concepts could be manually linked to model concepts B. This means that the rules and the models exist but not a tool to support the automatic transformation.

Methods	Models	Inter Model Transformation
VR-Wise	CUI	CUI $\rightarrow$ FUI
CoGenIVE	T, Di, CUI	(T, Di, CUI) $\leftrightarrow$ FUI
InTML	Di, CUI	T $\approx \rightarrow$ (Di, CUI), (Di, CUI) $\approx \rightarrow$ FUI
Contigra	CUI, Di	(CUI, Di) $\rightarrow$ FUI
Tres-D	T, U, C, Di, Do	T, Do $\approx \rightarrow$ CUI, CUI $\approx \rightarrow$ FUI
Desktop 3DUIs	T, CUI, Di	T $\approx \rightarrow$ CUI, CUI $\rightarrow$ FUI
Participative	T, CUI, Di	T $\approx \rightarrow$ CUI, CUI $\approx \rightarrow$ FUI
Task analysis	T, CUI, Di	T $\approx \rightarrow$ CUI

Table 8 Comparison of Model-Based methodologies

3D USER INTERFACES FOR INFORMATION SYSTEMS

Similarly to a 2DUI, a 3DUI can be decomposed into two parts: the presentation part (also called *front-end*) and the semantic core part (also called *back-end*) equipped with the semantic functions, the system data storage, and the communication layer. 3DUIs are often associated to various 3D systems, such as those in virtual reality, mixed reality, augmented reality, and 3D desktop environments [29]. We focus of the 3DUIs we propose is Information systems (IS) for desktop-based systems. An IS for an organization is a construction made up of four blocks [2]: (1) data, a partial representation of facts that interest the organization; (2) processes, that represent means to acquire, search, store, present, and convey information; (3) organizational rules, governing the implementation of informational treatments; and (4) human & Technical Resources, required for the functioning of IS.

An IS supports management tasks according two axes [2]: functioning level, which ranges from operational, decisional to strategic; and the structure level, which ranges from structured to informal. We primarily consider management tasks that are operational and structured activities typically corresponding to administrative tasks, which are defined to deal with routine activities [2]. In this scenario, a context of use is assumed to be quasi-constant: the physical environment is assumed to be an office setup; the user has known skills required conducting these administrative tasks, and a

desktop computer is considered as the main computing platform. Therefore, we consider that 3DUIs that correspond to other tasks than such administrative tasks are beyond the scope of this thesis, as well as contexts of use that significantly depart from this assumption. For instance, a mobile traveler could conduct administrative tasks, but the resulting context no longer satisfies our assumption. And so do 3D games, volumetric displays, Organic UIs.

For administrative tasks supported by IS several interaction styles are candidates [45]: form filling, multi-windowing, direct manipulation, iconic interaction, graphic interaction, multimedia interaction, and 3DUIs. 3DUIs were chosen as a potential interaction style for IS since this option is underexplored. IS examples that are not covered by the present work includes: non interactive contents (e.g., a surgery room), information visualization (e.g., 3D statistics), and custom 3D contents (e.g., a stadium).

MODEL-BASED DEVELOPMENT OF 3DUIs

Models

In this section we define concepts related to 3DUIs, i.e., the representation of 3D widgets, their characteristics, and how from a user task we can derive the 3DUI. These models were incorporated to UsiXML to support:

- 3D Rendering of 2D User Interfaces. The UsiXML model that considers 2D GUI was specialized to consider the attributes specific for their 3D rendering. The extension specialize the Concrete Interaction Objects (*CIOs*) by adding extra attributes to define their size (height, width) and position (top, left) when it applies.
- 3DUIs A second approach involves a true 3D presentation of the 3DUI. By true representation is meant going beyond an imitation of their 2D GUI counterpart. In that scenario it was not enough to rely on the existing model as concepts such as appearance, texture, shape, and behaviour were needed.
- Hapgets Haptically enhanced 3D widgets [23,24].

We create the 3DUI model that includes concepts to express haptic interaction and different representations for 3D UIs depicted in Figure 2.

Language

In order to be fully-MDA compliant, this works need a User Interface Description Language (UIDL). Some environments may includes models, and a transformational approach but do not have a genuine modeling language behind. It is not just because there is a XML language that a genuine modeling language may exist [46]. A genuine UIDL must be strongly defined based on a trilogy (semantics, syntax, stylistics), this is the case of UsiXML. A review of the literature of existing UIDLs was conducted in [15] to justify the selection of UsiXML as UIDL for our work.

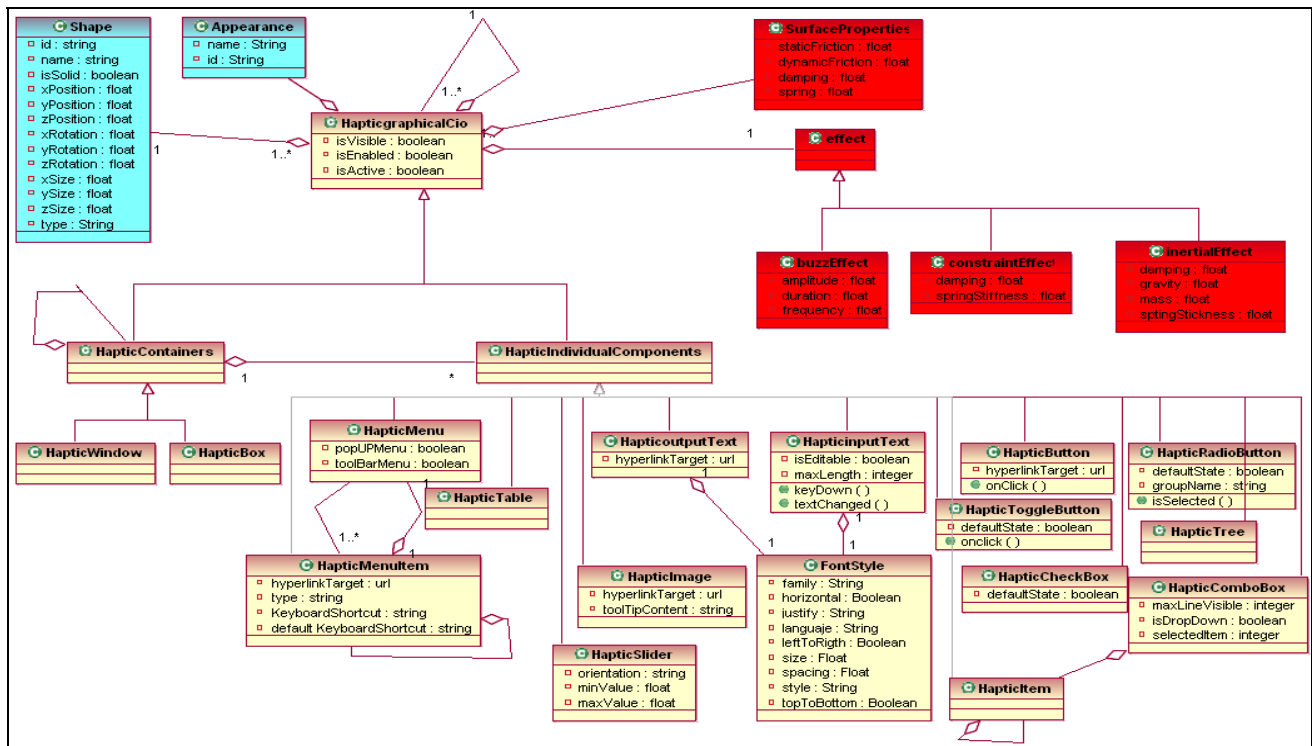


Figure 2 UsiXML 3DUI Concrete User Interface Model.

Regarding the UIDL trilogy we briefly comment that: the semantics are expressed as Meta-Models using the UML class diagram notation; the syntax is abstract expressed as XML Schemas and concrete expressed as a XML file; finally, the stylistics are the graphical representation used to represent the different artifacts involved in the methodology.

Software Tools

The set of software tools required to support the development of 3DUIs includes: model editors to assist the designer in constructing the models; design critics providing a designer with quality assessment facilities, models capturing explicit properties of the artefact are an ideal representation to perform evaluation; implementation tools translate a specification into a representation that can be used by a compiler, an interpreter or an interface builder; and transformation tools provide support to the designer to edit, store and execute model transformation rules.

Finding the right tool is a trade-off between six main criteria [41]: partial support of the tool not supporting the whole development process, learning time, building time, communication with other subsystems, extensibility and modularity. Supporting the evolution and the reuse of software remains a challenge. Although, it is hard to keep a valance we select the software to be used based on extensibility and modularity.

The software modules that support the methodology are:

- IdealXML [32]. Task, domain and AUI models are designed sing this tool. The semantics and models is from UsiXML. The notation of the task model is based on the stylistics from CTT [35] while its operators on the semantics of LOTOS operators. The domain model uses as stylistics UML class diagrams. The AUI uses an innovative expression for gathering abstract concepts related to the AUI model. Transformations from task and AUI model are supported.
- Usability Adviser [44]. Usability guidelines were introduced for each development step. Usability guidelines can be evaluated automatically using the Usability Adviser this software determines the usability of any UI specified in UsiXML.
- 3D Modelling was possible by relying on non-commercial tool: Alice (www.alice.org), Blender (www.blender.org), VUIToolkit [31] and Vivaty (<http://developer.vivaty.com/>).
- Software Support for Transformations. An analysis was reported for the transformation engines [18] used to support the transformational approach of our methodology.
- Haptic support was tested via the haptic browser, which was reported in [23,24].

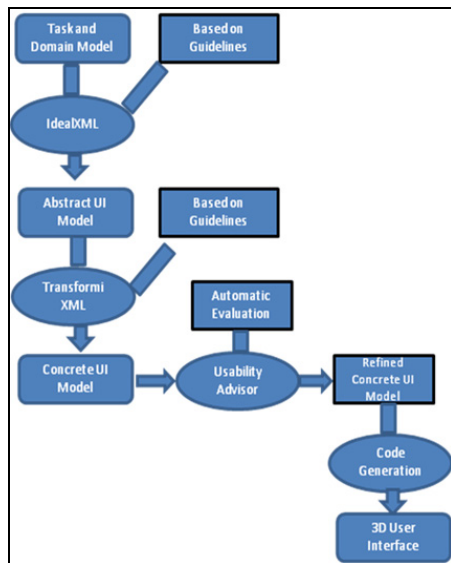


Figure 3. Outline of the method for developing 3DUIs.

METHOD AND CASE STUDY

The goal is not to come up with yet another Software Development Method but to reuse existing work and structure it accordingly. The result is a method that structures the development life cycle of a 3DUI of an IS in a principle-based way. The method follows an exploratory approach as its goal is to show a variety of possibilities to encourage design. It is said that is structured as it is based on a structured Framework, the Cameleon Framework [4]. Second, a set of instruments (evaluation guidelines [19,21], task patterns [21], canonical task types [20], rules for model to model transformations [17,18]) guide the use of the method then making it principle-based. The method (Figure 3) includes an evaluation of each development step, which can be performed manually or automatically. In order to provide another possibility to evaluate a 3DUI, our method could be an option to cope with the budget problem that rises when costly user experiments including experts are needed [Bowm02]. We rely on principles expressed as guidelines for modelling the task and abstract user interface (AUI) model. They are applied manually and refine the models. At the concrete user interface (CUI) model direct evaluation over the code is performed by evaluating rules and conventions or recommendations. The refined model is then used for the code generation. The next subsections details the development steps. Due to space reasons the following case study will not include haptic interaction in the final result.

Step 1: Task and Domain Modeling

In order to provide some means to designers for task modelling we propose [21]: 1) to reuse existing successful solutions existing as task patterns and 2) to follow a set of guidelines that might lead to a consistent task model that later can be object of transformations.

The 3D Universal interaction tasks patterns described in [3] are expressed in UsiXML task model notation. It is

not the scope of this paper to go beyond this description since most issues related to pattern-based design is extensively addressed in the literature. Even that the use of pattern-based development life-cycle is contradictory to model-driven engineering, as patterns are poorly structured or in many different ways. There has been some works [15] showing the potential of modelling patterns for task models and to reuse successful solutions. This could be easily extended into more detailed pattern Markup languages such as PLML (Pattern Language Markup Language).

Guideline based evaluation of task modelling is based on principles reflecting broad knowledge about task modelling coming from the literature. For instance, promote the use of systematic mechanism for the selection of task attributes, such as the task types [20].

Task Model Case Study

In this example we refer to a simple navigation task in a virtual office, with two interactive objects: an interactive table and an interactive screen. The user can interact with a table, navigate through the room and interact with a screen. For the navigate room, the task model uses the identified task pattern travel and Wayfinding. The interaction with big Screen indicates as the user interacts with screen task refers to a turn on/off a screen than renders video or images.

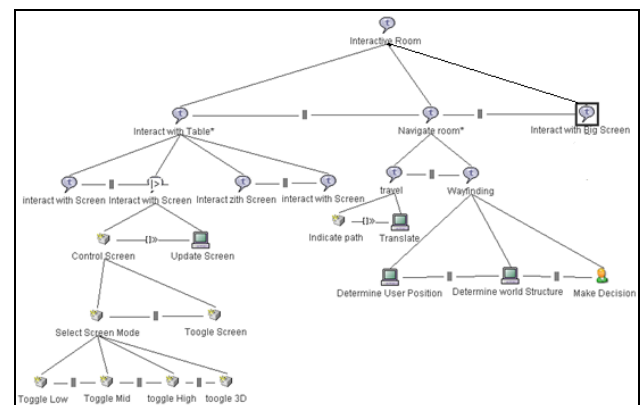


Figure 4. Virtual Office Task Model.

Step 2: Task and Domain Modeling

An AUI model can be generated automatically or produced manually from a task model following a set of heuristics. Various set of heuristics may fit this purpose depending on the type of AUI to be obtained: an AUI that reflects the task structure, an AUI minimizing navigation, an AUI compacting input/output. Most of them are compliant with the knowledge base from UsiXML. Although this level is independent of any modality, some guidance is still desired on how AUI might be structured considering further reifications into 3D CUI objects. Several metaphors have been introduced in order to display information or windows. If we imagine for instance a cube to render the different tasks as an Abstract Container (AC),

Question	Answer	Score	Options
What will be the representation of the 3D Toggle Button?	Switch	-	(1) 2D-3D Consistency
		--	(3) Easy to develop
		+	(4) Intuitional
		~	(5) Usability
		~	(1) 2D-3D Consistency
	Sphere	~	(3) Easy to develop
		~	(4) Intuitional
		~	(5) Usability
	2D representation	++	(1) 2D-3D Consistency
		+	(3) Easy to develop
		+	(4) Intuitional
	Haptic	+	(5) Usability
		+	(1) 2D-3D Consistency
		+	(3) Easy to develop
		+	(4) Intuitional
		+	(5) Usability

Figure 7. Questions and answer criteria to select a toggle button.

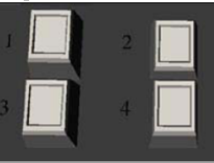
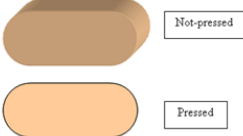
	Representation
2D Representation	
Switch	
Sphere	
Haptic	

Figure 8. Graphical representation for a toggle button.

In this particular example the decision was towards the use of a more 2D related representation, see options in Figure 8. The resulting specifications are obtained by realizing the above transformational development sub-steps. Figure 9, present a mock-up of the graphical UI corresponding to the CUI model.

Step 4: Final User Interface

The resulting CUI 3DUI can be encoded in a software tool, such as: Vivaty studio, Blender. The idea is not to start from scratch your models, this can be done but it is preferable to have a direct mapping in the tools and reuse existing work. Of course authors might produce their own objects if needed. The target languages in which we rely to concretize 3DUIs are: Java3D, VRML, X3D. The final rendering can be evaluated using any usability evaluation method. We are not aware of any system that performs usability evaluation directly on the code of a 3DUI (e.g., on VRML), although this could be a future avenue for automated evaluation. This is because at this stage, it is very complicated to analyze the code in a meaningful way. This is why we propose the use of automatic evaluation over the CUI model where semantics expressing the

3DUI can be object of evaluation [21]. For the final rendering guidelines can be followed before generating the code. We rely on existing knowledge on guidelines and some proposed such as those related to the hapgets [24].

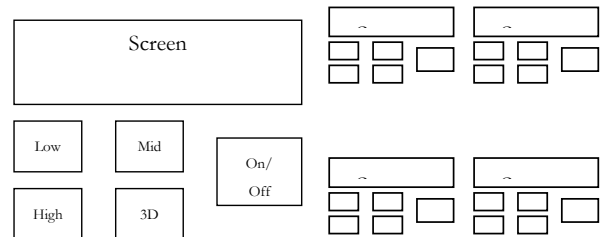


Figure 9. Mock-up of the Control Screen and Interactive table.



Figure 10. Virtual office with an interactive table.

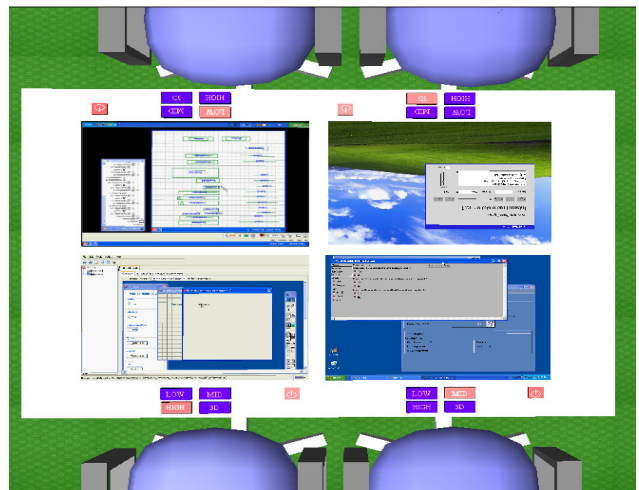


Figure 11. Top View of the virtual table.

FUI Rendering of the Case Study

The screenshot of Figure 10 shows the virtual office. Notice that the office follows the guideline “Virtual objects should be similar as much as the real objects” [25]. The Big screen, Figure 11 shows the interactive table composed of four control screens. The navigation task is controlled by the Cortona player plug-in., which support the navigation with the mouse and keyboard, as input devices, the user just decide where to go.

EVALUATION

Several cases studies have been developed around this methodology for different domain of application. Their goal was to serve as a proof of the concept of the different

principles introduced in the methodology, and to prove the feasibility of method through a set of case studies.

The first case study [17] was devoted to the development of an opinion polling system, a reasonably scaled example of a typical information system. A second case study [19,21] is dedicated to the development of an administrative application for a management school. Even that the problem complexity was moderate it has been chosen to illustrate the design diversity that offers 3DUI. A third case study [22] was devoted to the development of a flight navigation system for an aircraft. The complexity of this system is high but is more related to the algorithms that compute aircraft behaviour during the flight, while all this already exist. We conduct an in depth study to provide a MDA to the development of the Navigation Display UI and namely using a 3DUI instead the 2DUI. Finally, the fourth case study [23,24] was dedicated to the rendering of web site in the haptic web browser.

CONCLUSION

The development methodology relies on three main axes: models, method and language. The contribution to *UsiXML models* is summarized: more than 200 attributes, 90 classes, 100 relationships were added to the UsiXML models that corresponds mainly to aspects for 3DUIs and some more to task modelling concepts. A sanity check of the resulting models was made to consolidate them.

The *method* aspects adhere to the MDE paradigm. Models and transformations are explicitly defined and used, around 85 transformations rules (25 old, 15 adapted, 45 new). The method relies on the Cameleon reference framework then is said that is structured. It just provides means for forward engineering. A set of principles decomposed in: guidelines for the different development steps for evaluating the resulting models, task models relying on patterns and a canonical list of task type. All these principles promote systematicity when modelling 3DUIs.

The development steps reinforce existing knowledge on 3DUI development methods at different levels. In terms of transformational explicitness, abstraction layers independent of the modality of interaction. We provide some means to identify the diversity of concretizations of the 3DUI (i.e. its representation) for a task.

3DUIs are not intended for everybody. For the increasing number of users interested by 3DUIs, organizations need to be present in virtual online applications. For those users who are not interested by 3DUIs, our method benefits from using UsiXML and its MDA approach introduced in this paper that supports multi-path development, i.e., one source, many targets, an important benefit. These targets vary in computer platforms (e.g., mobile phone, desktop), programming languages (e.g., Java, Html). The goal of this work was not to prove that the method is better than another, neither the usability of the 3DUI produced. The goal was to define a method to develop 3DUI in a princi-

ple-based way as opposed to an opportunistic way. Therefore, the appropriateness of this method in this paper is for sure the major piece of work to investigate in the near future.

ACKNOWLEDGMENTS

We gratefully acknowledge the support of the ITEA2 Call 3 UsiXML project under reference #20080026.

REFERENCES

1. Beaudouin-Lafon, M., Instrumental Interaction: an Interaction Model for Designing Post-WIMP User Interfaces. *Proc. of ACM Human Factors in Computing Systems (CHI'2000)*, La Haye (The Netherlands), April 2000, ACM Press, CHI Letters 2(1), pp.446-453.
2. Bodart, F. Pigneur, Y., Conception assistée des systèmes d'information : méthode, modèles, outils, *Masson*, 1989.
3. Bowman, D.A., Kruijff, E., LaViola, J., Poupyrev, I., 3D User Interfaces: Theory and Practice, *Addison Wesley*, Boston, July 2004.
4. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J. A Unifying Reference Framework for Multi-Target User Interfaces. *Interacting with Computers* 15, 3 (June 2003) 289-308.
5. Carrol, J., HCI models, theories, and frameworks: toward a multidisciplinary science, *Morgan Kaufmann*, San Francisco, 2003.
6. Cockburn, A., McKenzie. 3D or not 3D? Evaluating the Effect of the Third Dimension in a Document Management System. In *Proc. of the ACM Conf. on Human factors in computing systems CHI'2001*, Seattle (USA), 31 March-5 April 2001, ACM Press, New York, 2001, pp. 434-441.
7. Cuppens, E., Raymaekers, Ch., Coninx, K., A Model-Based Design Process for Interactive Virtual Environments, *Proc. of Int. Workshop on Design, Specification, and Verification of Interactive Systems (DSVIS'2005)*, Newcastle upon Tyne (England), 13-15 July 2005, Lecture Notes in Computer Science, Vol. 3941, Springer, Berlin, 2005, pp. 225-236.
8. Dachsel, R., Hinz, M., Meißner, K. CONTIGRA: An XML-Based Architecture for Component-Oriented 3D Applications, *Proc. of 7th Int. Conf. on 3D Web Technology (Web3D'2002)*, Tempe (USA), 24-28 February 2002, ACM Press, New York, 2002, pp. 155-163.
9. De Boeck, J., Raymaekers, Ch., Coninx, K., A Tool Supporting Model Based User Interface Design in 3D Virtual Environments. In *Proc. of the 3rd Int. Conf. on Computer Graphics Theory and Applications GRAPP'2008* (Funchal, January 22-25, 2008). INSTICC - Institute for Systems and Technologies of Information, Control and Communication, 2008, pp. 367-375.

10. Deléglise, E., Paul, D., Fjeld, M., 2D/3D Web Transitions: Methods and Techniques, *Proc. of the 5th Int. Conf. on Web Information Systems and Technologies (WEBIST'2009)*, Lisbon (Portugal), 23-26 March 2009, INSTICC Press, 2009, pp. 294-298.
11. Dünser, A., Grasset, R., Seichter, H., Billingham, M., Applying HCI Principles in AR Systems Design, *Proc. of the 2nd Int. Workshop on Mixed Reality User Interfaces: Specification, Authoring, Adaptation (MRUI'07)*, Charlotte (USA), 11 March 2007.
12. Fencott, C., Isdale, J., Design Issues for Virtual Environments, *Proc. of Int. Workshop on Structured Design of Virtual Environments and 3D-Components at the Web3D 2001 Conference*, Paderborn (Germany), 2001.
13. Figueroa, P., Green, M., Hoover, H. J., InTml: A Description Language for VR Applications, *Proc. of 7th Int. Conf. on 3D Web Technology (Web3D'2002)*, Tempe (USA), 24-28 February 2002, ACM Press, New York, 2002, pp. 53-58.
14. Gabbard, J.L., Hix, D., Swan, J.E. II, User-Centered Design and Evaluation of Virtual Environments, *IEEE Computer Graphics and Applications*, 19 (6), November 1999, pp. 51-59.
15. Gaffar, A., Sinnig, D., Seffah, A., Forbrig, P., Modeling patterns for task models, *Proc. of 3rd Int. Workshop on Task Models and Diagrams for user interface design (TAMODIA'2004)*, Prague, 15-16 November 2004, Ph. Palanque, P. Slavik, M. Winckler (Eds.), ACM Press, New York, 2004, pp. 99-104.
16. Guerrero García, J., González Calleros, J.M., Vanderdonckt, J., and Muñoz Arteaga, J. A Theoretical Survey of User Interface Description Languages: Preliminary Results. In *Proc. of Joint 4th Latin American Conference on Human-Computer Interaction-7th Latin American Web Congress LA-Web/CLIHHC'2009* (Merida, November 9-11, 2009), E. Chavez, E. Furtado, A. Moran (Eds.), IEEE Computer Society Press, Los Alamitos, 2009, pp. 36-43.
17. Gonzalez-Calleros, J.M., Vanderdonckt, J., and Muñoz-Arteaga, J.M. A Method for Developing 3D User Interfaces of Information Systems. In *Proc. of 6th Int. Conf. on Computer-Aided Design of User Interfaces CADUI'2006* (Bucharest, 6-8 June 2006). Chapter 7, Springer-Verlag, Berlin, 2006, pp. 85-100.
18. González-Calleros, J.M., Stanciulescu, A., Vanderdonckt, J., Delacre, J.P., and Winckler, M. Comparative Analysis of Transformation Engines for User Interface Development. In *Proc. of 4th Int. Workshop on Model-Driven Web Engineering MDWE'2008* (Toulouse, 1 October 2008). N. Koch, G.-J. Houben, A. Vallecillo (Eds.), CEUR Workshop Proceedings, Vol. 389, 2008, pp. 16-30.
19. González-Calleros, J.M., Vanderdonckt, J., and Muñoz-Arteaga, J. A Structured Approach to Support 3D User Interface Development. In *Proc. of 2nd Int. Conf. on Advances in Computer-Human Interactions ACHI'2009* (Cancun, 1-6 February 2009). IEEE Computer Society Press, Los Alamitos, 2009, pp. 75-81.
20. González-Calleros, J.M., Guerrero-García, J., Vanderdonckt, J., and Muñoz-Arteaga, J. Towards Canonical Task Types for User Interface Design. In *Proc. of Joint 4th Latin American Conference on Human-Computer Interaction-7th Latin American Web Congress LA-Web/CLIHHC'2009* (Merida, 9-11 November 2009). E. Chavez, E. Furtado, A. Moran (Eds.), IEEE Computer Society Press, Los Alamitos, 2009, pp. 63-70.
21. Gonzalez-Calleros, J.M., Vanderdonckt, J., and Muñoz-Arteaga, J. A Structured Methodology for Developing 3D Web Applications. In T. Spiliotopoulos, P. Papadopoulou, D. Martakos, and G. Kouroupetroglou (eds.), "Integrating Usability Engineering for Designing the Web Experience: Methodologies and Principles", Chapter 2, IGI Global Inc., Hershey, 2010, pp. 15-43.
22. Gonzalez-Calleros, J.M. A Model-Driven Approach for Developing Three-Dimensional User Interfaces of Information Systems in a Principle-based Way, PhD Thesis, Université catholique de Louvain Press, Louvain-la-Neuve, February 2010.
23. Kaklanis, N., González-Calleros, J.M., Vanderdonckt, J., Tzovaras, D. Hapgets, Towards Haptically-enhanced widgets Based on a User Interface Description Language. In *Proc. of Workshop on Multimodal Interaction Through Haptic Feedback MITH'2008* (Naples, 31 May 2008).
24. Kaklanis, N., González-Calleros, J.M., Vanderdonckt, J., Tzovaras, D. Haptic Rendering Engine of Web Pages for Blind Users. In *Proc. of 9th Int. Conf. on Advanced Visual Interfaces AVI'2008* (Naples, 28-30 May 2008). ACM Press, New York, 2008, pp. 437-440.
25. Kaur, K., Designing virtual environments for usability, Ph. D. Thesis, City University, London, 1998.
26. Kim, G.J., Kang, K.C., Kim, H., Lee, J., Software Engineering of Virtual Worlds. In *Proc. of ACM Symposium on Virtual Reality Software and Technology VRST'98* (Taipei, November 1998). ACM Press, New York, 1998, pp. 131-139.
27. Latoschik, M.E., Reiners, D. Blach, R., Figueroa, P., Dachselt, R., *Proc. of Workshop of Software Engineering and Architectures for Realtime Interactive Systems SEARIS'2008* (Reno, March 2008). IEEE Computer Society Press, Los Alamitos, 2008.

28. Laudon, K.C., Laudon, J.P., Management Information Systems: Managing the Digital Firm. Pearson, 2009.
29. Laviola, J., Bringing VR and Spatial 3D Interaction to the Masses through Video Games. *IEEE Computer Graphics and Applications* 28, 5 (2008), pp. 10-15.
30. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., and Lopez, V. UsiXML: a Language Supporting Multi-Path Development of User Interfaces. In *Proc. of 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems EHCI-DSVIS'2004* (Hamburg, July 11-13, 2004). Lecture Notes in Computer Science, Vol. 3425, Springer-Verlag, Berlin, 2005, pp. 200-220.
31. Molina, J.P., A Structured Approach to the Development of 3D User Interfaces. Ph.D. thesis, University of Castilla-La Mancha, Albacete, Spain, 29 February 2008.
32. Montero, F., López-Jaquero, V., Vanderdonckt, J., Gonzalez, P., Lozano, M.D., and Limbourg, Q. Solving the Mapping Problem in User Interface Design by Seamless Integration in IdealXML. In *Proc. of 12th Int. Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'2005* (Newcastle upon Tyne, 13-15 July 2005). S.W. Gilroy, M.D. Harrison (Eds.), Lecture Notes in Computer Science, Vol. 3941, Springer-Verlag, Berlin, 2005, pp. 161-172.
33. Myers, B., Hudson, S. E., Pausch, R., Past, Present and Future of User Interface Software Tools, *ACM Transactions on Computer Human Interaction* 7, 1 (March 2000), pp. 3-28.
34. Neale, H., Nichols, S., Designing and developing Virtual Environments: methods and applications, *Proc. of Workshop on Visualization and Virtual Environments Community Club VVECC'2001*, Designing of Virtual Environments, 2001.
35. Paternò F., Mancini C., and Meniconi S. Concur-TaskTree: A diagrammatic notation for specifying task models. In *Proc. of IFIP TC 13 Int. Conf. on Human-Computer Interaction Interact'97* (Sydney, 14-18 July 1997), Kluwer Academic Publishers, Boston, 1997, pp. 362-369.
36. Pellens, B., Bille, W., De Troyer, O., Kleinermann, F.: VR-WISE: A Conceptual Modeling Approach for Virtual Environments, *Proc. of the Methods and Tools for Virtual Reality Workshop MeTo-VR'2005*, Gent (Belgium), 2005.
37. Polys, N.F., Hetherington, R., Brutzman, D., Gracacin, D., Engineering Virtual Environments with X3D, Tutorial at ACM Web3D 2005 Symposium, 10th Int. Conf. on 3D Web Technology, Bangor, March 2005.
38. Roberts, N., Andersen, D., Deal, R., Garet, M., Shaffer, W., Introduction to Computer Simulation, a system dynamics modeling approach, *Productivity Press*, 1983.
39. Shneiderman, B., 3D or Not 3D: When and Why Does it Work?, Human-Computer Interaction Laboratory & Department of Computer Science University of Maryland, Talk in Web3D, Phoenix (USA), 26 February 2002.
40. Shneiderman, B., Why Not Make Interfaces Better than 3D Reality, Virtualization Viewpoints. Theresa-Marie Rhyme (Ed.), November-December, 2003.
41. Shneiderman, B., Plaisant, C., Designing the User Interface. 4th Edition, *Addison Wesley*, Reading, 2004.
42. Smith, S., Duke, D., Design Support for Virtual Environments. *Proc. of Workshop: Design of Virtual Environments*, Oxford shire, 2001.
43. Swan, J.E., Gabbard, J.L. Survey of User-Based Experimentation in Augmented Reality? *Proc. of 1st Int. Conf. on Virtual Reality*, Las Vegas (USA), 2005.
44. Vanden Bossche, P., Développement d'un outil de critique d'interface intelligent : UsabilityAdviser, M.Sc. thesis, *Université catholique de Louvain*, Louvain-la-Neuve, 1 September 2006.
45. Vanderdonckt, J., A Small Knowledge-Based System for Selecting Interaction Styles, *Proc. of Int. Workshop on Tools for Working with Guidelines (TFWWG'2000)*, Biarritz (France), 7-8 October 2000, Springer-Verlag, London, 2000, pp. 247-262.
46. Vanderdonckt, J., Model-Driven Engineering of User Interfaces: Promises, Successes, and Failures, *Proc. of 5th Annual Romanian Conf. on Human-Computer Interaction (ROCHI'2008)*, Iasi (Romania), 18-19 September 2008, S. Buraga, I. Juvina (Eds.), Matrix ROM, Bucarest, 2008, pp. 1-10. ISSN 1843-4460

Implementation of Model-based User Interfaces for Pictorial Communication Systems using UsiXML and Flex

Antti Pitkänen

University of Oulu
The Department of Information
Processing Science
P.O. Box 3000
90014 Oulu
app@paju.oulu.fi

Seamus Hickey

University of Oulu
The Department of Information
Processing Science
P.O. Box 3000
90014 Oulu
seamus.hickey@oulu.fi

ABSTRACT

This paper introduces a process to combine a model-based user interface description language (UsiXML) with a Flex-driven rapid prototyping architecture for developing an Augmented and Alternative Communication (AAC) system based picture-driven user interface prototype. Multi-target user interface (UI) design, prototyping and development are difficult tasks and require lot of work when application is ported to a new device or an environment. Based on the work presented in this paper, Flex-based desktop and small-device RIA prototypes are constructed and used for prototyping AAC-based UIs in a rich web application environment.

Keywords

MB-UID, UsiXML, Flex, prototyping, RIA

INTRODUCTION

One of the main problems encountered when designing an interactive system is the creation of its user interface. Nowadays, UIs are becoming more complex, drastically increasing the effort and time spend in their development [10]. The typical industrial user interface engineering process still follows the traditional way of developing interfaces. User interface and interaction designers design the user interfaces and developers construct the working application based on these designs. The basic steps involved in this UI development process is describing the UI, prototyping it, code it with trials and errors, and repeat this process. UI builders can help to describe and perceive UIs, but these are only a partial solution for designing and implementing UIs.

In many cases when the UI is transported to a new platform or a device, significant re-design and coding effort is needed. Quite often, multi-target user interfaces should be adapted to the device's constraints, such as the screen resolution and preferred interaction techniques (e.g. text, graphical, voice-based, gesture) which requires the inclusion of the notion of plasticity in the development process [5, 7].

There is an increasing demand for more versatile methods for creating UIs for multiple different environments and devices. The motivation for this paper is to support the designing and the prototyping process of a multi-target (desk-

top, portable device) AAC-driven picture-based user interface prototypes.

Picture-based user interfaces uses a set of buttons with each button having a descriptive icon and one or two labels. The icons and labels used are based on Augmentative and Alternative Communication (ACC) methods. The main object of the AAC based UI is to function as a memory supporting device for supporting different tasks a user may want to accomplish, such as making coffee in kitchen or going out shopping and so on. In a coffee making scenario, the user doesn't necessarily have to know or remember the whole process of making the coffee, as the UI guides the user through the process when user presses different task related scenario buttons.

The Paper propose a method to combine UsiXML-language driven model-based user interface development (MB-UID) process with the Flex-based rapid prototyping and developing processes, for realizing the Pictorial Communication System (PCS) based UI (PCSUI), which is one example of an AAC method. The primary stress of the paper is to introduce an abstract way of combining UsiXML models with the Adobe Flash Catalyst-based high-fidelity prototyping process. Adobe Flash Builder 4 development environment is briefly introduced to describe the final step of the PCSUI development process for generating the final versions of the RIA-based (Rich Internet Application) [26] user interface prototypes.

BACKGROUND AND RELATED WORK

This section introduces the background and the related work of an XML-based RIA user interface prototyping and development which are of relevance to this paper.

The Development and the prototyping of multi-target user interfaces

The development of multi-target user interfaces requires the implementation of multiple versions of the same applications. With a model-based UI development approach, reusable, multi-target UIs can be designed and implemented more easily. MB-UID processes are based on the successive use of models describing the UI, with the integration of new information at each step of the developing process [29].

Designing the UI is a very complex task and the first design of the UI is rarely the definite version of it, and usually several iteration cycles are needed to reach the desired final result. Also different people involved in the development process have their own methods of expressing their ideas: some use drawings, sketches, pictures or screenshots of previously developed interfaces; some others instead, use paper prototypes or a simple computer-based prototype mock-ups [7]. It is quite complicated to accommodate all these different designs into one single design. In addition, it is even more difficult to convey UI design ideas to the developer that may misunderstand the designers goals, or developers may have their own views about how the UI should work.

Prototyping of the applications can be subdivided into low-fidelity, medium fidelity, high fidelity and mixed fidelity prototypes [24].

- (1) *Low-fidelity* (lo-fi) prototypes are usually a paper-based series of sketches or printouts, which does not allow realistic user interactions. Lo-fi prototypes are usually used early in the design process when trying to understand the user requirements and the expectations.
- (2) *Medium-fidelity prototypes* (me-fi) are refined versions of the low-fidelity prototypes and they are created on a computer, commonly with multimedia design tools, interface builders, or scripting languages.
- (3) *High-fidelity prototypes* (hi-fi) are refined versions of the medium-fidelity prototypes that typically have some level of functionality implemented, and may link to some sample data.
- (4) *Mixed-fidelity prototypes* have several different fidelities mixed in the same UI design. Beyond these previous fidelities, Coyette et al. [7] introduces the term multi-fidelity prototyping. It combines, within a single user interface prototype, several elements whose representations belong to the different levels of fidelity. In addition to these classifications, fidelities can be also divided roughly into paper-based prototypes, computer-based prototypes and to fully functional prototypes [17].

The majority of prototypes are developed using some type of support tool, Interface builder or multimedia design tool. Some of the tools support freehand sketching like DENIM [21], SUEDE [16], SketchiXML [27].

Most of today's prototyping tools are more focused on designing the look rather than the feel of the application. Many applications allows some prototyping of behaviours by giving the user a library of widgets to use, but the creativity of the user is limited by the selection of these predefined widgets, and customization of these widgets still requires some programming knowledge.

XML-based languages in user interface development

In recent years, User Interface Description Languages (UIDL) like UIML [3], XIML [25], XUL [37], UMLi [8], TeresaXML [31], UsiXML [33] and others UI markup languages like Openlazzlo [23], XAML [36] and MXML [6] has been successfully used for creating XML-based model-driven UIs. UIDLs specify the UI variations or a generic UI that is then further tailored for each individual platform or device. Specifications of the UI created with UID, can be reused and adapted according to constraints imposed by input/output devices, different contexts of use, or specific user preference [29].

UsiXML language

The Cameleon Reference Framework [29] based UsiXML (user interface eXtensible Markup Language) with TeresaXML is one of the few languages that supports four abstraction levels for the specification of the user interface (i.e., task & concept models, abstract UI, concrete UI and final UI) [18]. The user interface can be specified in terms of functionality, the object it manipulates and the graphic presentation of the user interface. Usually, the development work begins from the task model until reaching the final running program at the end, but the process can also be reversed or started from the other levels. Consequently, the abstract models can be produced from the more concrete models. Created models are reusable and multi-target versions of the UI can be produced from the models (e.g. different look & feel for a mobile systems) [18].

UsiXML has several tools for supporting the model-based implementation. With the IdealXML tool [15] the task-, the domain- and the abstract user interface (AUI) models can be created. GrafiXML [13] and SketchiXML [27] are used to produce the concrete user interface (CUI) of the user interface. TransformiXML [32] is used for mapping between the models and transform a UsiXML compliant specification into another UsiXML-model.

UsiXML models

A task model describes the various tasks to be carried out by a user when interacting with the system. Another concept level model, the domain model, consist of classes, attributes, methods and domain relationships that describes the various entities involved in that system and their relationships.

An Abstract user interface (AUI) model represents UI definition that is independent of any modality of interaction (such as graphical, vocal or tactile) and computing platform specificities [20]. An AUI is populated by Abstract Interaction Objects (AIO) and Abstract user interface relationships. AIOs may be of two types: Abstract Individual Components (AIC) and Abstract Containers (AC). An AIC may be further composed of multiple purpose of uses, called facets. Each facet describes a particular function an AIC may endorse in the physical world (input, output, control and navigation) [20]. A mapping model instead is used

for mapping the task-, the domain- and the AUI models together [20].

A Concrete user interface (CUI) model is a more concretized version of the AUI. A CUI model is composed of a Concrete Interaction Objects (CIO) (widget sets found on most graphical and vocal toolkits) and concrete relationships. Concrete interaction objects and relationships are further refined into graphical objects and relationships or auditory objects and relationships [20]. The Final User Interface (FUI) is instead the operational UI running on selected platform and environment.

RIA-environment and tools

Rich Internet Applications (RIAs) [26] have been seen as the next big evolutionary step for creating web-based applications. RIA-technologies combines two important characteristics, collaboration and interaction, and in this way change the way people can perceive, use and share the Web and software more generally [30]. Collaboration can be defined as the social aspect that allows a vast number of people to collaborate and share applications, services and data over the Web. Interaction is shown in RIA technologies by making it possible to build websites that behave and can be used pretty much the same way as desktop applications.

Adobe's Flash Catalyst and Flash Builder 4 for creating RIAs

Adobe Flash Builder 4 (beta) [11] and Flash Catalyst (beta)

[12] are new tools for creating Rich Internet Applications with the Flex-framework (MXML [6] & ActionScript 3.0 [1]). Flash Catalyst is an interaction design tool for rapidly creating the user interfaces without coding. It is used for creating the functional user experience from the UI-designs created in image processing programs (e.g. Adobe Photoshop or Adobe Illustrator). After this step Flash Catalyst project file (FXG) can be transported to the Flash Builder to add functionality and integrate it with needed servers and services.

PCSUI-application prototypes. The main language used to define UIs in these programs is MXML [6]. It is a XML-based user interface mark-up language and a part of Flex mark-up language. MXML is used to declaratively set up the user interface components and other aspects of the application and ActionScript provides the logic which handles user interactions with the application [6]. The Flex-framework itself enables developers to build and deploy applications that take advantage of the Flash platform [2].

Adobe Flash Catalyst enables a more creative approach for prototyping. Flash Catalyst can be considered as a hi-fi computer-based prototyping environment. When interconnecting developing process of the Flash Catalyst with the Flash Builder 4, fully functional prototypes can be produced in many cases with quite minimal coding knowledge.

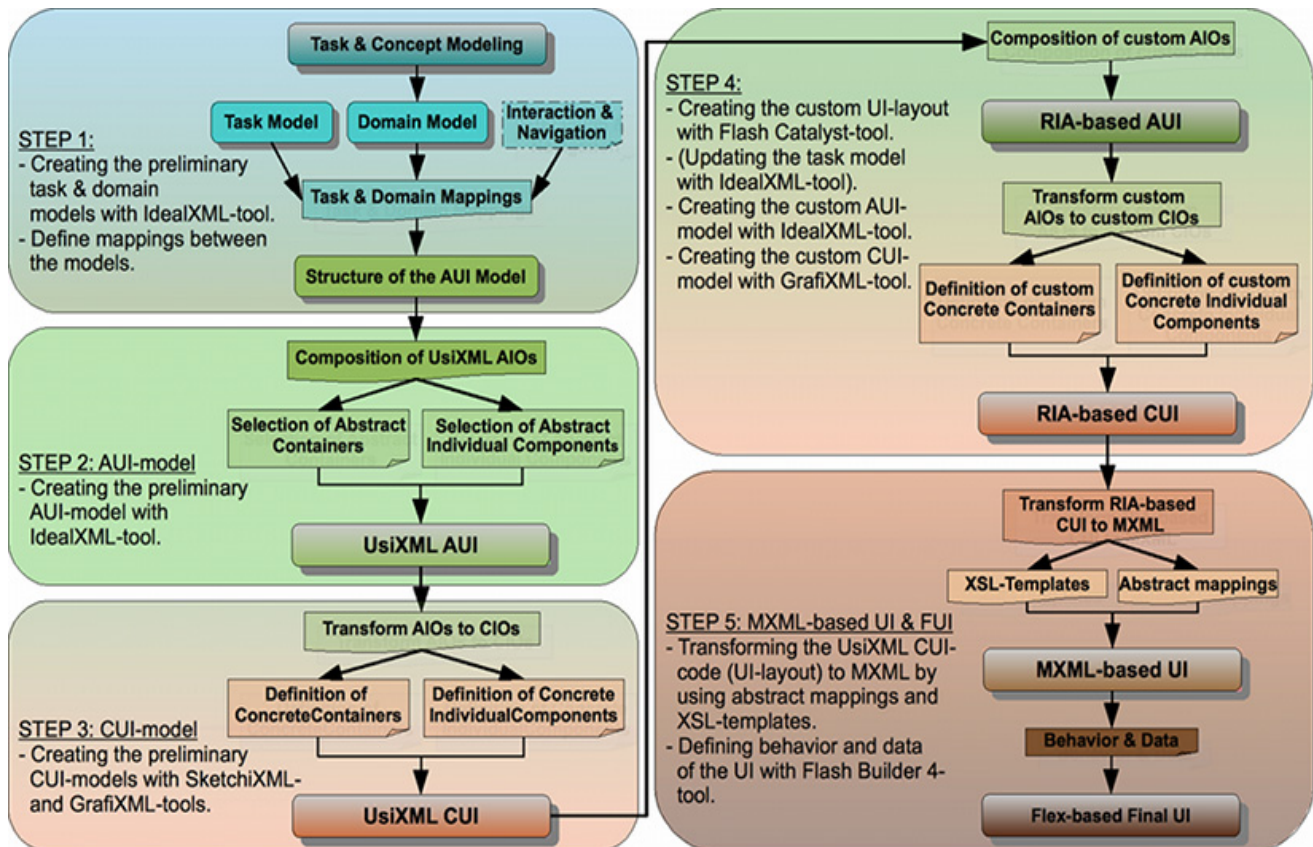


Figure 1. Workflow for creating the RIA-based

CASE STUDY: PCSUI

The Pictorial Communication System (PCS) based UI is created using model-driven Engineering (MDE) methods integrated with the RIA framework, in this case a Flex-language based architecture. The UsiXML-language is mainly used to implement the design phase models: the task model, the domain model, the mapping model and the abstract user interface (AUI). Based on these models, SketchiXML [27] and GrafiXML [13] are used for generating the simple CUI wireframe versions of the UI. Adobe Photoshop CS3 and Flash Catalyst is used for designing and creating the custom UI layout with the functional components, and Flash builder 4 is used for creating the final user interface with data and functionality. Figure 1 describes the overall process of the model-based user interface development linked with the Adobe's new design and the development environments. The implementation process starts from the UsiXML-concept models (the task and the domain), and ends by realizing two different versions of the prototype UIs (the desktop and the small device) running as Web applications.

A similar approach is represented by Martinez-Ruiz *et al.* [19]. They proposed a model-based RIA UI development method that refines the UI model from the abstract model (UsiXML) to the concrete model (XAML) using XSLT transformations (Extensible Stylesheet Language Transformations). Also Sorokin *et al.* [28] are proposing a framework for the model driven development of Adobe Flex RIAs which uses task models and international standard task-based metric and a development-oriented approach to integrate usability characterization and a evaluation comparison of different concrete user interface versions. This PCSUI work differs from the above mostly by concentrating directly on creating custom hi-fi prototypes of UIs by integrating new Adobe-tools (Flash Catalyst beta & Flash Builder 4 beta) and UsiXML.



Figure 2. An example of picture-based AAC-communication in use. This supports a process for making coffee.

Concept of the PCSUI and Implementation of prototypes

The user interfaces in this case study are based on the concepts introduced in [14] which describe Augmentative and Alternative Communication (AAC) [35] methods and an example of this UI is shown here (Figure 2).

Augmentative and Alternative Communication is mainly used as a method that supplements or replaces speech and writing as a communication method, and provides help and improves the communication process for persons who lack the ability to communicate due to some kind of disability [35]. The representation system used in AAC include gestures, hand signal, photographs, pictures, line drawings, words and letters, but also different ways of managing computer-aided communication tools are included. AAC may be unaided, that is, not requiring any external device for its use such as signing and gesture, or based on aided techniques, ranging from picture charts to the most sophisticated computer technology currently available [35].

There are a number of different pictorial systems to help people to communicate such as PCS, Blissymbols, Rebus and PIC [35]. Some of these symbol systems involve the use of gestures or signs, and other symbol systems are more pictorial or graphic. PCSUI is influenced by several of these AAC-driven symbol-based communication systems, but is not using particularly any of the previous system at this moment.

Picture symbols plays an important role in the daily life for normal people as well as those with disabilities. Actually, graphic or pictorial symbols are things that we see all around us everyday. Traffic- and restroom signs, including the symbols located in the airports are examples of everyday pictorial symbols. Some of the symbols are more obvious than others, but they all convey a visual message.

The PCSUI user interface concept is a button-based UI with a minimal degree of functionality so it would be easy to use by its target group, in this case, elderly people with memory disorders. The main elements are clickable buttons representing a particular task with an image and label(s), the non-clickable "buttons", the metadata textbox describing selected steps (tasks) in the UI and a few simple buttons for navigation (Quit, Menu, Next, Previous, Backward and Forward). Tasks are described in the user interface based on the XML-file tree structure. The root of the XML is hidden and describes the whole scenario of tasks concerning the specific location. The parent node(s) of the XML-file can be buttons or non-clickable elements of the user interface. They can have children node(s), they for one's part can have sub children node(s) and so on. In the prototypes developed so far, tasks are used more like post-it notes, that helps the user and works as a memory support. For example, when going out, the UI reminds the user to take their keys with him/her. However, in later versions of the UI, tasks should also trigger other events, for instance, the user can select tasks like calling someone.

From tasks to Concrete user interface

After the requirements of the conceived UI are gathered, the next step of the development process is to start working with UsiXML and create the preliminary task, the domain and the AUI models of the UI. Modelling can be started from the level that the developer prefers, but usually it is started from the task level. IdealXML-tool [15] is used for realizing these first step models. Figure 3 shows a preliminary desktop version task model of the PCSUI. A preliminary AUI-model and a simple domain model, shown in figure 4, is holding three main classes (main class PCSUI, and helping classes NodeTree and Node) of the UI. As a notice, the domain model described here is basically the final version with attributes and methods that are used for actualizing the final user interface versions of the application prototypes. Next, UsiXML CUI-tools are used for designing the simple version of the UI based on the AUI model gen-

erated in this step. The result of this step is UsiXML CUI-code and a simple wireframe version of the UI.

From CUI to final user interface

For generating the concrete user interfaces SketchiXML [27] and/or GrafiXML [13] tools can be used. SketchiXML is used for defining lo-fi sketches of UI prototypes, and producing independent interface specification code of UsiXML and UIML. UsiXML code can then be imported to GrafiXML-tool to generate hi-fi prototypes of the UI. At this moment GrafiXML-tool can automatically convert CUI-code to Java, XUL and XHTML. As custom graphical editor-based layouts and components (not the standard wireframe components) can be transported and brought to life with the Adobe-tools, UsiXML models and CUI-code have to be mapped to the MXML as it works as a basis for the elements created with Flash Catalyst or Flash Builder 4.

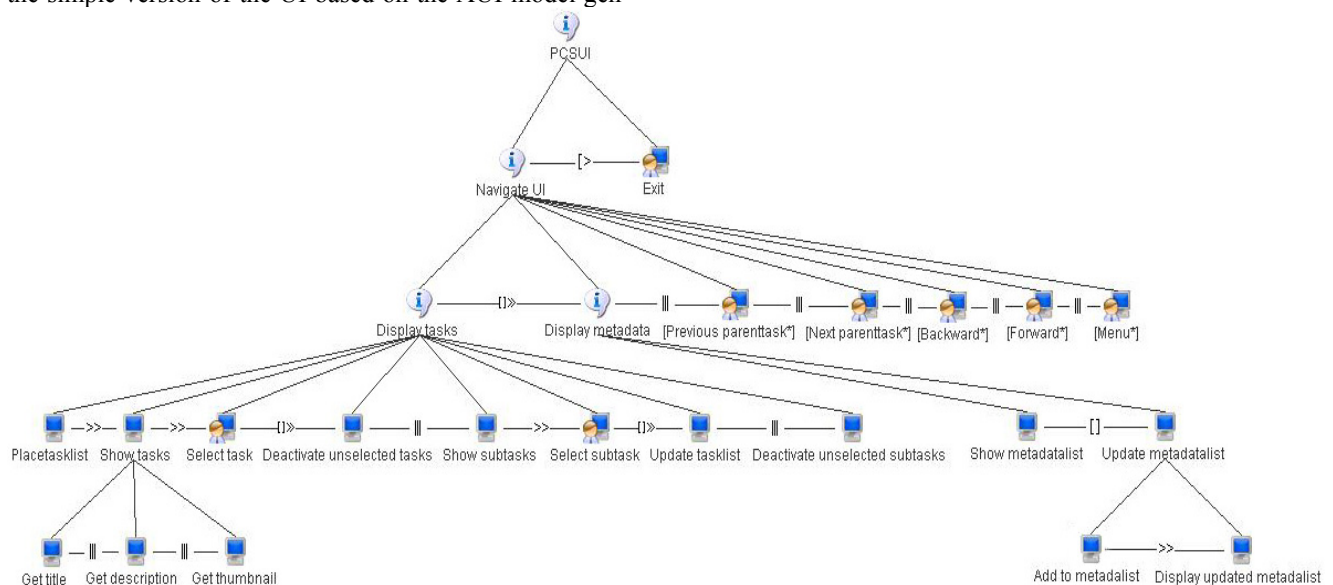


Figure 3. The preliminary task-model of the PCSUI.

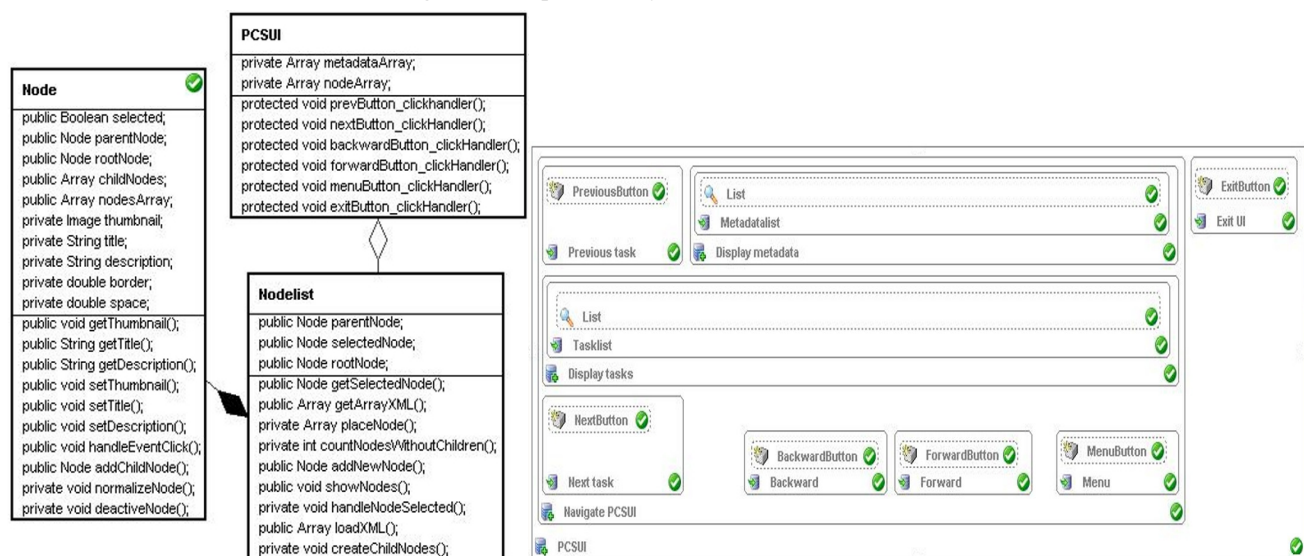


Figure 4. The domain model and the preliminary AUI-model of the PCSUI.

This is achieved by using abstract handmade mappings and XSLT-templates. Only the basic principles of the process will be demonstrated here. It has to be remembered, that the following process for connecting Flash Catalyst workflow with UsiXML workflow is still quite abstract and more complicated transformations (e.g., Graph Grammar transformations [9]) are not applied directly in this paper.

After the wireframe version of the CUI of the PCSUI application is created, work continues with Flash Catalyst and a simple MXML wireframe version of PCSUI is created based on the CUI model created earlier. MXML-code generated in this step is used to compare the code structure generated by Flash Catalyst to the code structure of the CUI generated by the GrafiXML-tool.



Figure 5. Design prototype of the PCSUI.

Also at this point simple prototyping is done with the Flash Catalyst adding the different states and the transitions to test out the UI structure and the functionality. After this, Flash Catalyst is mainly used for defining the basic layout structure of the UI (Figure 5). Because of the simplicity of the application structure, the whole UI can be defined by using only one single view, in which the whole interaction occurs. But the UI is lacking the typical RIA look-feel from it and doesn't really take advantage of the Flash Catalyst tool at its fullest. At this point, the graphic editor based design components are converted in Flash Catalyst to the interactive components (e.g., buttons, list) and containers for holding these components.

As a result of this previous phase, the MXML-code becomes more complicated and several new classes are automatically created for the basis of the custom MXML-components. For instance, the custom button has its own MXML-class, called Skin, which holds the data-elements like pictures and labels of the button. For this reason AUI and CUI models have to be rebuilt to correspond to the more complicated structure of the redesigned PCSUI application.

UsiXML AUI Model	UsiXML CUI Model	FLEX 4 component (Layout)
<u>Abstract Container (AC)</u>	<u>window, (constraint)</u>	<u>Application, WindowedApplication, (DesignLayer)</u>
<u>Abstract Individual Component (AIC)</u>	<u>Box, groupBox...</u>	<u>DataGroup, Group, Hgroup, Vgroup, Panel, Form, Canvas</u>
Facet:		Controls:
control	A trigger (e.g., button, toggleButton)	Button, PopUpButton, PopUpMenuButton, ToggleButton, HSlider, Vslider, VscrollBar, HScrollBar
input	Input (e.g., inputText)	TextInput, TextArea
output	Output (e.g., outputText)	RichText, Label, tooltip, Image, ProgressBar
navigation	A trigger (e.g., navigation)	LinkButton
control+output	Button (trigger) + output	Button+Image/Label
control+input+output	DatePicker, hourPicker, filePicker colorPicker, tree...	List, Datefield, DateChooser, DataGrid, ColorPicker, Tree
output+input	inputText	RichEditableText, Editable TextArea
control+input	comboBox, radioButton, checkBox checkBox	ComboBox, RadioButton, CheckBox
....		
navigation+output/output	navigation+imageComponent/outputText	LinkButton+Image/Label

Figure 6. Possible mappings between the AUI and Flex 4.

In Figure 6, based on the work of [28], one possible solution for describing the UsiXML AUI-elements with the Flex 4 can be seen. By combining different types of facets, more complicated MXML-components can be defined.

Figure 7 shows the updated version of the task model, that has more defined structure inside Display tasktree-task (formerly known as tasklist) dividing it to the Parent- and Childtasks. This is done mainly for separating these two abstract tasks, since they are more or less handled as a individual separate tasks inside the UI. The task model also separates tasks that are available in the portable- or in the desktop-version of the PCSUI.

Figure 8 instead shows an updated version of the AUI-model, on which buttons are constructed from several facets. Model also indicates which elements are available in the portable- or in the desktop-version of the PCSUI. After the component structure is defined in a new AUI, the next step of the process is to investigate how the MXML components can be expressed in the UsiXML CUI model. Defining the connection between the concrete user interface and the MXML-based UI, Graph Grammar rules may be used to redefine the internal structure of the CUI, and XSLT to link CUI-side components to the MXML-components. Because MXML-language is basically an enhanced version of the XML-language, like UsiXML, simple mappings between these languages can be defined in quite straightforward way.

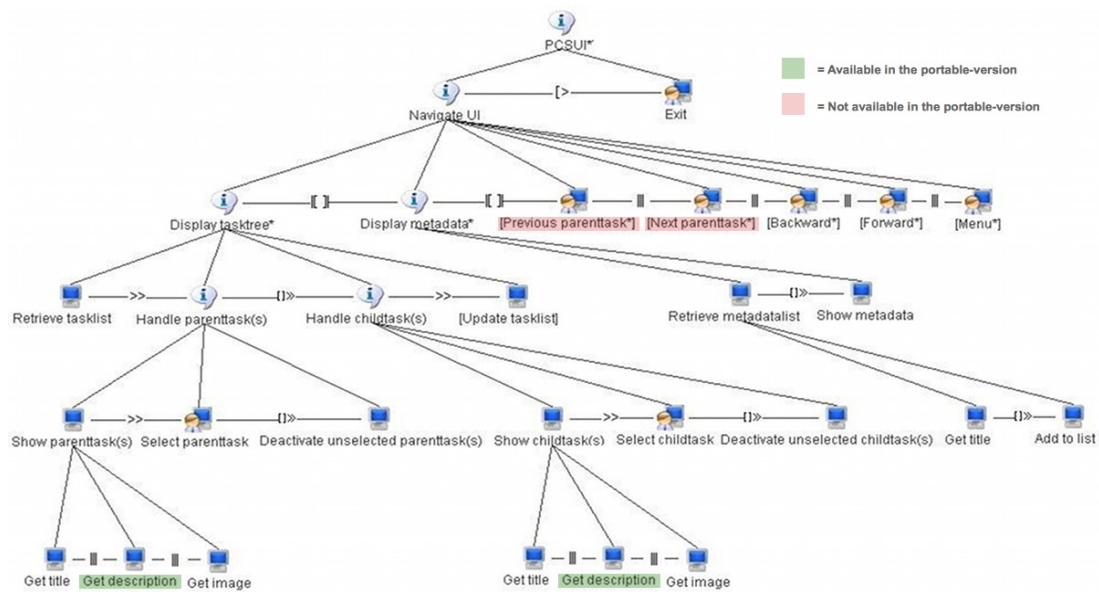


Figure 7. Updated task-model of the PCSUI.

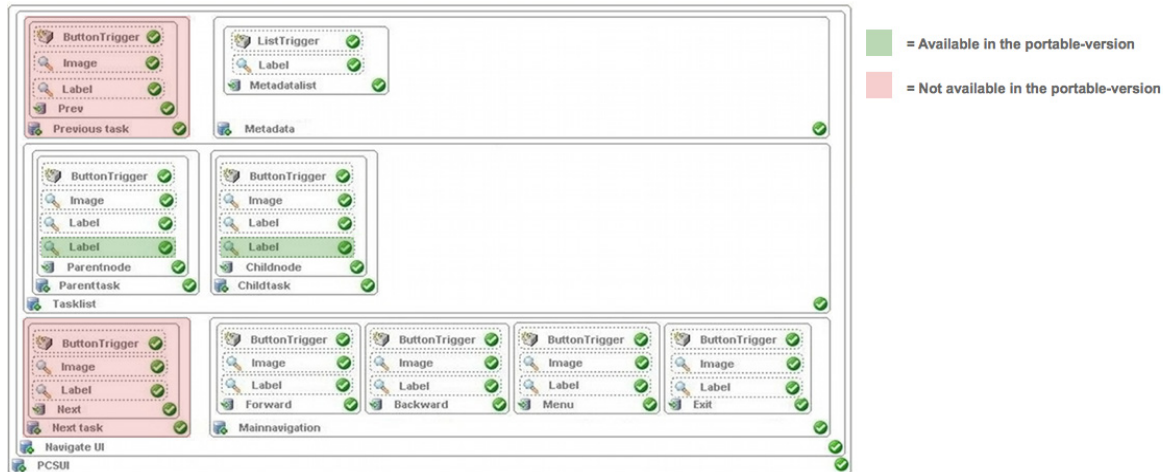


Figure 8. Updated AUI-model of the PCSUI.

Currently, the approach introduced here concentrates more on defining the abstract mappings between UsiXML-elements and the MXML-components, and defining the workflow between UsiXML and Flash Catalyst to create multi-target UI prototypes, so Graph transformation rules are not directly used. Generated mappings between languages can be as follows: MXML Window or WindowedApplication tag is mapped to the window element of the UsiXML CUI. Optional MXML DesignLayer-tag refers in this case to the constraint-tag of the UsiXML CUI. MXML Group-container refers the box (or groupBox) element of the CUI. Button refers to the button element at the CUI-level. RichText is the output_text_Component in a CUI model and BitmapImage instead refers to image_Component.

In the next step, simple XSL-templates are used to convert the CUI-based layout into MXML-language. Figure 9 demonstrates this transformation process by creating a custom MXML-button from a custom CUI button by using XSLT-transformations. For simplifying reasons, all the attributes defined in the CUI-code are not displayed on the MXML-code. Also most of the attribute values are set to default, and only the main XSL-template part is visible. In CUI code two different versions of describing buttons with multiple facets is defined. In first approach, Graph transformation-rules can be applied and the CUI-element button is reconstructed to hold other elements, like imageComponent and outputText. The second approach is to use XSLT-templates and rules to define more custom structure for MXML-button.

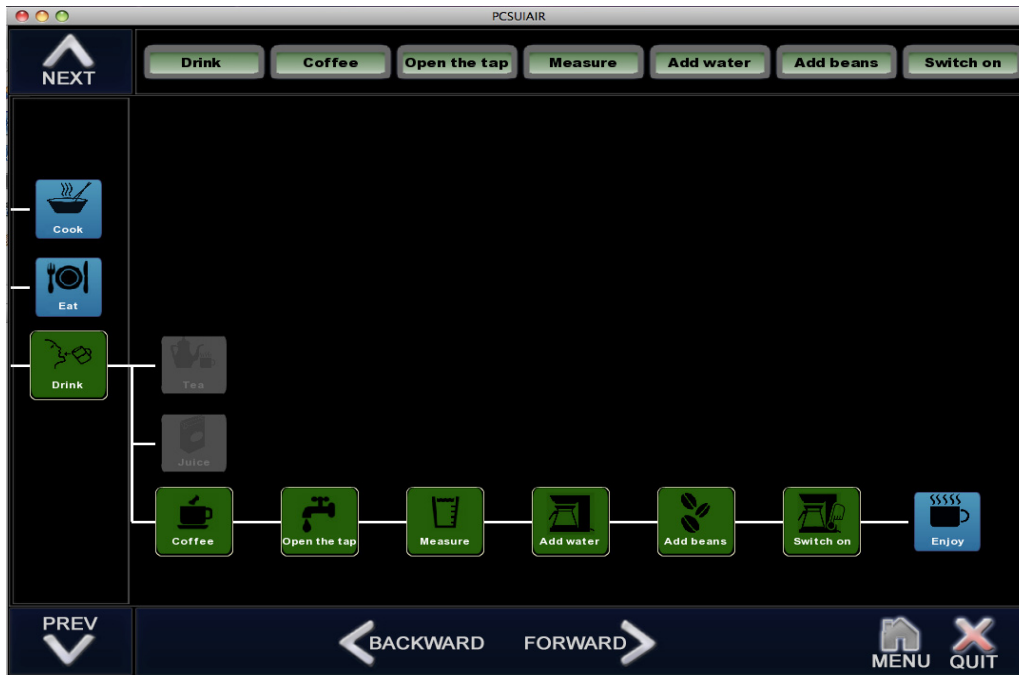


Figure 10. Desktop RIA-version of the PCSUI.

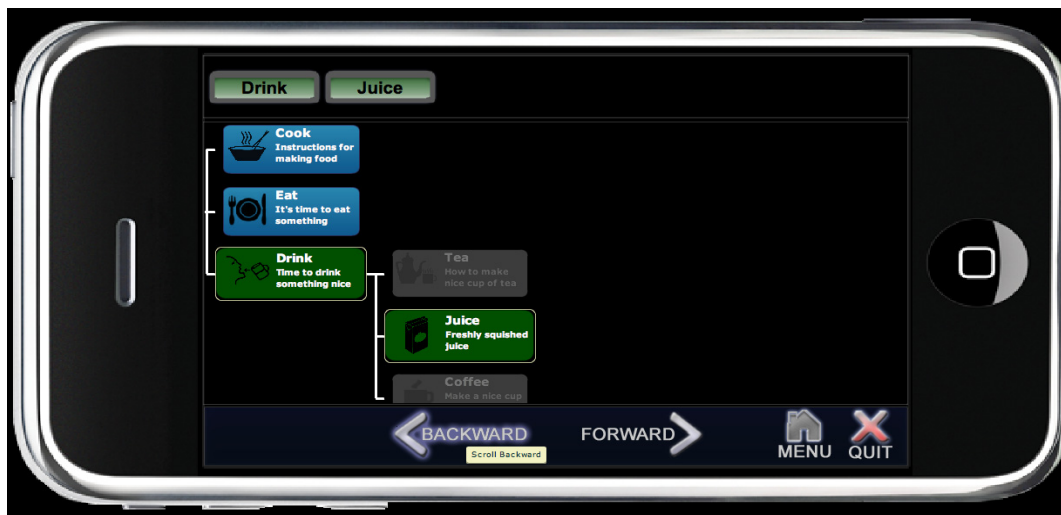


Figure 11. Portable RIA-version of the PCSUI.

The final step of the process is to produce the final user interfaces (FUIs) (figure 10 & 11). Adobe's Flash Builder 4 tool is used for actualizing this step, and with it the data and the functionality are added to the UI-frame generated in Flash Catalyst-tool. In this work the look of the UI is built with MXML and the functionality and data are added with ActionScript 3.0. Also in this phase, the UsiXML-based domain model of the PCSUI is used as a basis of the implementation process, when the final functionality of the application is added.

CONCLUSIONS AND FUTURE WORK

This paper introduced the model-based UI development combined with the Flex-driven prototyping and development environment for creating multi-target fully functional

prototypes. The paper proposed an abstract way of combining UsiXML models with the Flash Catalyst UI prototypes, enabling to continue the UI development process with Flash Builder 4-tool to create fully functional Web applications. Although the approach was successful for defining the connection between the UsiXML models and the Flash Catalyst side MXML-components, more work is needed to define the internal structure of MXML-components and how they can be presented by UsiXML and vice versa. Additional work is also needed to define a more fluent mapping process between these two frameworks, and how more complex UIs with behaviour included can be mapped and constructed.

UsiXML CUI-code for the custom button:

```
<button id="button_component_13"
name="button_component_13" icon="SourceOftheImage"
defaultToolTip="Task" content="/uiModel/resourceModel/cioRef[@cioid=button
_component_13]/resource/@content" defaultContent="Prev" isVisible="true"
isEnabled="true" glueHorizontal="left" glueVertical="bottom"
textColor="#000000">
<outputText id="output_text_component_0"
name="output_text_component_0" content="/uiModel/resourceModel/cioRef[@
cioid=output_text_component_0]/resource/@content" defaultContent="Title"
isVisible="true" isEnabled="true" fgColor="#ffffff" isBold="true" textSize="12"
textFont="Arial" textColor="#000000" textHorizontalAlign="right"/>
<imageComponent id="image_component_0"
name="image_component_0" icon="SourceOftheImage" imageWidth="40"
imageHeight="40" imageHorizontalSpace="4" imageVerticalSpace="4"
defaultToolTip="Task" content="/uiModel/resourceModel/cioRef[@cioid=image
_component_0]/resource/@content" defaultContent="Test" isVisible="true"
isEnabled="true" textColor="#000000"/>
</button>
```

Main XSL-template for creating the buttons:

```
<xsl:template match="button">
<xsl:if test="count(child:*) >= 1 or parent::box and count(./*) > 1">
<!--Custom flex button-->
<s:Button label="{@defaultContent}" id="{@id}" name="{@name}"
skinClass="component.{@name}" d:userLabel="{@defaultContent}"
toolTip="{@defaultToolTip}" enabled="{@isEnabled}" />
<!--Import custom-button skin from the component.xsl-template-->
<xsl:apply-imports/>
</s:Button>
</xsl:if>
<!--Else create firewrame-button-->
<xsl:if test="count(child:*) = 0 or parent::box and count(./*) = 1">
<s:Button label="{@defaultContent}" id="{@id}" name="{@name}"
d:userLabel="{@defaultContent}" toolTip="{@defaultToolTip}"
enabled="{@isEnabled}" />
</xsl:if>
</xsl:template>
```

The custom button MXML-code:

```
<s:Button label="Prev" id="button_component_13" name="button_component_13"
skinClass="component.button_component_13" d:userLabel="Prev" toolTip="Task"
enabled="true">
<s:Skin xmlns:graphics="assets.graphics.*">
<fx:Metadata {[HostComponent("spark.components.Button")]}</fx:Metadata>
<s:states>
<s:BitmapImage id="image_component_0" name="image_component_0"
d:userLabel="Test" source="SourceOftheImage" width="40" height="40" x="4"
y="4" toolTip="Task" visible="true" enabled="true"/>
<s:RichText id="output_text_component_0" name="output_text_component_0"
d:userLabel="Title" Text="Title" fontFamily="Arial" fontSize="12"
color="#000000" textAlign="right" visible="true" enabled="true"/>
</s:Skin>
</s:Button>
```

Figure 9. Part of the UsiXML custom CUI-code, XSL-template and MXML-output code for the custom button.

ACKNOWLEDGMENTS

This work has been Funded by the Academy of Finland-JSPS Core PSESC project. I would like to thank Mr. Jaakko Hyry, Goshiro Yamamoto and Daisuke Yoshitake for their help and assistance.

REFERENCES

1. ActionScript. <http://www.adobe.com/devnet/actionscript/> (Adobe Systems Incorporated, March 12th, 2010).
2. Adobe: Flash Player. <http://www.adobe.com/products/flashplayer/> (Adobe Systems Incorporated, March 12th, 2010).
3. Ali, M.F., Pérez-Quinones M.A., Abrams M. Building Multi-Platform user interfaces with UIML. In: Seffah, A., Javahery, H. (eds.), "Multiple User Interfaces: Engineering and Application Framework". John Wiley and Sons, New York (2003).
4. Bastide, R., Palanque, P. A Visual and Formal Glue Between Application and Interaction. *Journal of Visual Language and Computing* 10, 5 (1999).
5. Coenraets, C. An Overview of MXML: The Flex Markup Language. Adobe Systems Incorporated, March 12th, 2010. <http://www.adobe.com/devnet/flex/articles/paradigm.html/>.
6. Collignon, B., Vanderdonckt, J., and Calvary, G. Model-Driven Engineering of Multi-Target Plastic User Interfaces. In *Proc. of 4th IARIA International Conference on Autonomic and Autonomous Systems ICAS'2008* (Gosier, 16-21 March 2008). D. Greenwood, M. Grottkke, H. Lutfiyya, M. Popescu (eds.), IEEE Computer Society Press, Los Alamitos, 2008, pp. 7-14.
7. Coyette, A., Kieffer, S., and Vanderdonckt, J. Multi-Fidelity Prototyping of User Interfaces. In *Proc. of 11th IFIP TC 13 Int. Conf. on Human-Computer Interaction INTERACT'2007* (Rio de Janeiro, September 10-14, 2007). Lecture Notes in Computer Science, Vol. 4662, Springer-Verlag, Berlin, 2007, pp. 149-162.
8. da Silva, P., Paton, N. User Interface Modeling in UMLi. *IEEE Software* 20, 4 (July/August 2003), pp. 62-69.
9. Ehrig, H., Engels, G., Kreowski, H.J. and Rozenberg, G (eds.), *Handbook of Graph Grammars and Computing by Graph Transformation, Application, Languages and Tools*, Vol. 2, World Scientific, Singapore, 1999.
10. Fardoun, H., Montero, F., and Lopez-Jaquero, V.L. eLearnXML: Towards a model-based approach for the development of e-Learning systems considering quality. *Advances in Engineering Software* 40, 12 (2008) pp.129-.
11. Flash Catalyst beta. Adobe Systems Incorporated, March 12th, 2010. <http://labs.adobe.com/technologies/flashcatalyst/>
12. Flash Builder 4. Adobe Systems Incorporated, March 12th, 2010. <http://labs.adobe.com/technologies/flashbuilder4/>
13. GrafiXML. Université catholique de Louvain, Belgium 2004-2007, March 12th, 2010. <http://www.usixml.org/index.php?mod=pages&id=12>.
14. Hickey, S., Yamamoto, G., Pitkanen, A., Hyry, J., and Yoshitake, D. Implementation of a picture based user interface to assist the elderly suffering from memory problems. In *Proc. of the 8th International Conference on Virtual Reality Continuum and its Applications in Industry 2009*, pp. 351-356.
15. IdealXML. Université catholique de Louvain, Belgium 2004-2007, March 12th, 2010. <http://www.usixml.org/index.php?mod=pages&id=15>
16. Klemmer, S. R., Sinha, A. K., Chen, J., Landay, J. A., Aboobaker, N., & Wang, A. SUEDE: AWizard of Oz prototyping tool for speech User Interfaces. In *Proceedings of UIST 2000, The ACM Symposium on User*

- Interfaces and Software Technology*, CHI Letters, 2(2), 2000, pp.1-10.
17. Lim, Y., Pangam, A., Periyasami, S. and Aneja, S. Comparative analysis of high and low-fidelity prototypes for more valid usability evaluations on mobile devices. In *Proceedings of NordiCHI International Conference on Computer Human Interaction NordiCHI'06* (Oslo, Norway, October 14-18). ACM Press, New York, NY, 2006, pp. 291-300.
 18. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., and Lopez, V. UsiXML: a Language Supporting Multi-Path Development of User Interfaces. In *Proc. of 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems EHCI-DSVIS'2004* (Hamburg, July 11-13, 2004). Lecture Notes in Computer Science, Vol. 3425, Springer-Verlag, Berlin, 2005, pp. 200-220.
 19. Martínez-Ruiz, F.J., Muñoz Arteaga, J., Vanderdonckt, J., and González-Calleros, J.M., A First Draft of a Model-driven Method for Designing Graphical User Interfaces of Rich Internet Applications. In *Proc. of 4th Latin American Web Congress LA-Web'2006* (Puebla Cholula, 25-27 October 2006), J.A. Sanchez (ed.), IEEE Computer Society Press, Los Alamitos, 2006, pp. 32-38..
 20. Montero, F., López-Jaquero, V., Vanderdonckt, J., Gonzalez, P., Lozano, M.D., and Limbourg, Q. Solving the Mapping Problem in User Interface Design by Seamless Integration in IdealXML. In *Proc. of 12th Int. Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'2005* (Newcastle upon Tyne, 13-15 July 2005), S.W. Gilroy, M.D. Harrison (eds.), Lecture Notes in Computer Science, Vol. 3941, Springer-Verlag, Berlin, 2005, pp. 161-172
 21. Newman, M.W., Lin, J., Hong, J.I., Landay, J.A.: DENIM: An Informal Web Site Design Tool Inspired by Observations of Practice. *Human-Computer Interaction* 18, 2003, pp. 259-324.
 22. Oney, S., Myers, B., and Zimmerman, J. (2009). Visions for Euclase: Ideas for Supporting Creativity through Better Prototyping of Behaviors. Workshop on Computational Creativity, ACM Conference on Human Factors in Computing Systems, Boston, MA, April 4, 2009 pp.1-4.
 23. Openlaszlo. <http://www.openlaszlo.org/> (Laszlo Systems, Inc., March 12th, 2010).
 24. Petrie, J.N., Schneider, K.A. Mixed-fidelity Prototyping of user interfaces. In: *Proc. of DSV-IS 2006*, Springer, Heidelberg 2006. pp. 199-212
 25. Puerta, A., Eisenstein, J. XML: A Common Representation for Interaction Data. In *Proc. of ACM Int. Conference on Intelligent User Interfaces IUI' 2002* (San Francisco, January 13-16, 2002). ACM Press, New York, 2002.
 26. RIA. http://www.adobe.com/resources/business/rich_internet_apps/#open (Adobe Systems Incorporated, March 12th, 2010).
 27. SketchiXML. Université catholique de Louvain, Belgium 2004-2007, March 12th, 2010. <http://www.usixml.org/index.php?mod=pages&id=14>
 28. Sorokin, L, Montero, F., Martín, C.,. Flex RIA Development and Usability Evaluation. WISE Workshops 2007: 447-452.
 29. Sottet, J.S., Calvary, G., and Favre, J.M. Toward Model Driven Engineering of Plastic User Interfaces. International workshop on Model Driven Development of Advanced User Interfaces, MDDAUI, Jamaica, 2005.
 30. Taivalsaari, A. Mashware: The future of web applications. Technical report, Sun Microsystems, Feb 2009. pp. 3-11
 31. TeresaXML. Human Interfaces in Information Systems Laboratory, March 12th, 2010. http://giove.isti.cnr.it/tools/TERESA/teresa_xml.html.
 32. TransformiXML. Université catholique de Louvain, Belgium 2004-2007, March 12th, 2010. <http://www.usixml.org/index.php?mod=pages&id=34>
 33. UsiXML. Université catholique de Louvain, Belgium 2004-2007, March 12th, 2010. <http://www.usixml.org>
 34. Winckler, M., Trindade, F., Stanciulescu, A., and Vanderdonckt, J. Cascading Dialog Modeling with UsiXML. In *Proc. of 15th Int. Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'2008* (Kingston, July 16-18, 2008), Lecture Notes in Computer Sciences, Vol. 5136, Springer, Berlin, 2008, pp. 121-135..
 35. von Tetzchner, S., and Martinsen, H. Introduction to symbolic and augmentative communication. London: Whurr Publishers, 1992.
 36. XAML. <http://msdn.microsoft.com/en-us/library/ms752059.aspx> (Microsoft, March 12th, 2010).
 37. XML User Interface Language (XUL) 1.0, Mozilla Foundation, March 12th, 2010. <https://developer.mozilla.org/en/XUL>

WallShare: A Multi-Pointer User Interface

Pedro G. Villanueva

LoUISE Research Group
University of Castilla-La
Mancha

Campus Universitario, s/n,
Spain

pedro.gonzalez@uclm.es

Ricardo Tesoriero

LoUISE Research Group
University of Castilla-La
Mancha

Campus Universitario, s/n,
Spain

ricardo.tesoriero@uclm.es

Jose A. Gallud

LoUISE Research Group
University of Castilla-La
Mancha

Campus Universitario, s/n,
Spain

jose.gallud@uclm.es

ABSTRACT

We introduce a new system to improve the collaboration possibilities among the participants in face-to-face meetings and working groups, called WallShare. WallShare is a new interaction device and a new platform to develop collaborative applications. The proposed system provides a shared zone that is displayed by a projector over a wall. Users can collaborate through the shared zone using their own mobile devices, such as mobile phones, PDAs, laptops and so on. To use the shared zone, users have their own cursors that allow them to share any kind of files, such as images, or documents. WallShare has been proved helpful to support distributed user interfaces. The usability evaluation also showed us that WallShare users' can perform a set of tasks with effectiveness, productivity and satisfaction.

Keywords

HCI, UI Distribution, Interaction resources, Mobile devices

INTRODUCTION

Mobile devices are highly available among users, which has motivated the emergence of novel interaction devices mainly on the entertainment market. Traditional devices such as keyboards, screens and mice seem to belong to an ancient age and they are being gradually substituted by new devices that support ways of interaction such as natural or gestural interaction.

Many IT researchers keep trying to answer the question of how technology can improve people's life. This project focuses on how to improve the collaboration on face-to-face meetings or working groups by means of portable devices.

In this paper we present the WallShare application, a new collaborative system that is controlled by portable devices. WallShare allows mobile users to collaborate by means of sharing the same desktop to perform collaborative activities like sharing documents, images or videos in the same place at the same time.

WallShare implements the concept of distributed user interfaces as the interaction performed by several users on their mobile devices affects the unique and shared interface.

The paper is organized as follows. Section related work describes the related work in this research area. We will focus on distributed user interfaces, the design of new interaction devices and the design of groupware applications. Section WallShare motivation starts with the system goals and dis-

cusses some technologies initially raised. Section system description introduces the system from the point of view functionality, architecture and implementation and finally, discusses some benefits. Section supporting distributed user interfaces explains why WallShare is composed of distributed user interfaces. Section domain specific WallShare application presents other application scenarios for WallShare, and finally, the last section presents the conclusions and future work.

RELATED WORK

The related work is focused in the next research fields: Distributed user interfaces, the design of new interaction devices and the design of groupware applications. The modeling and design of distributed user interfaces (DIU) is facing the complex mobile and dynamic world of managing users' interaction with heterogeneous devices. The development of new devices that support the proposed DIU models provide users with an interface divided among the different devices and introduce new interaction and collaboration possibilities [1, 2, 3, 22, 23].

The distribution of an interface among different platforms can be seen in our computer desktops where users employ two or more screens to extend the interface. In [4, 5], the authors improve this idea with different proposals that employ simplex screen devices.

Some works employ distributed user interfaces to migrate resources or an entire system among different platforms, as in the work presented by Bharat & Cardelli [6]. The Pick&Drop paradigm offers a way to copy and paste resources between mobile devices making use of wireless network [7].

From the design of new interaction devices point of view, this work covers both the software and hardware perspectives and most relevant works on this line are described in the next lines. The ReacTable is a new tangible device based on a tabletop interface to compose music [9]. Similar devices have appeared during the last months. For instance, Touchpad Pro [13] allows users to control a pointer. However, multi-pointer is not supported. Another alternative that follows the same goal is Control Mac [8], which uses the iPhone to control presentations remotely. Microsoft Multipoint [10] is another similar device. It allows users to control a remote pointer using mouse devices.

In [11], an interaction system based on RFID technology allows mobile users (users with a mobile device) to interact with interactive panels. These panels are traditional information panels that were enriched with RFID tags allowing both information retrieval and updating through user gestures.

Another interesting work is the one presented in [12], where a mobile application for improving the visitor's experience in museums is described. There are similar works that claim to solve the problem of positioning a user in indoor spaces using different technologies.

From the collaborative system point of view, we have to highlight the design electronic meeting systems (EMS) [14] that allow meeting participants to perform collaborative tasks through electronic devices that are embedded in the room.

WallShare improves related work in the sense it provides users with a new interaction mechanism to perform collaborative tasks as sharing resources. As we explain later, the novelty of WallShare lies in the multipointer support.

WALLSHARE MOTIVATION

New interaction devices and systems have designed to accomplish a specific objective (entertainment, fun, work, etc.) in a given context. In this section we explain the motivation behind the creation of WallShare and its context.

There is a common starting point in the origin of many new interaction devices, the evidence about the continuous growing of mobile devices and intelligent systems in users' daily life. The users' interaction possibilities depend on different factors, as the connectivity capabilities of their mobile and other surrounding smart devices.

A second common motivation is the need to introduce collaboration capabilities in the mobile devices. These new functions can be reached by adding special hardware into the mobile device and/or by developing software that provides collaboration functions using the special or given hardware.

The third ingredient in many successful systems is the user's capability to change the context using the mobile device. This change can come in the form of sharing resources, playing collaborative games, or acting on the physical environment (turn on lights).

As we explained in previous sections, there are many examples of systems that employ these three ingredients: connectivity, mobility and context-awareness. Microsoft Surface and Nintendo Wii make use of mobile devices to allow users to interact with an intelligent table or a screen. EcoPanels employs a sensorized panel to allow users to interact by means their mobile device.

New interaction mechanisms need not popular hardware technology, especially in context-awareness applications. For example, EcoPanels system needs mobile devices integrate a RFID reader, the ReacTable and Microsoft Surface

need infrared cameras, Nintendo Wii uses a remote controller that makes use of different non widely used hardware technologies and so on. The deployment of these applications is something critical as can be noted in the different showed cases. Finally, widely distributed smartphones, mobile devices or PDA cannot be used to interact together with these new interaction systems, but with the addition of specific hardware component.

The previous paragraphs have established the challenge behind WallShare. Additionally, the non-functional requirements of the system have been defined and are the following:

- Common mobile devices: The system should not require specific or no widely accepted hardware integrated in the users' mobile devices.
- Common communications protocol: The system should use common or popular communication protocol as can be Wi-Fi or Bluetooth.
- Users' actions should influence on the environment: the user should be able to interact with other users and to influence in the environment some way.
- The system as platform: the system should not be an isolated application but a framework to implement different applications.
- Next section explains the main features of WallShare and the selected technology taking into account these requirements.

SYSTEM DESCRIPTION

In this section we describe the WallShare system. We first introduce the software functionality, then we describe the architecture of the system and, finally, we expose some details about implementation issues.

WallShare is a client-server system based on a shared zone that is projected on a wall or screen, which is clearly visible by the participants of a meeting. The shared desktop of the application is exposed on Fig. 1 and the PDA-client application is depicted on Fig. 2.

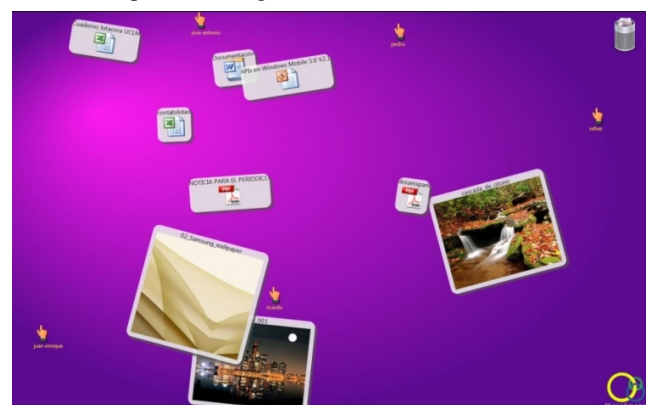


Figure 1. WallShare shared zone with different pointers (5) representing different users.



Figure 2. WallShare mobile client.

System functionality

The system functionality is provided in two versions, the mobile client and the server-desktop client. The mobile and server-desktop clients share a common set of functionality. However, the server-desktop client provides the user that leads the meeting extra functionality to manage the conversation.

The mobile client functionality

The client application is a mobile application that runs on participants mobile devices to share resources through a shared zone. Figure 3 shows the simplified version of the CTT diagram of the client application.

In order to access the WallShare system, participants download and install the client application into a mobile device from the WallShare server. This application is automatically updated whenever a new version is released in the server.

In order to interact with the system, participants connect their mobile devices to the server application. When participants get connected to WallShare, they receive a double feedback, a sound from the server machine, and a pointer representing each user on the screen.

An interesting aspect of WallShare is the possibility of providing each participant with the ability to control the movement of a cursor that represents him/her on the screen just by performing dragging gestures on the mobile device screen. Thus, users can use the mobile device as an enhanced X-Pointer device.

Therefore, connected participants are able to download and upload resources from and to the shared zone. To perform this task, users select resources from the mobile device through the client application and upload them performing

a simple gesture. When the resource was uploaded, it is shown on the shared screen.

In order to download resources from the shared zone, a participant points the resource to download with the cursor. Once the object was pointed, the user is able to perform a double click on the mobile device screen to download it to his/her mobile device.

WallShare also allows users to post notes on the shared zone as post-its (see Figure 4a). These post-its are really useful to highlight information that is relevant to the meeting. Post-its are an interesting way to post anonymous information.

A chatting system is provided by the WallShare application in order to send information that may, or may not, be relevant to the meeting (see Figure 4b). Unlike post-its, chat messages are associated to an author. Messages are displayed on the lower left corner of the screen.

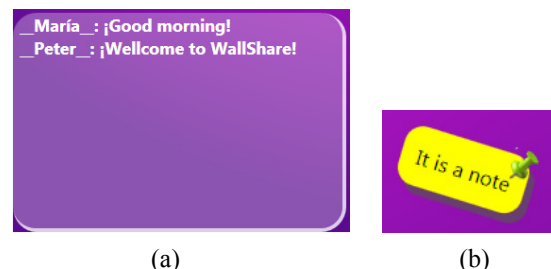


Figure 4. (a) Chat messages and (b) Post-it.

When a user logs out from the system, he/she receive a double feedback: a sound from the server machine and the pointer representing the user disappears from the screen.

The desktop server functionality

The server application provides some functionality that is very useful for the participant leading the meeting in order to manage the look of the shared zone. Figure 5 shows a simplified version of the CTT diagram of the server client application including:

- The customization of shared zone wallpaper: the system provides the adaptation of the shared zone to the environment according to, for instance, illumination issues of the meeting room or participants (i.e. color-blindness). It also may be used to distinguish different meetings within a single room.

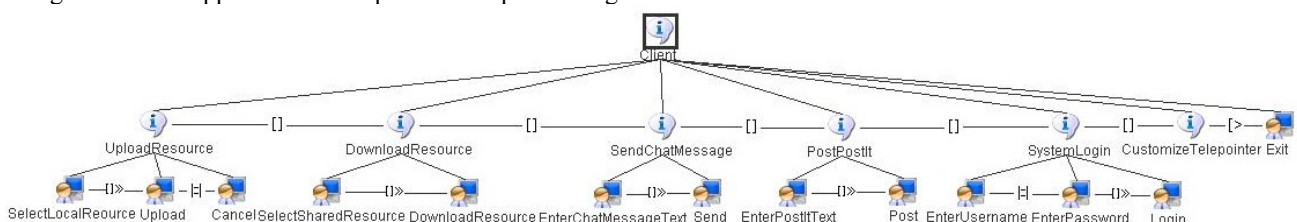


Figure 3. Client CTT diagram (simplified version).

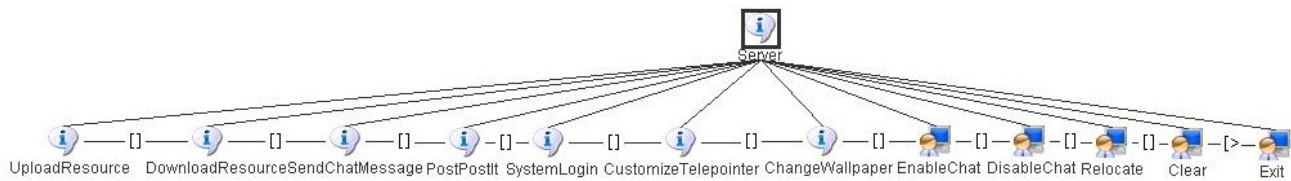


Figure 5. Server CTT diagram (simplified version).

- The chat enabling/disabling function gains some space from the shared zone when this functionality is not required. Besides, it also prevents chat abuse from participants.
- The resource relocation functionality avoids resource overlapping. The resource rearrangement is performed automatically using a random location algorithm in a fashion way.
- The shared zone clean function sends shared resources to the recycle bin. This functionality may be useful to set a starting point in the middle of a meeting. Figure 6 shows the shared zone with the presence of different users.

System architecture

The WallShare client-server application, depicted on Fig. 6, is based on two systems: the client and the server. Figure 7 shows the components that define each system.

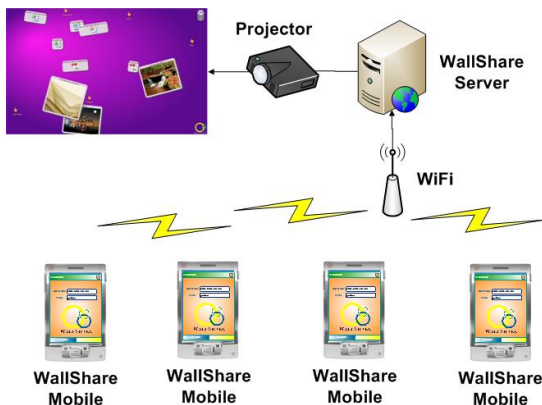


Figure 6. WallShare shared zone.

The server system

The main goal of the server system is the provision of communication, resource sharing and the coordination services to the client systems.

The server system runs on a desktop computer that is connected to a wireless network via a Wi-Fi or Bluetooth connection. It is also connected to the visualization system, i.e. a projector or large screen, to support the shared zone visualization to the participants.

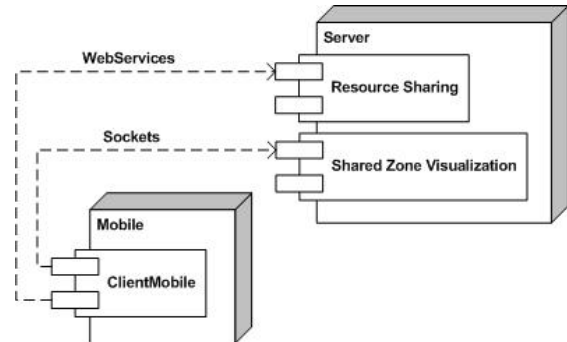


Figure 7. Client-server application.

The server system is composed by two basic components:

1. The resource sharing component
2. The shared zone visualization component

The resource sharing component is in charge of performing resource transferring between the client and the server applications. The resource transferring is performed when clients download/upload resources from/to the server. This communication is performed using Web Service technologies. On the other way, the shared zone visualization component is in charge of synchronizing and displaying client actions, such as cursor moving the on the shared zone. Besides, it is responsible for displaying resources sent to the resource sharing component on the shared zone.

Because of performance issues, we have decided to implement the communication between the clients and the shared zone component through sockets (instead of using web services).

The client system

The main goal of the client system is providing to the server the participants actions through an easy to use interface in order to share resources during a meeting.

The client application runs on the user mobile device and it is connected to the server application through a wireless network via a Wi-Fi or Bluetooth connection.

The client is connected to the server components according to the task to be performed. To upload or download a file, the client connects to the resource shared component of the server system to transfer the file via Web Services.

However, to perform actions of the shared zone (i.e. move the cursor on the wall, post a note, send a chat message, etc.), the client connects to the shared zone visualization component of the server system via a socket based connection to improve system performance.

Implementation issues

This section exposes how the WallShare prototype was implemented. Figure 8 shows the class diagram of the server application. Thus, the WallShare server application model is composed by the following classes:

- **WallShare.** The WallShare class manages resources shared and pointers moved for the users.
- **Resource.** The Resource abstract class represents every resource that can be shared to WallShare. Specific classes are Image and OthesResource.
- **Image.** The Image class represent a image resource.
- **OtherResource.** The OtherResource class every resource that it is not a picture. Image is different from OtherResources because Image shows the picture and OtherResources shows the application icon that to open the resource.
- **Pointer.** The Pointer class represents each pointer controlled by the user. A pointer is defined by its position (X and Y) and a icon.
- **WaitConnections.** The WaitConnections class waits connections from the client application and WaitConnections creates a ManagerClient for each connection.
- **ManagerClient.** ManagerClient is a class responsible for send and receive datas to and from the client applications. Resources (represented by instances of the Resource class)

The sequence diagram showed on Figure 9 explains how a connection is established between the server and a client, focusing on the connection manager object.

When a client is connected to WallShare, the object WaitConnections sends a request to the WallShare object, which is responsible to control the shared pointers, asking for creating a new pointer. WaitConnections object also creates a new ManagerClient instance, which will control the communication between WallShare and this new client. As the client performs actions in his/her mobile device, the actions are sent and collected by the associated ManagerClient. These actions are applied on the corresponding pointer client.

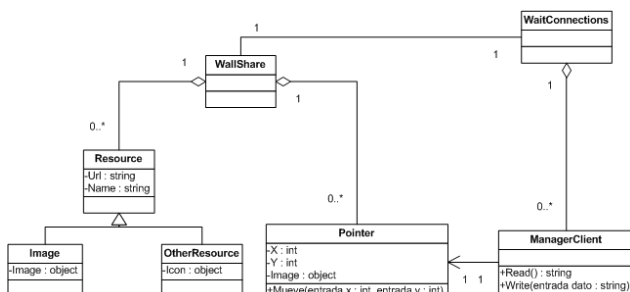


Figure 8. WallShare class diagram (server side).

While pointers represent users on the shared area, resources represent the objects (i.e. images, text documents, spread sheets and so on) that are exchanged through the

application. Images are shown directly on the shared zone. The other documents are shown using a representative icon.

The prototype has been implemented using mobile devices running Microsoft Windows Mobile and Microsoft Windows XP operating systems. The desktop application has been developed using the Microsoft .NET Framework 3.0.

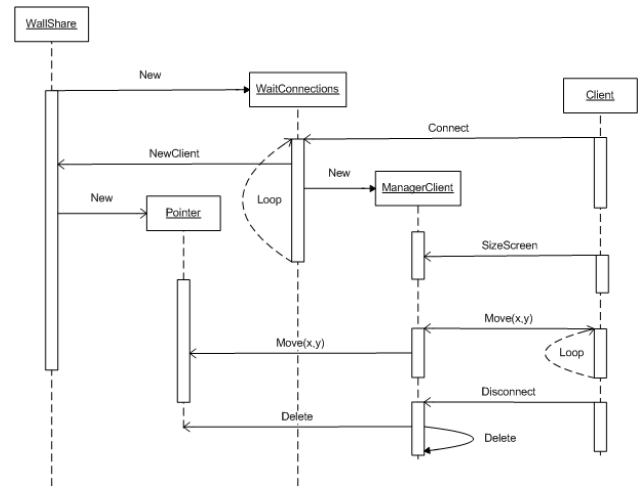


Figure 9. WallShare sequence diagram.

From hardware infrastructure point of view, the WallShare server requires a video projector, a projection zone, like a wall or a screen, a Wi-Fi or Bluetooth access point and a server machine. The client applications can be deployed on any portable devices, such as mobile phones, PDAs, laptops and so on supporting Bluetooth or Wi-Fi connectivity. A PC with Wi-Fi connectivity support can be used as well.

SUPPORTING DISTRIBUTED USER INTERFACES

As mentioned in the related work, the use of DUIs in mobile environments is a very interesting concept. The main goal of a DUI is to facilitate the tasks mobile users want to execute by providing them with an optimal configuration of a set of interconnected interaction resources. In this case, the interconnected interaction resources are the mobile screen and the shared desktop.

Mobile devices generally have a small screen. WallShare extends the mobile device interface with the projected interface.

The user can work with the device and when s/he moves the mouse over the client application, s/her associated cursor in shared desktop is moved as well. In this way the user sees the shared zone as an extension of his/her mobile device interface to the common shared zone of WallShare.

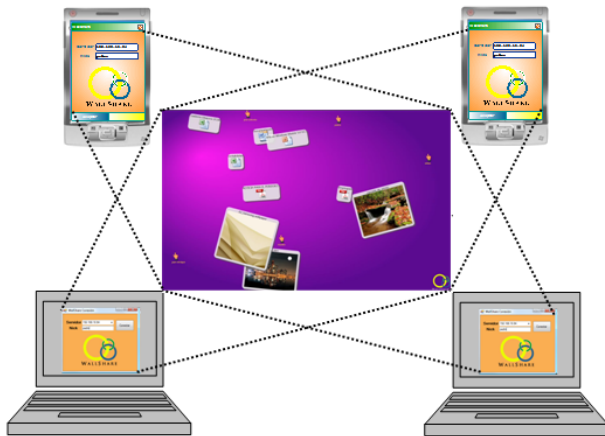


Figure 10. WallShare implementation of distributed user interfaces.

Figure 10 shows the WallShare's vision of the distributed user interface paradigm where four different users interact with their own mobile devices sharing a common public zone called UI WS. All four users are able to move their own pointers on this shared zone.

Domain specific WallShare applications

WallShare is not an isolated application but can be seen as a set of domain specific applications with similar collaborations needs. This means that WallShare can create platform beyond the presented application to improve face-to-face meetings.

Some of the scenarios are described in the next lines:

- **Games:** WallShare can be easily applied to support collaborative games where several users share common objectives. A puzzle is a good example of a WallShare game. All users see the evolution of the puzzle and collaborate to put the different pieces on the puzzle.
- **Education:** A number of different WallShare applications can be defined to support teaching activities. A group of students perform collaboratively a task as associate animal images with description.
- **Entertainment:** Restaurants, café or pub can use WallShare to allow customers to share their resources (images or video) while spend time with friends.
- **Science:** WallShare can be used in working groups to create documents in brainstorming meetings.

USER EVALUATION

User testing is a necessary activity when new interaction devices or techniques are proposed. The difficult arises at the time to choose an evaluation method therefore the newer defined aspects are going to be evaluated with precision.

In this section we present the results of WallShare usability evaluation based on the ISO 9126-4 standard [15]. Specifically, we have focused on the effectiveness, productivity and satisfaction as the main factors that influence in the quality in use concept.

Effectiveness has been measured using these three metrics: Effectiveness, Task completion and Error frequency. To measure productivity we have used Task time and Task efficiency. Satisfaction has been measured using a questionnaire based on the SUMI proposal [21].

The experiment was defined as follows. A group of 17 users, range ages from 19 to 47 years old, without technological expertise were asked to perform four specific tasks: (1) Installing the client version of WallShare and login, (2) To share a resource (upload and download), (3) Add a note with a specific text, (4) Send a message to the shared chat with a specific text. Additionally participants were asked to fill the SUMI based satisfaction questionnaire.

In this section we present the most relevant results. Figure 11 shows the Task Time productivity metric. The max and min values allow us to measure the confidence interval of the mean value. As can be noted, once the client application has been installed, users have not reported problems to use the system.

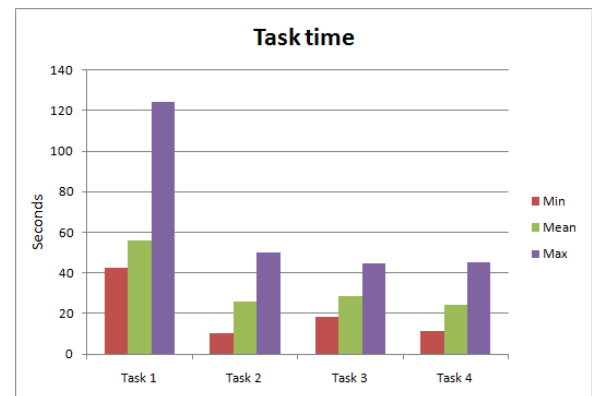


Figure 11. Task time (mean, max and min values) expressed in seconds.

Figure 12 shows the metric Task Efficiency as is defined in the ISO 9126-4 (the ratio between Task Effectiveness and Task Time). Satisfaction results showed a 77.4% of users with a positive opinion using Wallshare, 15.2% of users feel neutral and 7.4% were not satisfied using WallShare.

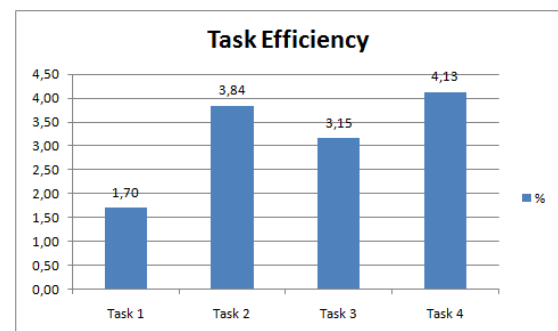


Figure 12. Task Efficiency. Each result shows the mean of the users' Task Efficiency.

Other benefits and expected results

If we take a look to the Motivation section, the requirements list can be examined:

- Common mobile devices: WallShare does not require any special or not common hardware.
- Common communications protocol: WallShare uses widely extended protocols as WiFi and Bluetooth.
- Users' actions should influence on the environment: In WallShare, users' interactions influence in the common shared zone, therefore it influences in the environment.
- The system as platform: WallShare can be seen not only an application but also as platform to build other different collaborative applications.

WallShare improves the interaction and collaboration among participants of meetings or working groups. In particular, some remarkable advantages of this approach are the following:

- WallShare support multipointer on the shared zone. The response time of the system is good enough as light movements of the user's mobile pointer correspond with movements of the remote pointer in the shared zone. Actions, such as double click on the mobile device screen, are mapped to similar commands in the shared zone (select, share, etc).
- WallShare works on mobile devices and also on laptop and PC's. The deployment of the client application is easy and can be done by downloading the application directly from the WallShare server to the mobile device.
- WallShare extends the functionality of mobile devices
- WallShare allows users to share resources: documents, video and audio.
- WallShare is intuitive and friendly: it employs natural interaction
- Collaborative face-to-face activities are well supported in WallShare as all users can share, publish and view their resources and comments on the common display.

MULTIPLATFORM UI SUPPORT FOR WALLSHARE

This paper exposes a multi-display application to support electronic meetings. In order to support the mobile and desktop version of the application two separated .NET applications were developed. The desktop client was developed using standard .NET framework and the mobile client was developed using the .NET Compact Framework. Thus, two applications have to be maintained at the same time when new functionality is introduced into the client application. To cope with this problem we have decided to model the application using UsiXML [18, 19]. A simplified version of the AUI model built with IdealXML [20] is exposed on Figure 13.

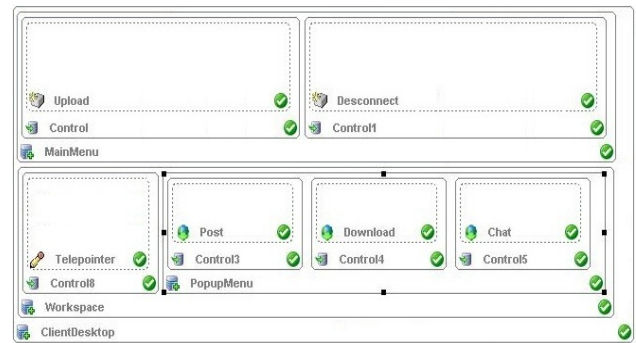


Figure 13. WallShare Client application AUI model.

Thus, UsiXML provides us a way to model the WallShare application in a platform independent way that allows us to maintain a single development line.

CONCLUSIONS AND FUTURE WORK

This paper presents an innovative collaborative system to improve face-to-face meetings or working groups called WallShare. WallShare takes advantage of the distributed user interface paradigm as the users' interactions, performed on their mobile devices, are shared in the common public interface. By means of WallShare, a group of people participating in a meeting or working group could share documents using a simple gesture on his/her mobile device. WallShare has been designed, implemented and tested with real users with good results. WallShare extend the functionality of everyday devices, as mobile phones or PDAs. WallShare also creates platform, which means the possibility to apply the system in many domain specific applications as entertainment, games, teaching and so on.

WallShare has been tested with a group of non-technical users getting a high degree of satisfaction. Quality in use metrics show us that users can perform common WallShare tasks with effectiveness, productivity and satisfaction.

Finally, this article presents simplified modes based on UsiXML for future implementations of the client using this technology. Future work includes the deployment of a new version for running on other mobile OS platforms.

REFERENCES

1. Joëlle Coutaz, Lionel Balme, Christophe Lachenal, and Nicolas Barralon. *Software Infrastructure for Distributed Migratable User Interfaces*. UbiHCISys 2003 Workshop on UbiComp 2003, 2003.
2. Peter Tandler, Thorsten Prante, Christian Müller-Tomfelde, Norbert Streitz, and Ralf Steinmetz. *ConnecTables: Dynamic coupling of displays for the flexible creation of shared workspaces*. Proceedings of the 14, Annual, ACM Symposium on User Interface Software and Technology (UIST'01), pages 11–20, 2001.
3. Chris Vandervelpen and Karin Coninx. *Towards model-based design support for distributed user interfaces*.

- In Proceedings of the third Nordic Conference on Human-Computer Interaction, pages 61–70. ACM Press, 2004.
4. Ken Hinckley, Gonzalo Ramos, Francois Guimbretiere, Patrick Baudisch, and Marc Smith. *Stitching: Pen Gestures that span Multiple Displays*. In AVI '04: Proceedings of the working conference on Advanced visual interfaces, pages 23–31, 2004.
 5. Jun Rekimoto and Masanori Saitoh. *Augmented surfaces: A Spatially Continuous Work Space for Hybrid Computing Environments*. In Proceedings of CHI'99, 1999.
 6. Bharat, K.A., Cardelli, L.: *Migratory Applications Distributed User Interfaces*. In: Proc. of ACM Conf. on User Interface Software Technology UIST'95. ACM Press (1995) 133–142.
 7. Rekimoto, J.: *Pick-and-Drop: A Direct Manipulation Technique for Multiple Computer Environments*. In: Proc. of UIST'97. ACM Press, New York (1997) 31–39
 8. Control Mac. <http://www.soydemac.com/2008/03/18/controlar-un-mac-desde-el-iphone/>
 9. Jorda, S. et al.: *ReacTable: Exploring the Synergia between Live Music Performance and Tabletop Tangible Interface*. Proc. of TEI 2007, ACM pp. 139-146. 2007.
 10. Microsoft Multipoint. <http://www.microsoft.com/unlimitedpotential/TransformingEducation/MultiPoint.msp>
 11. Tesoriero, R., Tébar, R., Gallud, J. A., Penichet, V. M. R., Lozano, M.: *Interactive EcoPanels: Paneles Ecológicos Interactivos basados en RFID*. Proceedings of the IX Congreso Internacional de Interacción Persona-Ordenador Interacción 2008. ISBN: 978-84-691-3871-7; pp 155-165.
 12. Tesoriero, R.; Gallud, J. A.; Lozano, M. D.; Penichet, V. M. R.: *A Location-aware System using RFID and Mobile Devices for Art Museums*. Fourth International Conference on Autonomic and Autonomous Systems. IEEE/CS Press. IARIA-ICAS 2008. Páginas 76 - 81.
 13. Touchpad Pro. <http://www.touchpadpro.com/>
 14. J. F. Nunamaker , Alan R. Dennis , Joseph S. Valacich , Douglas Vogel , Joey F. George, *Electronic meeting systems*, Communications of the ACM, v.34 n.7, p.40-61, July 1991
 15. ISO/IEC 9126-4: Software Engineering - Software product quality -Part 4: Quality in use metrics. October 2002.
 16. Terrenghi, L., Quigley, A., Dix, A., *A taxonomy for and analysis of multi-person-display ecosystems*. Personal and Ubiquitous Computing (2009). Springer-Verlag, 13:583-598.
 17. Izadi S, Brignull H, Rodden T, Rogers Y, Underwood M (2003): *Dynamo: a public interactive surface supporting the cooperative sharing and exchange of media*. In: Proceedings of the 16th annual ACM symposium on user interface software and technology (Vancouver, Canada, 02–05 November 2003). UIST'03. ACM, New York, pp 159–168.
 18. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., and Lopez, V. UsiXML: a Language Supporting Multi-Path Development of User Interfaces. In *Proc. of 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems EHCI-DSVIS'2004* (Hamburg, July 11-13, 2004). Lecture Notes in Computer Science, Vol. 3425, Springer-Verlag, Berlin, 2005, pp. 200-220.
 19. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., and Florins, M. UsiXML: A User Interface Description Language Supporting Multiple Levels of Independence. In *Proceedings of Workshop on Device Independent Web Engineering DIWE'04* (Munich, 26-27 July 2004), M. Lauff (Ed.), Munich, 2004.
 20. Montero, F., López-Jaquero, V. IdealXML: An Interaction Design Tool-A Task-Based Approach to User Interfaces Design. In *Proc. of 6th Int. Conf. on Computer-Aided Design of User Interfaces CADUI'2006* (Bucharest, 6-8 June 2006), Chapter 20, Springer-Verlag, Berlin, 2006, pp. 245-252.
 21. Kirakowski, J., Corbett, M.: SUMI: the Software Usability Measurement Inventory, in: British Journal of Educational Technology, Vol. 24 No. 3, 1993.
 22. Grolaux, D., Vanderdonckt, J., and Van Roy, P. Attach me, Detach me, Assemble me like You Work. In *Proc. of 10th IFIP TC 13 Int. Conf. on Human-Computer Interaction INTERACT'2005* (Rome, 12-16 September 2005). Lecture Notes in Computer Science, Vol. 3585, Springer-Verlag, Berlin, 2005, pp. 198-212.
 23. Melchior, J., Grolaux, D., Vanderdonckt, J., and Van Roy, P. A Toolkit for Peer-to-Peer Distributed User Interfaces: Concepts, Implementation, and Applications. In *Proc. of 1st ACM SIGCHI Symposium on Engineering Interactive Computing Systems EICS'2009* (Pittsburgh, July 15-17, 2009). ACM Press, New York, 2009, pp. 69-78.

Association between Requirements of Interaction of iDTV Learning applications and UsiXML models by addressing the need to structure TVilos in Crossmedia

Julio Guido Militão¹, Elizabeth Furtado^{1,2}, Ítalo Matos¹

¹UNIFOR, Universidade de Fortaleza – (Computer Science Department, LUQS)
1321, Washington Soares Ave, Fortaleza, CE, 60.811-905, BRAZIL, +55 85 3477 3079
militao@unifor.br, elizabet@unifor.br, italomatos@gmail.com

²UECE, Universidade Estadual do Ceará
1700, Paranjana Ave - Campus do Itaperi – Fortaleza, CE 60740-903, BRAZIL

ABSTRACT

This paper addresses the elicitation problem of interactive and educational requirements when developing T-Learning interactive systems. The framework AMITUDE is used to elicit such requirements of applications and the USiXML is used to model aspects when using these applications. The object of study is a system for handling educational content to be displayed in crossmedia: iDTV, computer and mobile. As a result we've obtained two working prototypes for creating and viewing classes.

Keywords

Interactive Digital TV, Distance Learning, TvILO, VILO

INTRODUCTION

Today, Digital Television is used mainly in homes of Classes A and B. TVs with higher resolutions and more advanced audio and video features also come with converters – called Set Top Box (STB) – which, in their majority, allow interactivity. That technology brings up the idea of using the television as a learning tool. Tomáš Klimčuk uses the term T-Learning meaning: learning with the assistance of a TV receiver and a remote controller. T-Learning course should give the viewer the option to select video content and any additional information, making it enjoyable for the viewer, that should later test his obtained knowledge and approve or not of the program [7]. This is not the first time that television is used for this purpose. In the beginning – with the creation of the *Telecurso* (in portuguese, classes broadcasted on TV) [4,13] – videos were shown as a complement to the content of the books for students. *Telecurso* uses TV to showcase classes through television shows. It was initially implemented in the countryside of the states of Ceará, Maranhão, and Piauí. It is considered the leader of Distance Education – in Brazil, it began in 1965 [4]. In Distance Education, the student has to make his own schedule so that he can watch the video in a determined hour of the day that best suits him.

The Interactive Digital TV (iDTV) solves the two problems mentioned above. The contents are broadcasted by the network and the T-learner can access them in the most convenient time for him. In this context, the interactivity will be conditioned to the existence of a DTV system with

a return channel that allows the effective communication between T-learners and teacher, who is usually the creator of the broadcasted content.

This research implies the existence of such system. This paper suggests the specification requirements of Interactivity and Contents for distance T-learning applications via iDTV. We use the conceptual framework AMITUDE [3] to guide the elicitation of requirements and the UsiXML [14] to formally specify them in several UI Models [15]. Two applications were identified and partially implemented: the TvILO Maker and the TvILO Viewer (Figure 1). The first one allows that content producers create VILO – Learning Objects with the use of Interactive Videos [1], which will be broadcasted on TV. These are defined as TvILOS on this paper. The second one will allow the visualization and interaction of the TvILOS from the STB, as well as charge them.

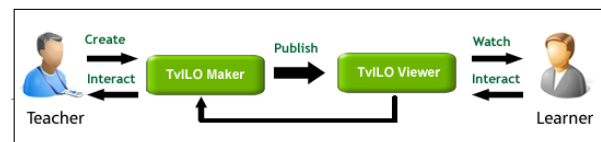


Figure 1. Relation between applications and target users.

The importance of this paper is in the context of the definition of the requirements for the iDTV applications, by analyzing and identifying aspects of interaction and educational contents that can (or can't) be associated to the UsiXML models. The results pointed to the association between all aspects of the AMITUDE (such as: Application, Modalities, and so on) and the UsiXML models. However there was the need of making an extension in the AMITUDE to address requirements of content for both applications such as to define the generated content and to register the behavior of the T-learner as she/he goes through a subject. There was also the need of presenting a solution to approach this extension, because there was no corresponding UsiXML model.

In works that approach the definition and use of UsiXML models [11] the structures of a content (such as: its infor-

mation architecture, navigation, specified contents by media) – considering the existence of a main device (such as in this case, the digital TV) and alternative devices (such as cell phones and web) – are not discussed. They usually mention that the objects of a domain (in our case, educational contents) are associated to user tasks in order to identify which data will be manipulated on User Interfaces. We have to consider that contents have their own architecture and navigation form, independently of relationships among tasks, and can be accessed in different ways in different devices in function of their specifications. This last need requires the specification of contents approaching the necessity of a T-learner to make the transition of one media to one more adequate to her interests.

The crossmedia transition concept in this paper refers to when the T-Learner needs to change the media and the device, so he can continue the interaction seeking complementary content that he is accessing in order to reach his learning goal. Media is all forms in which communication is made, such as newspapers, magazines, radio stations, websites, voice portal, television [9]. The necessity of transitions can derive of requirements related to continuity of the navigation process of the user through the content. For instance, an application being accessed via Digital TV can be the ideal one if the user is answering to questions and if she/he is at home. However, if he needs additional information and needs to leave home then the cell phone can be an alternative and adequate device to continue the access to the content.

The questions of this research are the following: How should the structure of a TvILO be modeled to make its broadcasting possible on iDTV? How can the structure of a TvILO be modeled considering the existence of alternative media from iDTV? How does the content reach the T-learner? It is important to mention that today, the iDTV is a brand new technology and it is full of technological restrictions. The primer solution in this paper in order to provide the crossmedia transition presents itself as another contribution to developers and producers of content that will be broadcasted.

THEORICAL BASIS

iDTV User Interface

The requirements of iDTV User Interface used on our project is based on MHP Usability and Style Guide [7].

The usability requirements must be such as: Maintain the coherence with User's Mental Model (by conserving the same TV notion); Assure the free context interaction (syntax free); Focus on "glanceability"; Avoid interferences, noises or visual alterations that harm the user's interaction with TV and, last but not least, allow interactive participation, universal accessibility and information Interactivity [17].

VILO X TvILO

VILOs are interactive headpieces that when used, allow a T-learner to answer questions about a certain content in exhibition [1]. VILOs use animation, static images and texts. The questions can be presented as a dissertation or as multiple choice. Figure 2 illustrates a situation in which the T-learner is going through a sequence of medias that form a VILO, and finds himself in a situation where it is necessary to intervene.

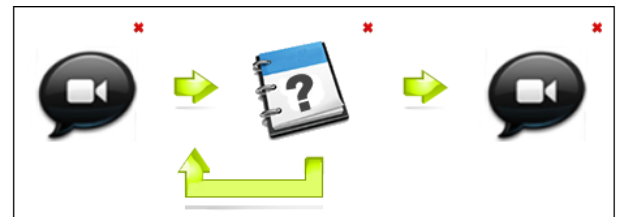


Figure 2. Information flow of a VILO.

Depending on the answer, the VILO sequence changes. In case of a correct answer, the content is either deepened or repeated. When the answer is not correct, a new explanation must be shown. With this method, the narrative becomes non-sequential and it helps on the continuous evaluation of the content because it is possible to know whether or not the T-learner comprehends the subject. The VILOs have the advantage of attending the needs of different learning styles. They can be produced in multiple idioms through dubbing, and can be used as a complement the work of the faculty.

They can also reduce or even eliminate the costs with lab supplies since pictures and animations can be used to simulate chemical reactions or objects that help the pedagogical matter. The production of a video has a high cost but it is an investment that can rapidly be even out because the video will be re-used in a wide way. The TvILO is an adaptation of VILO for the iDTV. In Lab tests, we have identified that Brazilian STB have hardware limitations which unable the charge of videos and sounds. Therefore we established alternatives to these issues.

The scene will be shown to the T-learner through comic books techniques (storyboards) and the audio will be interfused. Therefore, the videos are converted in sequels of static images and texts, as shown on Figure 3. The teachers must also create opportunities to motivate the user to a better navigation flux of the content, indicating the more appropriate media to each moment on the flow.



Figure 3. Information flow of a TvILO.

AMITUDE

AMITUDE is a model of use of an interactive system, a generic model of the aspects involved when someone uses such system. The system used can be helpfully modeled as having the following AMITUDE aspects: Application type, Modalities, Interaction, Task or other activity, Domain, User, Device and Enviroment of use. To illustrate the aspects, consider the following scenario: a User who Interacts with an Application in an Enviroment of use in order to do some Task or other activity by using certain interaction Devices and exchanging information with the system in certain input/output Modalities.

According the authors [3], usability is about fitting systems to people. When the AMITUDE aspects of a system model are exposed to users, we get usability information expressed in the language of fitting (Figure 4). AMITUDE is our main framework. The AMITUDE framework was chosen because it should cover several usability aspects. In order to create a usable system, the seven AMITUDE aspects must be analyzed to some satisfactory extent during specification and design [3]. In addition, we will use UsiXML [11] as a language of fitting.

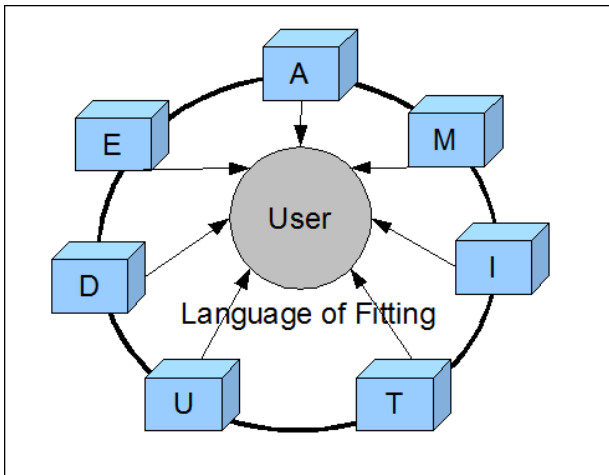


Figure 4. AMITUDE Exposed.

As a multimodal scenario we can refer to the crossmedia functionality. The t-Learner can operate the system using a touch screen cell phone, a TV Set or a computer. These devices have different ways of interact.

UsiXML

UsiXML consists of a User Interface Description Language (UIDL) that is a declarative language capturing the essence of what a UI is or should be independently of physical characteristics. UsiXML is different from a pure UI authoring language as it could be used as a specification language. The goal is not only to generate code, but also to have the capability of reasoning about the UI specifications: model checking, UI evaluation, model-driven engineering, maintenance of repository of UI cases or patterns, static and dynamic analysis and model testing[3].

UsiXML is composed of a set of models. We will use some UsiXML models to map the conceptual aspects of AMITUDE into a formal language called Language of Fitting (Figure 4).

AMITUDE vs. UsiXML

We associate the aspects of AMITUDE to UsiXML models and, from those models. The conceptual modeling of aspects was then specified on models UsiXML according to Table 1.

AMITUDE aspect	UsiXML model
A	Does not apply
M	CUI Model
I	Domain model
T	Task model
U	Context Model (User stereotype)
D	Context Model (Browser UA)
E	Context Model: (environment, hardware and software platform)

Table 1. AMITUDE vs. UsiXML model.

Not all aspects of AMITUDE need a representation model. The Aspect "Type of Application" is defined in a conceptual level and is defined in natural language.

Aspect Modality is defined as a Toolbox of Modalities. We use the CUI Model to identify the possible concret interaction objects that represent the use of languages Linguistic, modal or Arbitrary [3]. The Interaction is registered in the Domain Model and it has aspects of the Tasks using the CTT formalism.

The User, Device and Environment aspects were mapped in the Context Model. The element UserStereotype was designed to record each type of User and their experience in performing the task. The environment is being shaped to register elements Environment, Hardware Platform and software Platform. This part is not finished, due to definitions of crossmedia that are still being made.

In the case of C Aspect, the association of the content to the domain model in UsiXML would not solve our need. We needed - aside of modeling the relation of a content with a task and the relations among the contents - to model the treatment of errors of a specific object flow and to set browsing definitions that could involve the crossmedia transitions. This need was motivated to support the following scenario, for instance: the T-learner is answering the questions on Digital TV following the sequence of a TvILO defined by the teacher at TvILO Maker. If the user commits a mistake then the TvILO Viewer would give her/him the option of getting tips or help through another media.

CASE STUDY

As stated in the Introduction, this paper suggests the specification of requirements that will be applied in Distance Education applications via iDTV using the framework AMITUDE associated to UsiXML language. The relation of UsiXML and AMITUDE allows that conceptually we can evaluate the use of class viewer (TvILO Viewer) via iDTV and computer because the modeling will be abstract (using Abstract Interaction Objects). This method consists of two steps: first, evoking the requirements using AMITUDE, and then implementing concrete interfaces to evaluate its use.

TvILO Maker

1) Elicitation of requirements using AMITUDE

The Elicitation of Requirements of the TvILO Maker was made in sessions of paper prototyping with seven research teachers (who are end users) in order to identify their needs considering how they prepare their lessons, how they would use what they already do, and what are their difficulties. Six of them have experience in distance education, but they have had no experience with iDTV. During the experiment, a prototype was built using DIRCE Methodology [10] by teachers by using low cost materials such as cardboard, icons, colored cards, pen and paper (Figure 5). They were also motivated to think about scenarios where T-learners could alternate between iDTV and other devices.



Figure 5. Paper Prototyping experience using DIRCE Methodology [10].

A	Application Creative module of non-sequential didactic content (TvILO) to Digital TV.
M	Modalities The analogue and linguistic modalities were identified. The analogue will have more emphasis since it uses images, maps and diagrams to represent objects on the inter-

	face.
I	Interaction Interaction will occur between the system and the user through virtual and physical actions.
T	Tasks Interactive interface based on drag-and-drop of media to create a non-sequential class and export it to a format to be read by the Viewer of Content. Identified functionalities: • <i>Creation of TvILO</i> ◦ <i>Insert and Removal of Medias – Static Image, Text, and Question;</i> ◦ <i>Re-utilization of Media;</i> • <i>Export to XML.</i> • <i>Preview on TV</i>
U	Users Teacher (Responsible for planning and organizing the content to compose an educational program) and Programmer (Responsible for developing computer application codes).
D	Devices Computers, DTVs, Internet
E	Environment Site with computer and Internet. Web Navigator. Usually a collaborative work site that allows that many teachers to work together.

Table 2. Requirements of interaction of TvILO Maker.

After this experiment, we filled in Table 2 with all the AMITUDE aspects of the TvILO Maker application. The detailed results can be found in [8]. Regarding the User aspect, Belda [2] introduces other actors in the process of building a class. Among them we have the Instructional Designer, responsible for the elaboration of content presentation strategies and organizing the routes of interactive browsing, considering the media and educational aspects that influence the project) and the Illustrator (responsible for the production of pictures, cartoons, caricatures and other illustrations related to the educational content. When applying AMITUDE, we identified the absence of an aspect to represent the content requirements of the application. We included the C “Content” to register the architecture and the browse form of a TvILO (see Table 3). Figure 6 shows the result of the definition of a TvILO structure represented in a XML file that will be transmitted to Digital TV.

C	Content Main and complementary contents. The main content will be presented on iDTV, the other ones will be accessed through the other devices. Sequential navigation on main contents, and navigation to other devices (occurring a transition between medias) so the user can access complementary contents. Architecture of information top-down, being the contents organized by class and units.
---	--

Table 3. Content requirements of TvILO Maker.


```

class1.xml
<tvilo>
  
  <question id="2" text="1/3 + 2/3 is equals to" duration="3">
    <answers>
      <answer text="3" correct="false" next_id="4"/>
      <answer text="-1" correct="true" next_id="3"/>
      <answer text="1" correct="false" next_id="4"/>
    </answers>
  </question>
  
  
  
</tvilo>

```

Figure 6. XML Data Structures to file a TvILO.

2) Implementation of concrete interfaces

Figure 7 presents the implementation of the TvILO Maker. All the functionalities identified on aspect T of the Table 1 are implemented. It is possible to access the application through web at the address <http://trainingmaker.sidp.com.br/vmakertools/MlosInterface.html#>

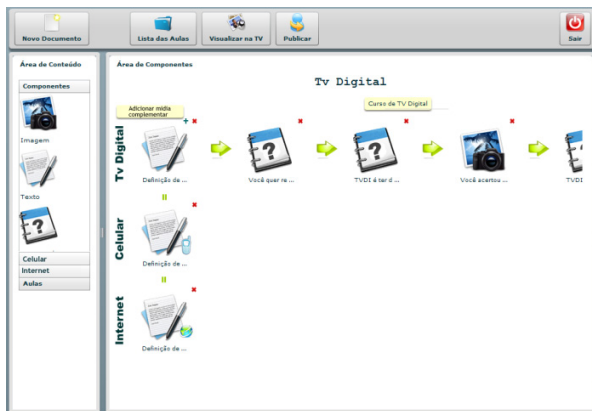


Figure 7. TvILO Maker screen

TvILO Viewer

1) Elicitation of requirements using AMITUDE

The elicitation of TvILO Viewer requirements was made along with the TvILO Maker. Applying the previous method, the AMITUDE aspects of TvILO Viewer were identified and registered as seen on Table 4.

A	Application Module viewer of non-sequential didactical content on Digital TV.
M	Modalities Idem Table 1
I	Interaction On TV, the remote controller will be used to browse and interact. The interaction can be enlarged with movement sensors and other devices connected through USB ports.
T	Tasks Interactive interface based on simple browsing on screens and typing texts using virtual keyboards. • Access the TvILO Viewer application; • Selecting the unit; • Select the class;

	• Browsing amongst media; • Feedback between T-Learner and Teacher via SMS; • Send the answers.
U	Users The T-Learner and the teacher who now have to analyze and evaluate the interaction of the T-learner
D	Devices DTV, STB, mobile phone, computer.
E	Environment The environment can change if there is a crossmedia transition. The main Environment is: Site with television, decoder and return channel with server access to the content of the media. That environment is usually the living room of the T-learner's house.

Table 4. Requirements of interaction of TvILO Viewer.

The aspect C refers to the teacher's need to monitor learners' participation. It is supposed to record the behavior of the learner while going through the class using iDTV application,. The format of this record must be standardized and sent to the teacher through the return channel to analysis and evaluation of the learner (see definitions on Table 5).

C	Content The Registration of the actions by the T-learner. Log of the frames visited by the T-learner. Register the answers of each question. The registration of launching of the surfed Medias, and of the choices made.
---	---

Table 5. Content requirement of TvILO Viewer.

2) Implementation of concrete interfaces

A concrete browsing interface on classes to Digital TV was implemented on Ginga (Ginga® is the middleware specification for the Brazilian Digital TV System. more: <http://www.ginga.org.br>). This interface will allow the crossmedia transition and temporal recording (log) of activities performed by the T-Learner for future analysis. one of the screens of the prototype showing a mathematical content can be seen in Figure 8. The T-learner has the option to know more about this content through a mobile device.

ENVIRONMENT

Figure 9 depicts the relation between applications and target users: the first user (a teacher) creates the classes (TvILOs), using TvILO Maker, and the second user (a T-learner) visualizes it via Digital TV, Web or Cell phone, using TvILO Viewer, as the start application. The necessity of using multiple devices was one of the motivations to choose UsiXML to map the requirements of interaction. The XML of the class is sent via a retuning channel and the application TVILO Viewer and UsiXML of the

Concrete Model of applied interfaces TvILO Viewer is sent via broadcast.

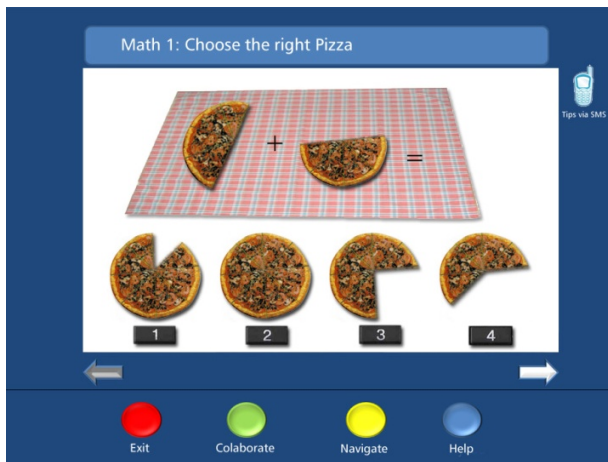


Figure 8. TvILO Viewer screen.

The third user (a technician) is responsible to put together this information into the transmission equipment (we use the Playout station for this purpose – see more http://www.eitv.com.br/index_eng.php). This transmitter sends to the Digital TV, situated in the lab, digital signals including the XML that contains the specifications of both the TvILO Maker application and the created contents.

The “C Aspect” of the TvILO Maker is set in order to recover the structure of a XML content and transmit it to the Digital TV. The TvILO Viewer application currently allows browsing on the content of Digital TV. The aspect C should be implemented through a group of web services that will allow the access and the inter-operability of STB with other platforms and communication media, such as the case of email servers, voice portals and SMS.

The applications TvILO Maker and TvILO Viewer are being currently implemented in Flex, NCL-Lua and Ruby on Rails. Flex is a highly productive, free open source framework for building and maintaining expressive web applications that deploy consistently on all major browsers, desktops, and operating systems. While Flex applications can be built using only the free open source framework - developers can use Adobe® Flex® Builder™ software to dramatically accelerate development - Lua has been used in many industrial applications (e.g., Adobe's Photoshop Lightroom), with an emphasis on embedded systems (e.g., the Ginga middleware for digital TV in Brazil) and games (e.g., World of Warcraft).

It was used a Web Service REST to the communication between TvILO Viewer and the TvILO Maker, and one XML defined as a protocol of this communication. By now, we didn't do any concrete implementation to TvILO Viewer regarding the crossmedia transition.

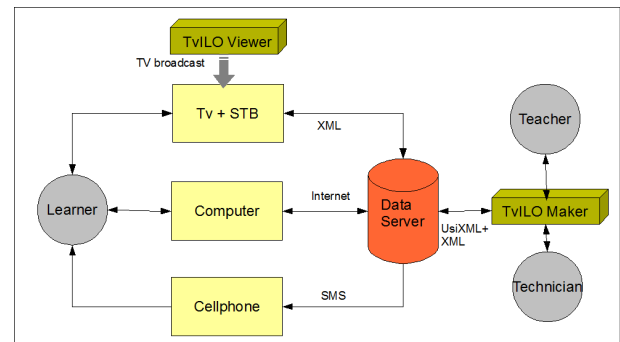


Figure 9. Relation between applications and target users

CASE STUDY AND PRELIMINARY RESULTS

We start the first course in April of 2010. We have chosen Fundamental School Mathematics for it. The method applied in the making of those classes was the Fedathi sequence [9]. The classes were made by teachers from Fundamental School and properly structured to teach math to students between 12 and 14 years old.

The making of this course also involved a simple lab experiment where this research is taking place. This experiment involved a teacher, 8 T-learners and a technician responsible for transmitting the course via digital signals transmitter, including a designer and the developer of the applications.

The technicians spent a week preparing the controlled environment. (such as recording cameras, questionnaires, etc.). This experiment happened in two days and The results are still being consolidated. We can highlight that development problems regarded mainly to the waste time to the iDTV application to start executing because it is taking time to put together Content XML and application specification when they arrive in STB. Usability problems regarded mainly to: i) the necessary adjustments of information on TV screen that can be different in function of the TV size, and ii) the difficulties T-learners had when performing a question using the remote control and a virtual keyboard.

CONCLUSION

This paper is important for its association between framework AMITUDE and UsiXML language. In this association adaptations took place. These adaptations expanded the model of interaction in order to allow the content data storage leaving more complete the domain models of an application. In this work, we showed how the structure of a TvILO can be modeled to make possible its broadcasting on iDTV, and how the structure of a TvILO can be modeled considering the existence of alternative media from iDTV. We hope the aspects of the use of the expanded framework, analyzed in the project phase, will enable the success of the future iDTV learning applications.

First, we hope the modeled applications “fit” users and their interests while we are developing an avant-garde educational application to use in multiple devices. The importance on developing new iDTV applications is directly linked to the evolution of technology in Brazil and in the world. When we develop learning applications we are enlarging the right to education to a greater part of the population.

Second, after the experiment we realized it is necessary to investigate new ways of interaction with devices. In STB case, in which we are limited to the remote control, we intend to look for devices of direct interaction such as movement detection. However we think crosmedia transition can provide users with alternative options for interaction.

ACKNOWLEDGMENTS

We thank UFC, CAPES, UNIFOR-MIA and Mil Tecnologia for the support.

REFERENCES

- Ahmad, A. and Ley, P. (2007). VILOs – video interactive learning objects. Retrieved March 5, 2010.
- Belda, F. Rolfsen. Um modelo estrutural de conteúdos educativos para televisão digital interativa, 2009. Tese de Doutorado
- Dybkaer, L., Bernsen, N.s O. Multimodal Usability. Springer, NY, 2009.
- Farias, M.. O telensino no cenário nacional e local das políticas de educação a distância. In: Farias, Isabel Sabino. Docência no telensino: saberes e práticas. Fortaleza: Secretaria da Cultura e Desporto do Governo do Estado do Ceará, 2000.
- Farias, I.; Nunes, J.; Cavalcante, Maria Marina Dias. Telensino: percursos e polémicas. Fortaleza: Edições Demócrito Rocha, UECE, 2001.
- Fernandes, J. Lemos, G.; Silveira, G.. Introdução à Televisão Digital Interativa: Arquitetura, Protocolos, Padrões e Práticas. Disponível em: <<http://www.cic.unb.br/docentes/jhcf/MyBooks/itvdi/texo/itvdi.pdf>>. Acessado em: 17 de abril de 2010.
- Klimčuk, Tomáš. Development environment for development of graphical user interface of iDTV. Master thesis. Computer Science and Engineering. 67f. May 2008.
- Lara Schibelsky , Godoy Piccolo , Maria Cecília , Calani Baranauskas, Understanding iDTV in a developing country and designing a T-gov application prototype, Proceedings of the 7th ACM conference on Designing interactive systems, p.379-385, February 25-27, 2008, Cape Town, South Africa
- LIMA, Ivoneide Pinheiro; SOUZA, Maria José Araújo; CUNHA, Francisco Gêvane Muniz; BORGES NETO, Hermínio. A Sequência de Fedathi como Proposta Metodológica no Ensino-Aprendizagem em Matemática Anais do XV EPENN - Encontro De Pesquisa Educacional Do Nordeste: Educação, Desenvolvimento Humano E Cidadania, vol. único, junho 2001, São Luís (MA), p594. (CDD: 370.981)
- Mendes, M. S., D.F. DIRCE - Design da Interação e levantamento de Requisitos com foco na Comunicação e Exploração de ideias: experiências de uso aplicadas em sistemas de criação de conteúdo para a televisão digital. Dissertação de Mestrado (Mestrado em Informática Aplicada) - Universidade de Fortaleza (UNIFOR). 176f. Fortaleza, 2009.
- Miyamaru F., Transições Entre Mídias: Arcabouço Tecnológico Para Aplicações De Mídia Cruzada. Dissertação de mestrado. USP. 2010.
- Paterno, F., Santoro, C. One Model, Many Interfaces. Proceedings Fourth International Conference on Computer-Aided Design of User Interfaces, pp. 143-154, Kluwer Academics Publishers, Valenciennes, 2002.
- LISBOA, R. P. ; GUSSI, Alcides Fernando ; NETO, H. B. . Telensino no Ceará: Uma Trajetória Histórica e Novas Perspectivas com a TV Digital. 2009. Congresso Internacional da Afirse - V Colóquio Nacional da AFIRSE / Secção Brasileira. Outubro 2009. Universidade Federal da Paraíba – João Pessoa(PB) Brazil.
- Vanderdonckt, J. and Gonzalez Calleros, J. UsiXML, a User Interface Model and Language Engineering approach. *W3C MBUI Incubator Group*. April, 2009.
- Vanderdonckt, J. Model-Driven Engineering of User Interfaces: Promises, Successes, and Failures. In *Proc. of 5th Annual Romanian Conf. on Human-Computer Interaction ROCHI'2008* (Iasi, 18-19 September 2008), S. Buraga, I. Juvina (Eds.), Matrix ROM, Bucarest, 2008, pp. 1-10.
- Vanderdonckt, J. A MDA-Compliant Environment for Developing User Interfaces of Information Systems. In *Proc. of 17th Conf. on Advanced Information Systems Engineering CAiSE'05* (Porto, 13-17 June 2005). Lecture Notes in Computer Science, Vol. 3520, Springer-Verlag, Berlin, 2005, pp. 16-31.
- SAMBA Project. System for Advanced interactive digital television and Mobile services in BrAzil. Available at: <http://www.ist-samba.eu/>. 2007.

Model-driven Generation of Ergonomic User Interfaces in Visualization Services

Guillaume Gronier, Sylvain Kubicki, Alain Vagner, Lou Schwartz, Eric Montecalvo, Thomas Altenburger
Public Research Centre Henri Tudor
29, av. J.F. Kennedy
L-1855 Luxembourg-Kirchberg
{first name.name}@tudor.lu

Gilles Halin
Research Centre in Architecture and Engineering (CRAI)
2, rue Bastien Lepage
F-54001 Nancy
gilles.halin@crai.archi.fr

ABSTRACT

The main goal of this work is to provide a cooperative IT services-based support for projects where heterogeneous actors teams from different organizations have to collaborate in order to realize a set of activities. This kind of “highly collaborative” environment exists in the construction sector, e.g. during building construction stage, and requires some flexibility and efficiency regarding work method and supporting tools. The research applies model-driven architecture to interface generation and takes into account people’s profile in order to customize generated interfaces and to improve their ergonomics. A case study is also presented in the visualization services design.

Keywords

User interfaces, adaptation, usability, visualization, architecture, visualization services, construction sector.

INTRODUCTION

In the framework of highly-collaborative environments, such as construction projects involving numerous practitioners, there is a clear need for interface plasticity. Plasticity is the capacity of interactive systems to adapt to the context of use while preserving the usability of the system [10]. Since the considered context is dynamic, the user interface should be able to adapt itself accordingly. An important issue of plasticity is the application of the concepts of Model Driven Engineering (MDE) to the user interface generation process. With MDE applied to plasticity, it should be possible to model the different models representing the application and its context in order to generate an efficient user interface.

So as to better understand this kind of process, the first priority is to identify the main characteristics of highly collaborative environments, then see how we can adapt an interface to the profile of a user, and to his context; the link between our models and the Service Oriented Architecture (SOA); the main languages for MDA-oriented user interface generation; and finally the main usability techniques that can be used in our context in order to certify the quality level of the generated interfaces.

In Architecture, Engineering and Construction (AEC) industry, our case study sector, each construction project

requires well-adapted software-based services and user interfaces to improve the efficiency of business collaborations as well as the quality of end-user experience. In such collaborative environments each user is involved in a wide network where he plays a role, characterized by his position in the activity organization, his skills and competencies, his contractual relationship to the other parties. Therefore the services and Human-Computer Interfaces (HCIs) offered to him have to be well adapted to these characteristics of use.

ERGONOMIC USER INTERFACE ADAPTATION

Personalizing User Interfaces

Nowadays, Human-Computer Interactions are part of our everyday life, from computers to mobile phones, and in the near future, they will be practically in every artifact. There is a proliferation of devices, acting in more and more different contexts and interacting with more and more users, thanks to the ever-going technical breakthroughs. This propagation of electronic devices tends to complicate interaction paradigms, that leads to loose the users’ attention.

In order to make the user focus on his task among a set of devices, new interaction paradigms came as a response, such as the “pervasive computing” (a.k.a. “ubiquitous computing” abbreviated “ubicom”). Introduced for the first time by Weiser [35], this interaction paradigm describes seamless interactions between the user and his surrounding electronic devices to an extent that the devices’ presence is omitted by the user. While “pervasive” and “ubiquitous” literally means “manifesting throughout everything”, by speaking of pervasiveness or ubiquity, Weiser is also referring to the seamless aspect of interactions.

Context-Awareness

A way to achieve pervasive interactions is to tailor systems that are sensitive to the interaction’s context. The system then reacts accordingly and proposes services adapted to the user’s task in a transparent manner. So called systems are known as “context-aware” systems and were firstly mentioned by Schilit and Theimer [31] while the authorship of the first application of such systems is commonly accorded to Want, Hopper, Falcao and Gib-

bons [34], even though their application is limited to a location-awareness. Context-aware systems rely on the capture of contextual information (e.g. user's location and surroundings, time of the day, etc.) and on the system reactions depending on these information to better fit the user's needs or enhance its effectiveness. Working with contexts enables to gather richer information that will unlock new processing possibilities without (or with less) user intervention.

Despite Schilit and Theimer [31] described a context-aware application as being able to "adapt itself according to its location of use, the collection of nearby people and objects, as well as changes to those objects over time", researches were mainly focusing on location as the only relevant contextual element [1] [7]. Besides location-aware application, most researches were domain specific, limiting the usage of contextual information. It is only by the years 2000 that appeared more elaborated definitions of what context-awareness is. Schmidt, Beigl and Gellersen [32] are among the first to have pointed out the limitation of location-aware system. Dey's definition [11] is the one that united the domain's actors and is stated as follow:

"A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task."

Such systems are characterized by the acquisition of contextual data (or sensing), the processing of these data and the applicative reaction to the result of the processing, considering the reaction to be relevant to the user's task [2].

According to Bolchini, Curino, Quintarelli, Schreiber and Tanca [5] or Hong, Suh and Kim [17], one of the common usages of context awareness is the adaptation of user interfaces, i.e. to present interfaces that are well suited to the user's task in his current context in order to enhance usability (e.g. a context-aware mobile phone could present only productivity features to the user if it identifies the user's location as being "at work" or reject any incoming call if the user is attending a meeting).

User Interface Adaptation

Interfaces adaptability is not a new subject. During the last years, a lot of studies have been carried out on the subject, but they were mostly oriented on Adaptive Hypermedia Systems (AHS). Brusilovsky works [6] focused on what can be adapted (content and navigation) and on adaptation methods and techniques. Since, these works have evolved and multiplied (especially applied to web sites). Besides the generic context, interface adaptation can be executed at different levels, namely at the user and device ones.

Interface adaptation according to the device

This kind of adaptation can be considered as one of the simplest but can be tricky since the interaction paradigms

and modes often vary from one device platform to another.

For example, numerous interfaces have to be specifically redesigned to take into account touch interactions on the new batch of touch devices, since they are not completely compatible with mouse driven inputs (*mouseover* events are for example unavailable on touch devices). On the Web, several languages and technologies enable us to adapt user interfaces to the device or user-agent. CC/PP¹⁰ profiles contain some detailed descriptions of the characteristics of a device such as screen size, input devices, etc. These profiles can be manipulated at the web interface behavior layer (i.e. JavaScript) through the use of DCCI¹¹ interfaces. Another opportunity for graphical adaptation will be based on style layer, using CSS3¹² module "media queries"¹³, which will allow adapting layout and presentation to the technical characteristics of the device.

Interface adaptation according to the user profile

One important source of interface adaptation perspectives consists in the study and modeling of the user profiles. It includes his identity information, his preferences and interests as well as his capacities. The user profile can also be useful to define user disability, as an example a visual interface will be adapted into an audio interface for a blind; or some set of colors will be never used for a color-blind. Adaptation work will focus on the different interaction channels now available on mobile devices (e.g. entry: keyboard or tactile screen, vocal command, body movements).

Adaptation to user profile implies modeling the user and profiling it, explicitly by asking for his personal data or preferences, or implicitly. As it relies on an analysis of the user behavior, implicit profiling is more efficient in the sense it is not biased by the inherent uncertainties of human assumptions.

Usability

User-centered design and quality of generated HCI

Usability oriented development processes offer a considerable advantage to both user's accessibility and from a strictly economic perspective. Many authors have mentioned this in their publications [27]. Although this approach is frequently used in the marketing domain, many

¹⁰CC/PP, Composite Capabilities/Preference Profiles: Structure and Vocabularies 1.0, W3C Recommendation, 15 Jan. 2004, available on the website: <http://www.w3.org/TR/2004/REC-CCPP-struct-vocab-20040115/>

¹¹DCCI, Delivery Context: Client Interfaces 1.0 – Accessing Static and Dynamic Delivery Context Properties, W3C Candidate Recommendation, 21 Dec. 2007, available on the website: <http://www.w3.org/TR/DPF/>

¹²CSS3, Introduction to CSS3, available on the website: <http://www.w3.org/TR/css3-roadmap/>

¹³Media Queries, available on the website: <http://www.w3.org/TR/css3-mediaqueries/>

companies still rely on development processes that have not been adapted to a true user-centered approach. Some studies even focused on a metric approach to the “usability gain” and found cost / benefit ratios oscillating between 1:10 and 1:100 [13].

The ISO 13407 norm [18] (“Human-centered design processes for interactive systems”) presents an approach to interactive system development that focuses specifically on making systems usable. It underlines that the application of human factors and ergonomics to interactive systems design enhances effectiveness and efficiency, improves human working conditions, and counteracts possible adverse of use on human health, safety and performance. Then applying usability to the design of systems involves taking account of human capabilities, skills, limitations and needs.

However as a majority of development-oriented companies seem to remain reluctant to adopt real user-centered development processes and as development time frames tend to get continuously tighter, it is increasingly more important to analyze the degree to which a user-centered approach could be directly integrated into automated development cycles. This perspective seems particularly attractive because it can potentially contribute to conciliating the time constraints in development and the user requirements.

Usability criteria

Both the conceptual and methodological issues to deal with relate to the “usability criteria”. The criteria are the indicators the usability models are built on; they are used in an operational context to evaluate to what extent the development or the finished IT product is in line with usability requirements. In fact, as shown in our earlier research works [20] the literature offers an important variety of conceptual frameworks around the concept of usability. As a result, the usability criteria appear to be very numerous, diverse and the models seldom agree on the proposed criteria. Differences are frequent and pertain to:

- the number of criteria per model;
- the “abstraction level” of the criteria;
- the ability to operationalize the criteria;
- the inter-criteria orthogonality.

In addition to a very large inter-model diversity, we also showed [20] that the conceptual frameworks in the usability domain suffer from an additional issue: some authors define their models as pertaining only to development situations; other authors would like their model to be applied only to usability evaluations; while a third category do not take into account this potential difference but nevertheless don’t fit both situations. As a result, the overarching conceptual framework lacks consensus and there seem to be no models that deal with all these requirements.

To finish, note that the involvement of users in the development process provides a valuable source of knowledge

about the context of use, the tasks, and how users are likely to work with the future system [18]. This contributes to the development and understanding of the models for profiling and personalization described above.

CASE STUDY: VISUALIZATION SERVICES DESIGN FOR HIGHLY COLLABORATIVE ENVIRONMENTS

Highly collaborative business environments are characteristic of numerous business fields and projects. If collaboration exists inside firms and enterprises, we consider it particularly challenging when collaborative projects involve several firms, for short durations, in unpredictable contexts. In such organizations, business services as well as User Interfaces (i.e. visualization services) face the challenge of rapidly adapting to dynamic business environments.

Business actors’ requirements in highly collaborative environments

The concept of adaptation of User Interfaces has also to take into account the adaptation to the requirements of business actors regarding visualization, also called usages. That is the aim of the case study presented here.

The usage-centered engineering has been introduced by Constantine and Lookwood [8] to develop software and HCIs that support all the tasks that users have to accomplish. This method suggests a systematic process using abstract models to guide user-interface designers. The usage-centered design is proposed as an alternative to user-centered design, which relies on three main techniques: user studies (to identify the users’ need), rapid prototyping (to get user feedback), and usability testing (to identify usability problems [9]). The difference between usage-centered design and user-centered design [9] is that the focus is not users but usage, i.e. the tasks intended by users and how these are accomplished.

By extension, we consider this approach really powerful to express the requirements based on the real business tasks of actors in a particular targeted domain. In collaborative environments, these business tasks are often closed to the role of actors in an organization, but also to the business services (i.e. IT or non-IT services) supporting a project’s collaboration. Role-specific usages in Architecture, Engineering and Construction (A.E.C.) projects have been defined in previous works [23] and served as inputs for visualization interfaces design.

The A.E.C. Collaborative Context

The Architecture, Engineering and Construction sector (A.E.C.) is characteristic of such highly collaborative environments. Construction projects involve numerous practitioners, for short durations (project duration), in various contractual contexts (private projects, public-private partnerships etc.). Business practitioners (e.g. public owners, engineers and architects, contractors...) work together and carry out their own methods and models. Therefore in such environments groupware services and the associated User Interfaces have to fit the specific

requirements related to 1) the collaborative situation, and 2) the specialized roles involved. Especially we consider as particularly challenging the adaptation of User Interfaces to the particular requirements of the various roles using it.

Modeling Collaboration activities and support services

A model-based approach has been used in parallel works in order to:

- Represent collaborative situations (i.e. building projects). A metamodel of collaborative activities has been designed [22], and enables instantiating specific models of construction projects;
- Describe services, and especially the ones supporting collaboration (groupware services). A metamodel of service is currently under design and aims at representing services through different viewpoints (Business, service solution and technical views).

Adaptable Visualization Services

Visualization services implemented in Human-Computer Interfaces of business services have to fit usage of actors who have specific practices according to their role in a collaborative activity. Usages differ from traditional user profiles in context-aware computing by increasing their acceptance with:

- Organizational and operational roles of a project's actor in a particular cooperative situation;
- Particular requirements generated by the available business services in the collective project.

Therefore User Interfaces could be designed according to the usages. They can be identified as "Adaptable Visualization Services" which could be chosen, integrated and used by the actors to perform their activity inside a highly collaborative project [23]. Moreover the approach of AVS has been introduced previously by Dymáček, Hocková and Kintr [14] and aimed at developing visualization services able to be re-used in various contexts, and with different data types.

Design approach

In such environments the "mapping" between 1) usage, 2) domain characteristics and 3) Business Service attributes on the one side, and 4) Visualization Service attributes on the other side is essential. The value of an AVS proposition highly relies on its alignment to the user requirements (i.e. the usage). Therefore, focusing on AVS design, the main challenge is to map the concepts of usages to the ones of Adaptable Visualization Services such as shown in Figure 1.

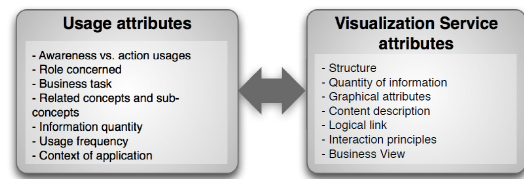


Figure 1. Mapping between usage and AVS attributes.

Case study: AVS design

The case studied here consists of a first experiment aiming at designing an AVS's platform on the basis of usages that we identified with practitioners. This research work helped us to consolidate a design process and led to the development of four AVS integrated in a prototype platform [37].

Usages related to the role of "architect" have been identified. The approach was based on existing practices related to document management in construction projects. We distinguished between two types of usages: the ones related to the understanding of the state of the cooperative context (i.e. awareness usages) and the others related to the specific actions the role has to perform (i.e. action usages). These usages have then been described according to a defined metamodel. A first set of 37 architect-specific usages has been identified. In a second stage we have set-up an experiment protocol in order to validate their usefulness with practitioners. Interviews have been carried out aiming at verifying qualitative aspects of usages (are they really useful) and quantitative aspects (among the 37 usages, what are the preferred ones?).

Once some visualization techniques were described, we tried to find a logical manner to associate them with usages. Attributes of usages were compared to attributes of visualization services. Two usages have been selected, and for each one, two visualization services have been selected.

Two usages have been chosen and four related AVS have been implemented in a prototype platform called E.V.A. (Experimental Visualization Application). Figure 2 shows the E.V.A. interface. A basic selection of a usage and a particular visualization enable the user to test it.

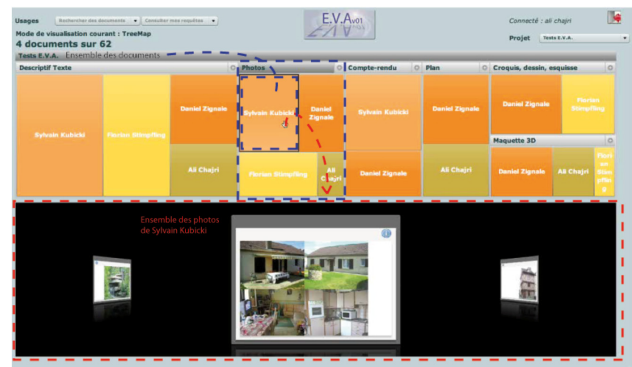


Figure 2. E.V.A. user interface.

The final part of this AVS design example is related to the validation of the proposition. As usages were previously validated with practitioners (see above) we had a first assessment of their utility. A second experiment stage led us to test the AVS, once implemented, with users. The process relied on testing tasks performed by users and a final questionnaire/discussion to gather qualitative feedback.

TOWARDS A GENERATION METHOD

This case study of AVS design in highly collaborative environment (here the A.E.C. industry projects) shows the importance of aligning User Interfaces to the requirements of users in a specific context (comprising both a targeted domain and possibly existing business services). Beyond AVS design based on usages modeling, one can recognize the importance of User Interface (UI) personalization. The following parts describe a proposal for a generic method enabling the generation of personalized UI and based on models.

A generic method: the Genius method

The method proposed here is based on an iterative process made from 4 successive major steps. This iterative process is inspired from classical methods such as "User-centered design" [18] and agile project management, since the involvement of the users is a key concept. The software framework aims at streamlining the collaboration between users and developers and automate the feedback channel.

The first step is related to the modeling of the business domain in which we wish to support collaboration through services and HCIs. By "business domain" we mean any relation to the tasks of the operator and the context in which these tasks occur. Thus, four main elements are considered:

- The project (tasks, goals, duration, planning...) [11] and the aspects related to the characterization of collective work [26] [28]. This first element is named the "*collaboration model*". It is close to the meta-model of collaborative activities presented in the case study;
- The organization, as a complex social construction, which has an impact on the IT services and HCIs. This element is named the "*organizational model*";
- The services designed for the domain. In this point we consider more precisely the e-services specifically composed to answer to the requirements of a specific collaborative situation (e.g. a construction project). The related model is called the "*service model*" (cf. case study);
- The user profile [2] [30] described in a "*user profile model*".

The second step consists in automatically generating a Human-Computer Interface based on the modeling of the various issues identified during the step 1. The method consists in modeling and implementing the transfor-

mations between models necessary to perform. The result of this stage is characterized by an adaptive interface [9].

In our approach this step is enhanced through ergonomic criteria [3] [19] from which the interface is generated. This is one of the fundamental points of this research project. Indeed, this step involves the automatic generation of user interface complying with ergonomic heuristics that can be referenced in the literature. Note also that we consider the ergonomics of human-machine interfaces such as the association of various factors like aesthetics [16] [33], acceptance [36], usability [24], perceived usefulness, and so on. This step thus gives a first set of inputs to the question whether heuristic criteria can easily be adapted in order to guarantee an improved level of usability. It will give us a better understanding of what could be the best way to operationalize these criteria.

The third step of our method allows multiple users to use and test the generated user interfaces. This is an important step in our research because it collects data from a field study. In our case, the field study will be the Architecture, Engineering and Construction sector, as previous UI design process are available (see case study). However the method could be applied to many other cases characterized by highly collaborative activities. The generated application should preferably be used in "real world" conditions in order not to disturb the UI assessment and to foster adoption.

Finally, the fourth stage of our method is consists in a feedback that improves the understanding of models developed in step 1. These feedbacks will enable collecting information on the use of the interface created previously. They will provide valuable information useful in designing future generations, adding to their plasticity. The key part of this stage will be the interpretation of these feedbacks and the mapping of the underlying concepts to the four main models of our framework. Our method is summarized in Figure 3.

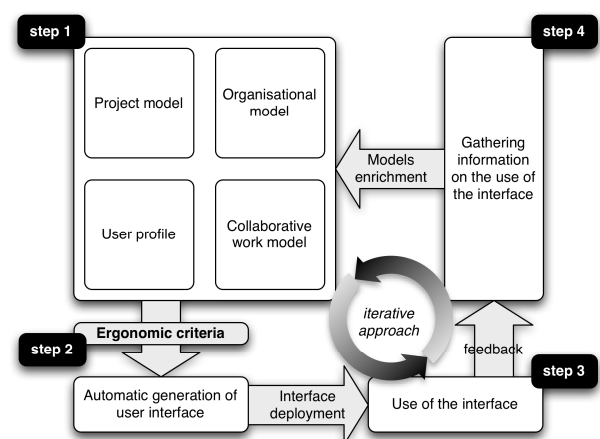


Figure 3. The Genius method.

Integrating UsiXML in the Genius method

As previously described, to improve the usability of auto-generated UIs, the Genius method considers ergonomic criteria (based on heuristics that can be referenced in the literature) within the model-driven user interface design. Moreover, it defines an iterative approach wherein the first step is focused on the modeling of business domain that impact the automatic generation of UIs. This business domain considers four main elements: the project, the organization, the collective work and the user profile. A part of the modeling of these elements can take advantage of innovative previous works such as UsiXML [37] [38].

UsiXML¹⁴ is a XML-based mark-up language that describes a UI for multiple contexts of use such as character, graphical, auditory or multimodal interfaces. Currently UsiXML only supports interaction device independence, computing platform independence and interaction modality independence¹⁵. One main goal of user interface description languages as UsiXML, is to handle the complexity, the heterogeneity of final platforms and the different contexts of use. To achieve this, UsiXML provides a high level of abstraction to design UIs, by using a set of models involved in the UI conception. These models are used to provide UI designers with a way, models and tools to design interfaces in order to auto generate the final UI according to the MDE approach [15] [39]. This method aims at separating the design tasks from the development ones. A relationship between the required models in the Genius method and the available models defined in UsiXML can be established. The Domain Model, Task Model, Context Model, Resource Model, Abstract UI Model and Concrete UI Model are identified as potential interesting models to enrich the modeling of business domain defined into Genius. The links between each of these models is handled by classic MDE transformations (e.g. Mapping and Transformation model defined in UsiXML). During the modeling of software and the transformation of models, the Genius method aims at identifying how to detect the potential ergonomic issues to improve the usability of final generated UI.

The scope of UsiXML models is wide but does not consider all aspect of a user interface. In order to help UI designers to reach a better usability level, the Genius project attempts to identify where and how the modeling of ergonomic heuristics and the validation process can take place into the MDE approach. Many UsiXML related works [4] already propose different approaches which can contribute to improve the integration of ergonomics criterion in the Genius method. This is particularly innovative because many studies show the difficulty of systematizing the ergonomic design of interfaces. Moreover, the Genius project considers the UI design within an iterative ap-

proach, including user's feedbacks in the final step of iteration. The aim is to provide a basis for statistical analysis related to a number of performance indicators to be identified, and next enhance or adapt the initial modeling of software according to the feedbacks of users and the ergonomic recommendations.

The outcome of the next version of UsiXML will enhance the modeling language through the addition of versatile context-driven capabilities based on $\mu 7$ concept¹⁶. Another high priority in the Genius project is to identify the main characteristics of highly collaborative environments, and see how we can adapt the interfaces to the profile of a user and to his context. So as to better understand this kind of process, by initially considering a large set of contexts, we can extend the research area to determine more characteristics involved in highly collaborative environments.

CONCLUSION

We presented here the current state of our work on the model-driven generation of ergonomic user interfaces, based on a case study about visualizations services design. We plan to generate usable and context aware user interfaces and refine them through an iterative and semi-automatic process. This type of process could be of interest for the AEC collaborative context, in which it is necessary to design visualization services adapted to their users and to the specificities of each collaborative situation. Our approach is based on model-driven engineering, and we make an extensive use of several models such as project (or domain) model, organizational model, user profile model and collaborative work model. All these models have to be productive and lead us to UI models that will be implemented through UsiXML as the most versatile language for context-aware and model-driven UI design. The usability question will mainly be addressed through two ways: a priori we will constrain our model to the validation of ergonomic criteria formalized as rules, and a posteriori with the study of user behaviors and the adaptation of our models.

Our main objective is now to refine our method and metamodels and to build tools assisting the work of UI designers and business experts in the design of the solution. Then we plan to experiment our main assumptions in real cases of UI design for the AEC sector.

ACKNOWLEDGMENTS

The research in this paper was carried out within the project GENIUS, funded by the "Fonds National de la Recherche" (Luxembourg). The authors would also like to thank Vincent Koenig (University of Luxembourg) for his contribution in the research project.

¹⁴ www.usixml.org

¹⁵ UsiXML Specification v1.8

¹⁶ ITEA2. Project profile available on the website: www.itea2.org/public/project_leaflets/UsiXML_profile_oct-09.pdf

REFERENCES

1. Abowd, G.D., Atkeson, C.G., Hong, J., Long, S., Kooper, R., and Pinkerton, M. Cyberguide: A mobile context-aware tour guide. *Wireless Networks* 3, 5 (2004), 421-433.
2. Baldauf, M., Dustdar, S., and Rosenberg, F. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing* 2, 4 (2007), 263-277.
3. Bastien, J.M.C., Scapin, D.L., and Leulier, C. The ergonomic criteria and the ISO/DIS 9241-10 dialogue principles: a pilot comparison in an evaluation task. *Interacting with Computers* 11, 3 (1999), 299-322.
4. Beirekdar, A., Vanderdonckt, J., and Noirhomme-Fraiture, M. A Framework and a Language for Usability Automatic Evaluation of Web Sites by Static Analysis of HTML Source Code. In *Proc. of 4th Int. Conf. on Computer-Aided Design of User Interfaces CADUI'2002* (Valenciennes, 15-17 May 2002). Kluwer Academics Pub., Dordrecht, 2002, pp. 337-348.
5. Bolchini, C., Curino, C.A., Quintarelli, E., Schreiber, F. A., and Tanca, L. A data-oriented survey of context models, *ACM SIGMOD Record* 36, 4 (2007), 19-26.
6. Brusilovsky, P. Adaptive hypermedia: an attempt to analyse and generalize. *International Centre for Scientific and Technical Information, available on the web-site: <http://www.wis.win.tue.nl/ah94/Brusilovsky.html>*
7. Cheverst, K., Davies, N., Mitchell, K., Friday, A. and Efstratiou, C. Developing a context-aware electronic tourist guide: some issues and experiences. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2000), 17-24.
8. Constantine, L.L., and Lockwood, L.A. *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Inc, Addison-Wesley Educational Publishers, 1999.
9. Constantine, L., and Lockwood, L.A. Usage-centered engineering for Web applications. *IEEE Software* (2002).
10. Coutaz, J., and Calvary, G. *HCI and Software Engineering: Designing for User Interface Plasticity*. In *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies, and Emerging Applications*, 2008, 1107-1125.
11. Dey, A. K. Understanding and Using Context. *Personal Ubiquitous Computing*. 5, 1 (2001), 4-7.
12. Diaper, D., and Stanton, N. The handbook of task analysis for human-computer interaction. *Lawrence Erlbaum Associates Inc*, 2003.
13. Donahue, G.M., Weinschenk, S., and Nowicki, J. Usability is good business (1999), available on the web-site: www.compuware.com.
14. Dymáček, T., Hocová, P., and Kintr, M. Adaptable visualization service: through uniformity towards sustainability. *First International Workshop on Model Driven Interoperability for Sustainable Information Systems (MDISIS'08)*, Montpellier, France, June 16, 2008 (2008).
15. Guerrero García, J., Lemaigre, Ch., Vanderdonckt, J., and González Calleros, J.M. Model-Driven Engineering of Workflow User Interfaces. In *Proc. of 7th Int. Conf. on Computer-Aided Design of User Interfaces CADUI'2008* (Albacete, 11-13 June 2008), Springer, Berlin, 2009, pp. 9-22.
16. Hasan, B., and Ahmed, M.U. Effects of interface style on user perceptions and behavioral intention to use computer systems. *Computers in Human Behavior* 23, 6 (2007), 3025-3037.
17. Hong, J.Y., Suh, E.H., and Kim, S.J. Context-aware systems: A literature review and classification. *Expert Systems with Applications* 36, 4 (2009), 8509-8522.
18. ISO 13407. *Human-centred design processes for interactive systems. International Standards for Business, Government and Society*.
19. Jacko, J. *The human-computer interaction handbook*. Lawrence Erlbaum Associates Inc, 2002.
20. Koenig, V. Contribution à l'étude de l'utilisabilité dans le contexte des systèmes d'information à usage professionnel : conception d'un laboratoire d'utilisabilité et applications. Thèse de doctorat en sciences psychologiques, Université Libre de Bruxelles, Belgique, 2006.
21. Koenig, V., and van de Leemput, C. U-Lab: an out-of-the-box solution for any usability problem? A presentation on methodological and practical issues. *Annual Meeting of the Belgian Psychological Society*, Ronéo, Ghent (2005).
22. Kubicki, S., Bignon, J.C., Halin, G., and Humbert, P. Assistance to building construction coordination. Towards a multi-view cooperative platform. *ITcon Electronic Journal of Information Technology in Construction* 11 (2006), 565-586.
23. Kubicki, S., and Halin, G. Usage-centered design of adaptable visualization services. Application to cooperation support services system in the AEC sector. *IESS 1.0. First International Conference on Exploring Services Sciences*, Geneva, Switzerland, 17-19 February 2010, Lecture Notes in Business Information Processing n° 53, (2010).
24. Malhotra, Y., and Galletta, D.F. Building systems that users want to use. *Communications of the ACM* 47, 12 (2004), 88-94.
25. Mayhew, D.J., and Bias, R.G. Cost-justifying web usability, in Ratner J. (Ed), *Human factors and web development. Second Edition* (pp. 63-87), USA: Lawrence Erlbaum Associates, 2003.

26. Newlands, A., Anderson, A.H., and Mullin, J. Adapting communicative strategies to computer-mediated communication: an analysis of task performance and dialogue structure. *Applied Cognitive Psychology* 17 (2003), 325-348.
27. Nielsen, J.. *Usability Engineering*. Academic Press, San Diego, CA, 1993.
28. Pinelle, D., Gutwin, C., and Greenberg, S. Task Analysis for groupware usability evaluation: modelling shared-workspace tasks with the mechanics of collaboration. *ACM Transactions on Human Computer Interaction* 10, 4 (2003), 281-311.
29. Raijmakers, B. Usability-Tests ist ein Mittel, kein Ziel, in Beier M. and von Gizycki V. (Eds.). *Usability, 3th edition*, (pp.129-157), Berlin and Heidelberg: Springer, 2002.
30. Rich, E. User Modelling via Stereotypes. *Cognitive Science* 3, 4 (1979), 329-354.
31. Schilit, B., and Theimer, M. Disseminating active map information to mobile hosts. *IEEE* 8, 5 (1994), 22-32.
32. Schmidt, A., Beigl, M., and Gellersen, H.W. There is more to context than location. *Computers & Graphics* 23, 6 (1999), 893-901.
33. Tractinsky, N., Cokhavi, A., Kirschenbaum, M., and Sharfi, T. Evaluating the consistency of immediate aesthetic perceptions of web pages. *International Journal of Man-Machine Studies* 64, 11 (2006), 1071-1083.
34. Want, R., Hopper, A., Falcao, V., and Gibbons, J. The active badge location system. *ACM Transactions on Information Systems* 10, 1 (1992), 91-102.
35. Weiser, M. The computer for the 21st Century. *ACM SIGMOBILE Mobile Computing and Communications Review* 3, 3 (1999), 1559-1562.
36. Wu, J.H., Chen, Y.C., and Lin, L.M. Empirical evaluation of the revised end user computing acceptance model. *Computers in Human Behavior* 23, 1 (2007), 162-174.
47. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., and Lopez, V. UsiXML: a Language Supporting Multi-Path Development of User Interfaces. In *Proc. of 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems EHCI-DSVIS'2004* (Hamburg, July 11-13, 2004). Lecture Notes in Computer Science, Vol. 3425, Springer-Verlag, Berlin, 2005, pp. 200-220.
37. Vanderdonckt, J. A MDA-Compliant Environment for Developing User Interfaces of Information Systems. In *Proc. of 17th Conf. on Advanced Information Systems Engineering CAiSE'05 (Porto, 13-17 June 2005)*, O. Pastor & J. Falcão e Cunha (eds.), Lecture Notes in Computer Science, Vol. 3520, Springer-Verlag, Berlin, 2005, pp. 16-31.
38. Vanderdonckt, J., Model-Driven Engineering of User Interfaces: Promises, Successes, and Failures. *Proc. of 5th Annual Romanian Conf. on Human-Computer Interaction ROCHI'2008* (Iasi, 18-19 September 2008), S. Buraga, I. Juvina (Eds.), Matrix ROM, Bucarest, 2008, pp. 1-10. ISSN 1843-4460.

Service-Oriented Architecture for Supporting Collaborative User Interface Development

Hildeberto Mendonça¹, Kênia Sousa², Jean Vanderdonckt²

Université catholique de Louvain

¹Place du Levant, 1 - ²Place de Doyens, 1 - 1348 Louvain-La-Neuve, Belgium
{hildeberto.mendonca, kenia.sousa, jean.vanderdonckt}@uclouvain.be

ABSTRACT

Professionals working in organizations that conduct any user interface development life cycle are more and more involved in a collaborative setup where competences and resources are distributed in time and space. In order to support this shift of practice, a service-oriented architecture is defined and developed according to principles of model management. In this paradigm, user interaction development is decomposed into activities, which could be supported by model management operations. These operations are in turn converted into services, developed according to the service-oriented architecture. A distributed user interaction development life cycle consequently involves the following steps: a method engineer defines the activities to be conducted for the user interface of a project, the method definition is imported in the software architecture to enact the method by assigning responsibilities to team members, and these members then perform their responsibilities through the services corresponding to the operations. This paper also presents a taxonomy of services for supporting the user interface development life cycle that is unique.

Categories and Subject Descriptors

D.2.2 [Software Engineering]: Design Tools and Techniques – *Computer-aided software engineering (CASE), User interfaces* H.5.2 [Information Interfaces and Presentation]: User Interfaces – *Graphical user interfaces (GUI), User Centered Design*. I.3.6 [Methodology and Techniques]: Device Independence, Language, Standards. H.5.3 [Information Interfaces and Presentation]: Group and Organization Interfaces - *Collaborative computing, Computer-supported cooperative work*.

General Terms

Design, Human Factors, Standardization, Languages.

Keywords

User interaction design method, service-oriented architecture, meta-models, business process modeling, model management.

INTRODUCTION

Service-Oriented Architecture (SOA) has been successfully applied to a wide variety of domains of human activity, inside and outside computer science, such as, but not

limited to: database management, distributed computing, cloud computing. One discipline of computer science that has received little or no attention with respect to SOA is Human-Computer Interaction (HCI), which is the discipline that is aimed at defining and applying a user interface development life cycle for any project. This has been for a while primarily because HCI has not evolved as fast as other disciplines in integrating SOA concepts and experience and because integrating Software Engineering (SE) techniques in HCI has been challenging for many years. This paper is aimed at addressing the need of supporting a structured user interface development life cycle using SOA.

Although still evolving, one reference framework has today received some consensus throughout the HCI community: the Cameleon Reference Framework (CRF) [5] proposes a set of models, compatible with MDA [16,24], that provide the necessary support for addressing the current challenges posed by User Interface (UI) development. This framework consists of 5 models distributed in 4 levels of abstractions that are intended to express the UI development life cycle. The CRF proposal achieves the necessary support for different platforms, devices and contexts of use and a considerable effort is running to provide computational and methodological applicability. Different implementations exist today that exhibit various levels of compliance with the CRF.

One of the bases of this work was to define a language to represent all aspects of the CRF in a computational form. The language UsiXML [15] was created as a XML extension to describe UIs for multiple contexts of use and it motivated the implementation of a set of tools to manipulate the language [25]. Every day, the integration between all these tools becomes more critical to guarantee the life cycle's completeness and coherence. At the same time, all possible operations in a life cycle could not be provided by tools because there are some dimensions to be considered, as management, control, integration with external environments, and so on.

We are now proposing one approach that could standardize the integration between tools and make the application of methods possible to support the entire life cycle, including the integration of these methods in a pre-existent and deployed development process. The basic idea is to provide all model operations as services that could be re-

used by different tools and applications. The UsiXML language will provide the necessary support to represent models in a structured and reasonable form.

The service's concept is the same used by SOA that considers it as a software component, but with special ability to improve the software composition and distribution. A service is published in the World Wide Web (WEB) infrastructure and can be invoked by a software to execute part of its logic. Applying it in our problem, a service can provide a set of potential benefits such as:

- Reduce the complexity of the current tools and applications available and improve the architecture of the future implementations;
- Enable the collaborative work locally, distributed (e.g. intranet), and remotely distributed (e.g. internet);
- Avoid the duplication of implementations and value the variety of implementation strategies;
- The number of services available is directly proportional to the number of possibilities to create new applications and methods; and
- Totally based in widely accepted patterns, managed by W3C, OMG, OASIS, etc.

The research aims to provide a consistent solution to allow a collaborative work in the UI design and to bring a definitive contribution to facilitate the application of HCI practices in real world organizations. This paper intends to present the foundations and the structure of the research, talking about models, methods and technology, but every time thinking about how it can contribute to the software's end-users.

This paper is organized as follows: Section 2 presents related works; Section 3 shows the use of services for model management; Section 4 presents the method specification; Section 5 details the deployment of services to support communication; Section 6 illustrates the method execution with a case study; Section 7 concludes this work by presenting the contribution and future work.

RELATED WORK

In order to achieve competitive results and still address constant organizational changes, SOA has been widely applied to bring agility in the business process definition and improve the communication between organizations and even between distributed people of the same organization.

A model proposed by Colombo et al. [6] shows that web services, the key technology to concretize SOA, are an effective solution to let software systems, developed by different organizations and spread across the world, interoperate. This model has support to i) *Agent-actors*: identification of stakeholders and roles; ii) *Core Service*: a particular concrete resource which is offered by a Software

System; iii) *Service Description*: a syntax description about the service interface to show the potential offered; iv) *Service Discovery*: a process to discover new services in the network to become available for services composers; v) *Service Composition*: the service capability to execute in cooperation with other services in a same transaction; vi) *Service Publication*: to make services available in the network, exposing their service descriptions to be recognized by other applications; and vii) *Service Monitoring*: production of statistics data to build quality metrics. All these concepts allow the method automation, creating services to manage artifacts, requirements, tests and so forth, putting all the services in a logic sequence and changing it whenever necessary or creating different versions of the method, considering the size or the complexity of projects.

One collaborative approach to improve the user interaction design process is not a strongly explored subject as it really deserves. Actually, there exists a real desire to increase the return of investment on the HCI adoption as we have noticed in some works [3,7,10,22], which use different approaches like reducing the gap between Software Engineering and Human-Computer Interaction or creating new techniques to deliver a well-accepted user interface.

We are not trying to create a new form to reduce the gap between SE and HCI or a new technique. But, with SOA, it is possible to reduce the mentioned gap and use new techniques throughout the activities. It is fundamental to use models, represented in a structured language, such as UsiXML, to improve the performance by model transformations, avoiding to create models since the beginning.

MANTRA (Model-based eNginneering of multiple interfaces with TRAnsformations) [4] is a model-driven approach for the development of multiple UIs. MANTRA is structured by abstraction levels similar to the CRF [5]. The model transformations, such as adapting the AUI according to the requirements and transforming the adapted AUI into several CUIs, are described in ATL (Atlas Transforming Language) [12], a hybrid model transformation language that allows both declarative and imperative constructs to be used in transformation definitions. The declarative style of transformation is based on specifying relations between source and target patterns. The imperative part in the transformation language is explicitly encoded by the developer. In contrast with our approach, MANTRA uses web services to implement the final application, not as a strategy to improve the method applicability and adaptation to the dynamism of the organization.

Wolff et al. [27] propose an approach and a tool to support transformations between models supported by patterns. The transformation between the dialog graph (specification of views, their association to tasks, and the transitions between tasks and views) and a platform-independent UI model (PIM) is produced by mapping

views to windows and elements of views to buttons. The connections with the task model are kept, which facilitates tracing actions on elements back to tasks and the simulation of dialog graphs. The transformations are not fully automated, but they are supported by humans using interactive tools. For instance, the transformation from Platform-Independent Model (PIM) to Platform-Specific Model (PSM) is achieved by the designer replacing (via “drag & drop”) PIM elements by a pre-designed component, that is, UI patterns.

Similarly, Sinnig *et al.* [23] use patterns during the application of a model-based development methodology with tool support based on XML-representation of patterns for each level, namely dialog, presentation and layout patterns.

Even though the use of patterns during User Interface Development (UID) promotes re-use, standardization, and efficiency to the transformation of models; model transformation logic are coded inside the tools, thus making it more difficult to reuse implementation strategies.

These approaches have tools to support UID methods, but the descriptions of these tools do not mention any technology that supports collaborative work. Thus, these tools support the creation of models and automated transformations in some level, but professionals cannot use them to communicate and share their work.

These approaches follow a formalized method, but their supporting tools do not provide facilities to change the sequence of the method activities, thus restricting the pos-

sibilities to adapt the method. Some approaches use XML-like files to store the models or XML-based implementation language, thus, they are aligned with standards. Following, we will demonstrate how our approach addresses these issues.

UsiXML follows a multi-path UI development based on the CRF [5], which defines UI development steps for multi-context interactive applications in a simplified manner. Now, we present that this framework can also be applied with a distributed team. Figure 1 depicts that in each level of the CRF, it is possible to consider people working in different time and space, but still connected through the execution of web services that support the method.

The MBUID method used to illustrate the workflow execution is divided in four phases: *inception*, *elaboration*, *construction*, and *transition*. These phases are composed of activities, organized in five disciplines: requirements, analysis and design, implementation, deployment, and test. The main roles are: system analyst, usability expert, requirements reviewer, UI designer, software architect, implementer, tester, and deployment manager.

The artifacts are a combination of SE and HCI, for instance, UML diagrams [10] and UsiXML models [22], respectively. The method engineer is the one responsible for the definition and maintenance of the method. To facilitate his/her work, standard notations and workflow editors can be used, which are presented in the next sections.

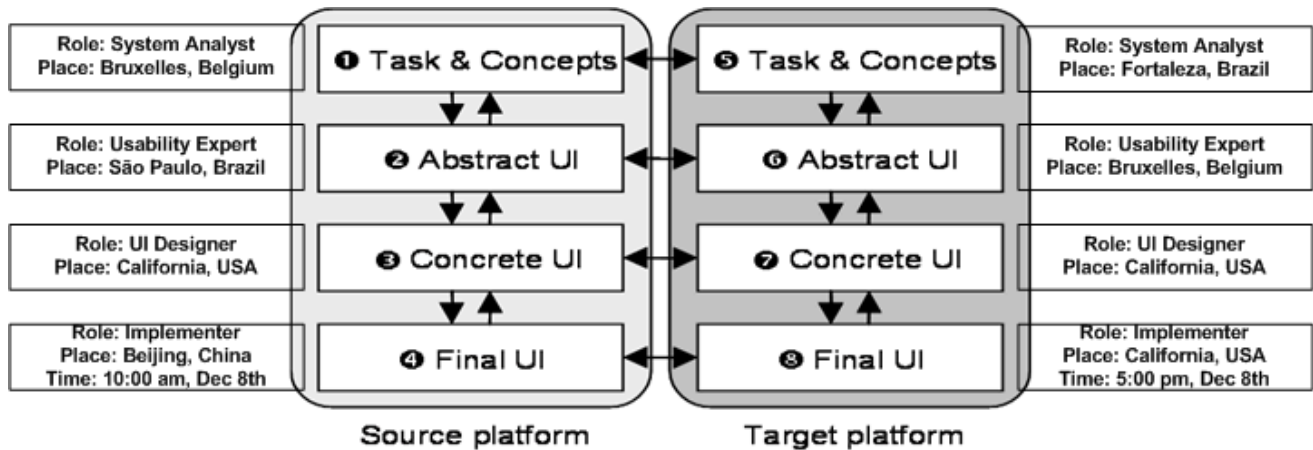


Figure 1. Distributed team applying the Cameleon Reference Framework.

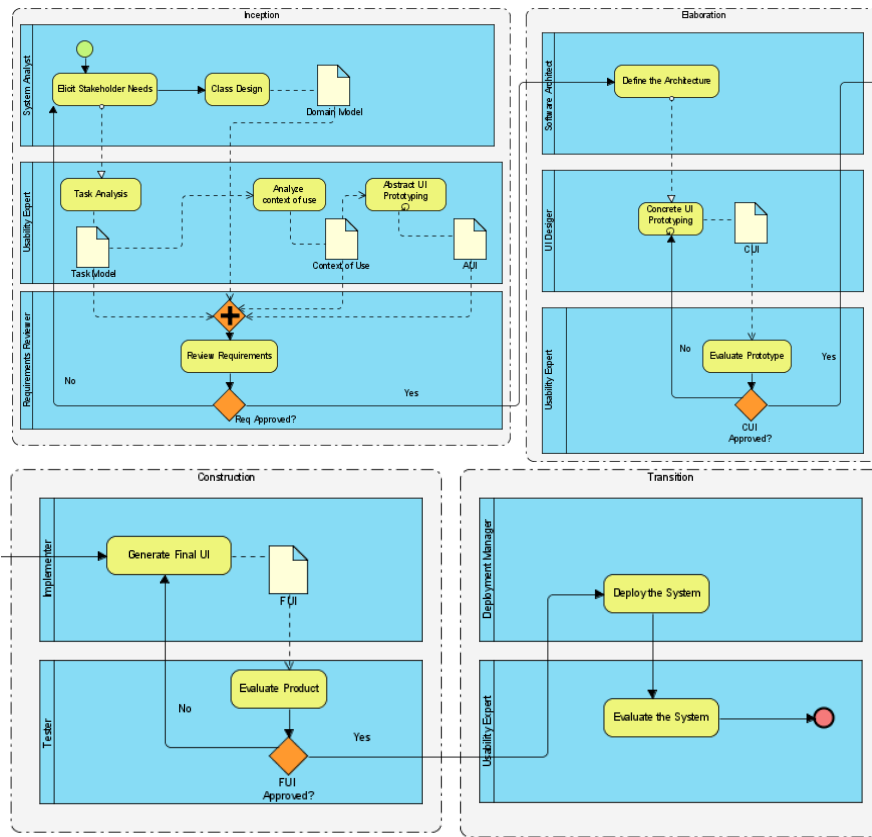


Figure 2. Example of method defined using BPMN editor.

SERVICES FOR MODEL MANAGEMENT IMPLEMENTATION

The UsiXML language describes the user interface for multiple context of use such as character user interfaces, graphical user interfaces, auditory and vocal user interfaces, virtual reality, and multimodal user interfaces. As a language explicitly based on the Cameleon Reference Framework, it adopts four development steps, which are (Figure 1):

- 1) *Task & Concepts (T&C)*: describe the various user's tasks to be carried out and the domain-oriented concepts as they are required by these tasks to be performed.
- 2) *Abstract UI (AUI)*: defines abstract containers (AC) and individual components (AIC) [15] two forms of Abstract Interaction Objects (AIO) [25] by grouping subtasks according to various criteria (e.g., task model structural patterns, cognitive load analysis, semantic relationships identification), a navigation scheme between the container and selects abstract individual component for each concept so that they are independent of any interaction modality, such as graphical, vocal, tactile, or haptic modalities. The AUI is said to be *independent of any interaction modality*.
- 3) *Concrete UI (CUI)*: concretizes an abstract UI for a given context of use into Concrete Interaction Objects

(CIOs) [25] so as to define widgets layout and interface navigation. It abstracts a final UI into a UI definition that is independent of any computing platform. While the AUI is independent of any interaction modality, a CUI assumes that a particular interaction modality has been chosen, but that this CUI remains *independent of any computing platform*.

- 4) *Final UI (FUI)*: is the operational UI i.e. any UI running on a particular computing platform either by interpretation (e.g., through a Web browser) or by execution (e.g., after compilation of code in an Integrated Development Environment (IDE)).

Across these four development steps are distributed five models, which are task model, domain model and context model in the first step, Abstract User Interface (AUI) in the second step, and Concrete User Interface (CUI) in the third step. The language does not consider the Final UI, but offers the necessary support to concept and generate it, when it is a specific technology [25]. Task & Concepts, respectively AUI, CUI, are the manifestations of Computing-Independent Model (CIM), respectively PIM, and PSM.

The models have a natural logic sequence. However, its sequence could not support many situations. For instance: if we already have a CUI and a new device needs to be considered, it is necessary to transform that CUI in an

AUI, to have a model platform independent. Moreover, there are many other possibilities that are treated by a model management approach.

It intends to be a higher level programming interface that offers a set of operations, supported by mappings between models [2]. Our approach is aimed at providing a set of model management operations as services to allow a collaborative work oriented by one or more methods, to integrate all available tools and to improve the architecture of the current and future tools. Other types of services are necessary to promote control and integration, as depicted in Table 1.

Type	Service	Description
Model management	Transform AUI to CUI, merge	Operations over models.
Control	Version check in / check out, exclude, add, etc.	Physical control over managed models.
Integration	Generate class diagram, refactoring, trace	Integration of UsiXML models with other models and processes.

Table 1. Types of services.

A service has different levels of granularity. A unique service can be responsible for an entire process, a sub-process, a process step or an operation of a process step [9]. The granularity is inversely proportional to the possibilities of services composition. So, we choose a low level of granularity to increase the applicability of the services in different situations.

In some situations, a specific service can be invoked separately by a tool, but, in most of the situations, an atomic service collaborates with other services to produce significant results. For example: a tool can call a specific transformation service to transform an AUI model in a CUI model. It will be done, but in a real application, a CUI Model could already exist and it is necessary to use a comparison service to check all changes made over the original CUI model and call a refactoring service to propagate each change in all managed models.

A SOA provides support for process orchestration, which is used to describe how services can interact with each other to provide new applications for each specific situation in the organization [20]. Orchestration promotes the services composition to make possible simple logics, as explained in the last example, and complex process definition, such as a software development process.

But, as described by Sadiq et al. [21], orchestration is a creative work and, until now, it needs to be done by hu-

man beings. So, this concept is not self evident and to be applied, we need to consider a specific domain, as HCI, a specific application, as model management, and a method to describe in details how everything works.

SERVICE ORCHESTRATION TO SUPPORT UID METHODS

A method supporting model management addresses model transformations as a means to produce UIs for a variety of platforms, such as in [4,23,26]. But, we intend to encompass a broader definition by considering the application of standards in the method definition.

To make a representative method, that is, to make it clearer to understand, apply and implement, it is important to present it in a flow format. Business Process Modeling Notation (BPMN) [18] was proposed to be applied in the representation of organizational processes. The use of BPMN in the method definition is important because: i) it has become a pattern for process modeling; ii) many software is available in the market implementing it; iii) it has been intended as a human-readable layer that hides the complexity of designing transactional business processes; and iv) there is a strong integration with the SOA.

There are a lot of elements in the BPMN notation. To take a complete overview of the notation, the specification is available on [18]. Since BPMN can be used in any business process modeling, we want to demonstrate that it is in accordance to Software Process Engineering Meta-model (SPEM), as described in the next sub-section, before actually presenting the method using BPMN elements.

We associate SPEM and BPMN to help method engineers specify the method and also to adhere to formal SE specifications. SPEM is a meta-model for defining software development processes and their components [19].

We associate SPEM and BPMN to help method engineers to specify the method using a workflow, and also to adhere to formal SE specifications. SPEM is a meta-model for defining software development processes and their components [18]. In a general view, the structure of the SPEM meta-model is organized as follows: *ProcessComponent* is a set of process description to assemble a process; *Process* is a process component intended to stand alone as a complete process; *Lifecycle* is the behavior of a process and it is defined as a sequence of phases; *Phase* is defined with the constraint of being executed sequentially with a series of milestones spread over time; *Iteration* is a composite of a phase with a minor milestone; *Discipline* partitions the activities of a process according to a common theme; *WorkDefinition* is the work performed in the process; *Activity* (specialization of *WorkDefinition*) is a piece of work performed by a process role; *Step* is an atomic element of an activity; *ProcessPerformer* performs a set of work definitions in a process; *ProcessRole* (spe-

cialization of ProcessPerformer) is responsible for specific work products by performing and assisting in specific activities; *WorkProduct* is an artifact produced, consumed, or modified by a process; and *Precondition* is a constraint that is expressed in terms of the states of the work products that are parameters of the work definition.

The lifecycle, phase, and iteration can be represented as a BPMN sub-process. For instance, a lifecycle can be represented as a sub-process composed of phases. The discipline can be represented as a BPMN group in order to delimitate the activities that are part of the same discipline. The activity can be a BPMN activity, since it can represent tasks, operations, and actions. When an activity is consisted of atomic elements, a step can be represented using a BPMN task, which is an atomic activity. A process performer can be represented as a BPMN pool since both are more generic representations of process participants and a process role (a subclass of a process performer) can be represented as a BPMN lane, which is a sub-partition within a pool. Work product is a BPMN data object since both are considered as artifacts in the process. Precondition can be a BPMN gateway that controls the sequence flow; or a BPMN conditional flow, which has condition expressions evaluated at runtime to determine whether or not the flow will be used.

SPEM suggests the definition of a method using UML diagrams, in particular, the following UML notations are useful: class diagram, package diagram, activity diagram, use case diagram, sequence diagram, and state chart diagram. Method engineers can use any BPMN editor to define the process workflow and make it available for professionals in their organization as guidance for their work, as depicted in Figures 2 and 3: the business process notation is used by tools to draw the method flow and all objects shown here can be found in the left tab.

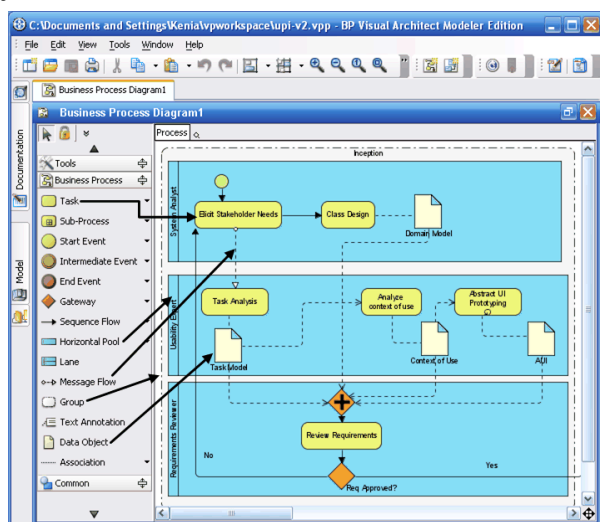


Figure 3. Defining the method with a BPMN editor.

Method Description

The method description is base on the following phases.

In the inception phase, the system analyst elicits requirements, then the usability expert describes the tasks, the context of use, and the prototype in abstract level while the system analyst performs class design, then the requirements reviewer analyzes all the models that have to be approved before going to the next phase. In the elaboration phase, the software architect defines the architecture and the UI designer designs the UI in a concrete level, then the UI is evaluated and approved by the usability expert before going to the next phase. In the construction phase, the implementer implements the UI in the final level.

The tester evaluates the UI before approving it for the final phase. In the transition phase, the deployment manager installs the system in the environment where the usability expert evaluates the system with end users, who have to approve it to consider the project finalized. Throughout the method, when an artifact is not approved, a new iteration can be executed until an acceptable version of the artifacts is produced.

Figure 3 illustrates the method using a flow chart organized according to the RUP structure [14] in four phases: inception, elaboration, construction, and transition. This flow was designed using the tool Business Process Visual Architect [26]. These phases are composed of activities, organized in five disciplines: requirements, analysis and design, implementation, deployment, and test. The main roles are: system analyst, usability expert, requirements reviewer, UI designer, software architect, implementer, tester, and deployment manager. The main UsiXML models are: task model, domain model, context model, abstract UI (AUI), and concrete UI (CUI). With the use of a UID method for the application of services for model management, we can present how SOA provides support for professionals working in a distributed manner towards the creation of consolidated artifacts.

Method Implementation

Each method activity, described in BPMN, is represented as a BPEL task, which is associated with services. Many services are invoked by the engine during the BPEL execution. The engine will control the execution of the flow, the state of process, the time spent in each task, saving important data for the process analysis. A method activity can be associated with one or more services of different types and a specific service can be used by various activities. This flexibility in their association makes it easier for the method engineer to define and improve the workflow by selecting available services. The services invoked by BPEL are called web services in an IT environment. [14] considers that web services are the main element of a SOA. It represents the business functionality and application logic available in conceptual business architecture to users, costumers and other services on an IT network.

Three types of web services available in the network to support the method:

1. Transformation Services: transformation between models;
2. Operation Services: operations performed over the models;
3. Software Engineering Services: execution of activities related to SE.

The *transformation services* make transformations between UsiXML models, passing one or many UsiXML models as input and receiving another UsiXML model as output. For instance, task model and domain model as input for the transformation into an abstract UI model. The *operation services* receive as input UsiXML models created or edited using the UsiXML tools (e.g., IdealXML for task and domain model, SketchiXML or GrafiXML for Concrete UI) in order to perform some of the main model management operators: match, compose, diff, modelGen, and merge. The *SE services* receive as input UsiXML models created or edited using the UsiXML tools in order to perform the following services: version control and traceability. These services handling various models are performed by accessing them from a common repository.

Method Deployment

Figure 4 represents the infrastructure adopted for the method execution, but it can be simplified or improved, considering the availability of resources. The idea is to make clear where the method is executed and how the models are manipulated. Once the method is defined in BPMN, it can be transformed in BPEL code using a tool that implements mapping rules. The BPEL engine will effectively execute it in the organization network. The execution of BPEL and the management of instances of this execution by the BPEL engine are called Orchestration. The organizational processes are executed by a set of available services, reusing what is already done and extending the current business rules.

The process orchestration, implemented by BPEL, makes a lot of calls to web services available in the network. The web services could be available in many points of the network and the BPEL Engine needs to call many web services on different places. But, it could be a problem in the future because the connections can become interlaced, thus increasing the possibility of service coupling. To solve this problem it is recommended to use an Enterprise Service Bus (ESB) to be an intermediary between the BPEL Engine and all web services available in the network. This technology is an implementation of a broker, an architectural pattern to structure distributed software systems with decoupled components that interact by remote service invocation. The ESB is responsible for coordinating communication, such as forwarding requests, also for transmitting results and exceptions. All communication intermediated by the Enterprise Service Bus are

defined by Simple Object Access Protocol (SOAP), a XML based protocol that works over TCP/IP to support web service messages between servers and clients. The entire solution is XML based. The models are defined in UsiXML, which will be transported on the network in a XML message, to be processed by services exposed using XML signatures, to be orchestrate by a BPEL (a XML to describe the execution of business logic), defined by BPMN, whose metadata is in XML format.

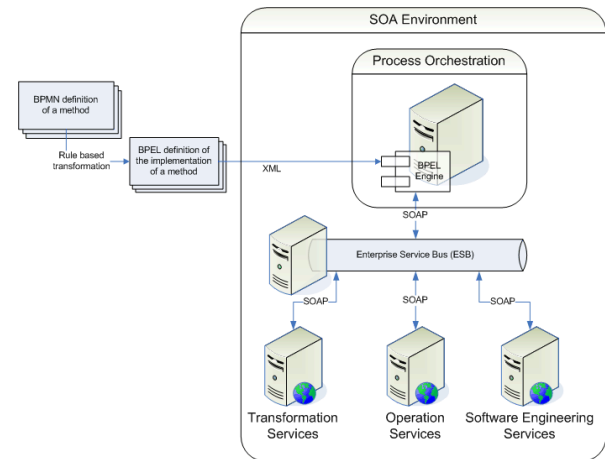


Figure 4. The architecture proposed.

Method Execution

Methods are then executed according to the normal SOA principles of orchestration. Table 2 presents the services created for each method activity.

Activity	Type of Service	Service Interface
Task analysis	Software Eng.	controlModel(taskModel)
Class design	Transformation	transformDomainModel(classDiagram)
Analyze Context of Use	Operation	addContextModel(contextModel)
Review Requirements	Software Eng.	evaluateModels(evaluatedModels)
Abstract UI Prototyping	Software Eng.	controlModel(auilModel)
ConcreteUI Prototyping	Software Eng.	controlModel(cuilModel)
Generate Final UI	Transformation	generateFUI(language, cuilModel)

Table 2. Association of Method Activities and Services.

SUPPORT FOR COMPUTER-MEDIATED COMMUNICATION

Technically speaking, the implementation of services in a specific platform is called Web Services [13]. The added term "Web" means that the service is available in an internet environment, and it can be invoked locally or remotely by other applications. All the communication is managed by a protocol called Simple Object Access Protocol (SOAP), based on XML, which is responsible for the encapsulation and transport of request messages from the client to the server and response messages from the server to the client. The availability of a web service can

improve substantially the management of the UsiXML models. One of the first steps is to migrate all the code in the tool implementation that manipulates UsiXML models to web services. It will reduce the complexity of the tool's code, make the implementation available for other tools, and improve the architecture of future tool implementations. A catalog of available services, among other things, will avoid the duplication of implementations.

A SOA is the basis to enable a correct design and implementation of all model management web services. According to [9], "SOA represents a open, agile, extensible, federated, composable architecture comprised of autonomous, QoS-capable, vendor diverse, interoperable, discoverable, and potentially reusable services, implemented as Web services". Part of the above advantages is supported by web services and orchestration, a concept introduced in Section 3.

As we discussed in Section 4, BPMN will provide the higher level definition of a SOA application to execute methods to offer support in the UID. Currently, BPMN can be transformed in the Business Process Execution Language (BPEL) [1]. In fact, most of the tools that work with BPMN can generate BPEL [11], reducing the effort of achieving an automated process. BPEL is a XML based language that can be executed by a BPEL engine that allows composition of web services and communication between them. Each method activity, described in BPMN, is represented as a BPEL task, which is associated with web services. The engine controls the execution of the flow, the state of process, the time spent in each task, saving important data for the process analysis.

Figure 5 represents the SOA support for methods and tools. We can see the transformation of BPMN in BPEL and the deployment of a BPEL in its engine. The engine can orchestrate services inside and outside the organization, widely distributed. A service-oriented application client will execute instances of the method and tools can make use of a particular web service and participate of the method execution, offering the support for a particular activity.

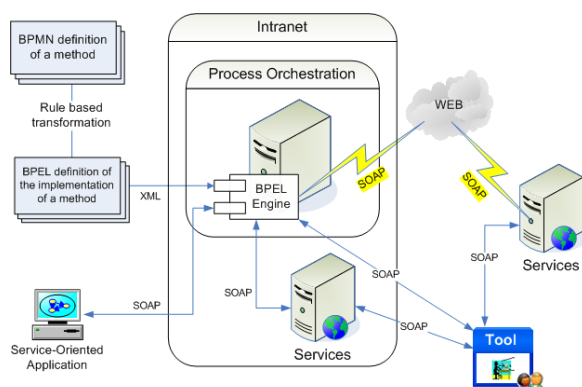


Figure 5. The SOA support for methods and tools.

CASE STUDY

Consider a software organization, located in Europe, which is responsible for a project for the design of UIs for an e-commerce web site for desktops and Palm Tops. But, the organization needs an expert in UI design for Palm Tops. Therefore, the directors decide to hire such an expert for this specific project, but who works remotely from South America. In order to improve their communication, the project manager trains the hired professional on the method and on the method execution application so all the outcomes are shared, that is, they are available for all professionals at anytime, anywhere. We now present how the execution of the method can be automated with SOA by depicting prototypes of the method execution application, a service-oriented application, and images of the models created during the execution of certain activities. During the *Task Analysis* activity of the inception phase, the usability expert uploaded the UsiXML task model file (Fig. 6). Whenever the usability expert makes changes in this model, he/she can upload the new version and a service is called to perform version control operations.

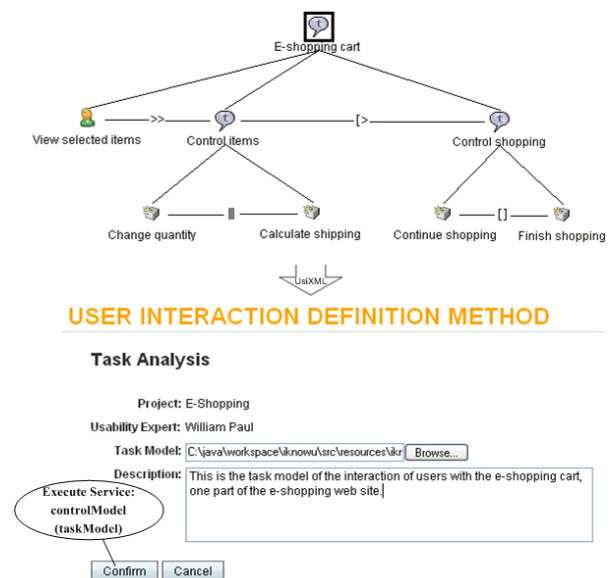


Figure 6. Task Analysis.

During the Analyze Context of Use activity of the inception phase, the usability expert can inform the data about the context of use (Figure 7). In this application, there are also pages devoted for user profile and platform. After requesting the data to be saved, a service is called to perform a model management method that creates the UsiXML context of use file. During the Abstract UI Prototyping activity of the inception phase (Figure 6), the usability expert uploads the UsiXML AUI model file, which was generated using IdealXML. The AUI could also have been automatically generated by another tool, which can call specific services that perform the transformation from conceptual models (task, domain, and context of use) into the AUI. When the resulting UsiXML AUI is uploaded, a

service is called to execute version control operations. If a new version of the AUI is uploaded, a service for refactoring is called to make the appropriate changes in the models associated to this AUI model, such as the conceptual models or even in the CUI, in cases when it was created, for instance, in a previous iteration of the lifecycle. During the Class Design activity of the inception phase, the system analyst can create the class diagram using any UML editor, upload the XMI file of the class diagram, and then request the generation of the UML class diagram into a UsiXML domain model (Figure 8).

USER INTERACTION DEFINITION METHOD

Context of Use Analysis

Project: E-Shopping
System Analyst: John Smith

Environments

New

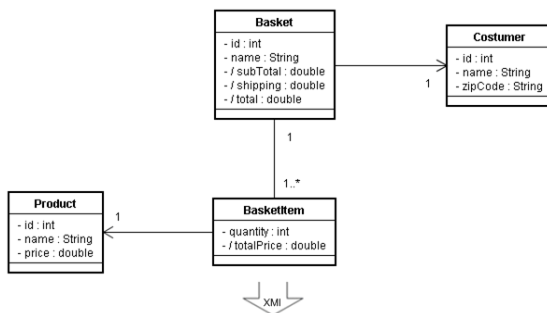
Name	Characteristics	
Home	Noisy: kids playing, light: weak	Delete
Office	Silence: people concentrated; light: bright	Delete
Cyber Café	Noisy: people talking, music playing; light: bright	Delete
		Delete
		Delete

Execute Service: addContextModel (contextModel)

Next >> Confirm Cancel

Figure 7. Analyze Context of Use.

This service performs a specific model management operation, which facilitates the integration of artifacts prepared using another modeling language with UsiXML models. It is visible here that any modeling tool can be used, and each one can have a different specification, visualization, among other aspects. With this application, we reinforce consistency in applying the method by providing such services for specific transformations.



USER INTERACTION DEFINITION METHOD

Class Design

Project: E-Shopping
System Analyst: Mary Anne

Class Diagram Definition File (XMI): C:\java\workspace\know\src\resources\ukr Browse...

Description: This is the UML class diagram with the classes and their properties to support the tasks of the e-shopping cart.

Execute Service: transformDomainModel (classDiagram)

Transform to Domain Model Cancel

Figure 8. Class Design.

During the *Review Requirements* activity of the inception phase, the requirements reviewer evaluates each of the models mentioned previously. When the reviews are saved, a service is called to send e-mails with the status of the evaluation and the list of possible changes for the professionals responsible for the models that need changes. This is an example of a service that facilitates the coordination between professionals. This service is associated to the other activities that evaluate artifacts, such as the *Evaluate Prototype*, *Evaluate Product*, and *Evaluate the System* activities. During the *Concrete UI Prototyping* activity of the elaboration phase, the usability expert creates the CUI model. When he/she uploads it, a service for version control is called. Similarly to all other models, the upload of a new version of the model triggers the service for refactoring. Since the models are in a common repository, it is easier to perform such a service. During the *Evaluate Prototype* activity of the elaboration phase, the expert in UIs for Palm Tops also plays the role of a reviewer and verifies if specific guidelines for this device are addressed in the CUI. This is one of the method activities that triggers refactoring services, since changes made in one model have to reflect on all associated models. During the *Generate Final UI* activity of the construction phase, the implementer selects the language to generate the UI. When the implementer requests the generation of HTML and WAP files, the service is called to generate the Web page and the page in a Mobile Phone (Fig. 9). Consider that this is an automatically generated UI, in which the designer will still incorporate the organization's style guide into it. For the moment, (X)HTML, Java, and XUL codes are automatically generated.

USER INTERACTION DEFINITION METHOD

Generate Final User Interface

Project: E-Shopping
Programmer: Mary Anne

Final User Interface Types

☒ HTML ☐ PDA
☐ Java/Swing ☒ Mobile Phone

Execute Service: generateFUI (language, cuiModel)

Generate Cancel

JSF JSP Java

Figure 9. Generate Final User Interface

These screens depict the most common interactions with the application that manages the method lifecycle. When professionals perform these activities, they interact with the application to: document the outcomes of their work (i.e. uploading files), perform their work (i.e. defining the context of use), receive outcomes that help in their work (i.e. use the generated FUI and add business rules in it), control the status of their work (i.e. view which activity of the process is being executed, receive notifications of which activity they need to perform), etc. This case study demonstrates that the execution of a method supported by a service-oriented application reinforces consistency in a project and even across different projects. Also note that if the method flow of activities is changed in the BPMN editor, so is the associated BPEL, therefore, the services will be invoked in a different sequence, accordingly to the method, thus, increasing the flexibility to make changes in the method, whenever appropriate.

CONCLUSION

This work presented an approach that uses SOA to support the transformation between models during the UID lifecycle. The specific issue we address is the support for communication through the use of an application that allows sharing and integrating the work performed by the software organization teams either locally (e.g. professionals designing the UI physically located in the same organization), distributed (e.g. professionals accessing the organization intranet), or remotely distributed (e.g. professionals accessing the internet to use the common application). Considering the existence of a set of tools that manipulate UsiXML during the UID lifecycle, we provide the foundation to integrate them through the reuse of services that manipulate UsiXML models to perform transformations. With the case study, we exemplified how real world organizations can use a standardized architecture to reuse implementation strategies in different projects to promote the institutionalization of UID by supporting collaborative work. There are some topics that are open for future work. For the method, we will work on its specification in details, and define how to make adaptations, depending on the organization or project. For the services, the next step is to implement them. Concerning project management, we intend to monitor the method using an engine that generates statistical data (e.g. comparison of project planned dates with real dates), and improve the method based on the results of the statistical data.

REFERENCES

1. BEA Systems, IBM Corporation, Microsoft Corporation, SAP AG, Siebel Systems, Business Process Execution Language for Web Services, 1.1, May 2003.
2. Bernstein, P. Applying Model Management to Classical Meta Data Problems. In: Conference on Innovative Data Systems Research (2003) 209-220.
3. Bias, Randolph G.; Mayhew, Deborah J. Cost-Justifying Usability: An Update for the Internet Age, Morgan Kaufmann Publishers Inc., San Francisco, CA. (2005).
4. Botterweck, Goetz and J. Felix Hampe. Capturing the Requirements for Multiple User Interfaces. 11th Australian Workshop on Requirements Engineering (AWRE'06), Adelaide, Australia, 2006
5. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J. A Unifying Reference Framework for Multi-Target User Interfaces. *Interacting with Computers* 15, 3 (June 2003), pp. 289-308.
6. Colombo M., Di Nitto E., Di Penta M., Distant D., Zuccalà M. Speaking a Common Language: A conceptual Model for Describing Service-Oriented Systems. In ICSOC'2005, 48-60.
7. Constantine, Larry and Lockwood, L. Software for Use: A Practical Guide to Models and Methods of Usage-Centered Design. Addison-Wesley, Reading.
8. Blind reference
9. Erl, Thomas. Service-Oriented Architecture – Concepts, Technology, and Design. Prentice Hall, Crawfordsville, Indiana (2005).
10. Göransson, Bengt; Gulliksen, Jan; Boivie, Inger. The usability design process - integrating user-centered systems design in the software development process. *Software Process: Improvement and Practice* 8(2): 111-131 (2003).
11. Harvey, Michael. Essential Business Process Modeling. O'Reilly Media (2005).
12. Jouault, Frédéric; Kurtev, Ivan. Transforming Models with ATL. Model Transformations in Practice (Workshop at MoDELS 2005), Montego Bay, Jamaica (2005).
13. Kreger, Heather. Web Services – Conceptual Architecture. IBM Software Group (2001).
14. Kruchten, Philippe. The Rational Unified Process - An Introduction. 2 ed. Addison-Wesley, New Jersey (2000).
15. Limbourg, Q., Vanderdonckt, J., *UsiXML: A User Interface Description Language Supporting Multiple Levels of Independence*, in Matera, M., Comai, S. (Eds.), "Engineering Advanced Web Applications", Rinton Press, Paramus, 2004, pp. 325-338.
16. Mellor, S.J., Scott, K., Uhl, A., Weise, D.: MDA Distilled: Principles of Model-Driven Architecture. Addison-Wesley, NY, 2004.
17. Montero, F., Víctor López Jaquero, V., IDEALXML: An Interaction Design Tool and a Task-based Approach to User Interface Design, Proc. of CA-DUI'2006, Chapter 20, Springer-Verlag, Berlin (2006) 245-252.
18. OMG, Business Process Modeling Notation Specification, 1.0, February, 2006.
19. OMG, Software Process Engineering Metamodel Specification, Jan. 2005.
20. Peltz, C. Web Services Orchestration and Choreography. *IEEE Computer*, Vol. 36, 10 (2003) 46-52.

21. Sadiq, Waqar; Racca, Felix. Business Services Orchestration - The Hypertier of Information Technology. Cambridge University Press, Cambridge (2003).
22. Schaffer, Eric. Institutionalization of Usability: A Step-by-Step Guide, Addison Wesley, (2004).
23. Sinnig, Daniel; Gaffar, Ashraf; Reichart, Daniel; Sef-fah, Ahmed; Forbrig, Peter. Patterns in Model-Based Engineering. In Proc. of CADUI'2004, Madeira, Portugal (2004) 195-208.
24. Soley, Richard. Model Driven Architecture. Available at: <http://www.omg.org/mda/presentations.htm>. OMG White Paper (2000).
25. Vanderdonckt, J. A MDA-Compliant Environment for Developing User Interfaces of Information Systems. In *Proc. of 17th Conf. on Advanced Information Systems Engineering CAiSE'05 (Porto, 13-17 June 2005)*, O. Pastor & J. Falcão e Cunha (eds.), Lecture Notes in Computer Science, Vol. 3520, Springer-Verlag, Berlin, 2005, pp. 16-31.
26. Visual Paradigm. Business Process Visual Architect. Available at: <http://www.visual-paradigm.com/product/bpva/>. Accessed on March 30th, 2007.
27. Wolff, Andreas; Forbrig, Peter; Dittmar, Anke; Reichart, Daniel. Linking GUI elements to tasks: supporting an evolutionary design process. Proc. of TAMODIA'2005, Gdansk, Poland (2005) 27-34

TRIAD: Triad-based Rich Internet Application Design

F. J. Martínez-Ruiz¹, Jean Vanderdonckt¹, Jaime Muñoz Arteaga²

¹Université catholique de Louvain, Louvain School of Management

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

{Francisco.martinez, jean.vanderdonckt@uclouvain.be} – www.isys.ucl.ac.be/bchi/members/java

²Universidad Autónoma de Aguascalientes Aguascalientes, Mexico

jmunozar@correo.uaa.mx

ABSTRACT

Current trends in web development still are attached to the web page paradigm. Nevertheless, new uses of already available technology and recent development in terms of concepts, as the asynchronous communication, have produced a new generation of web applications: Rich Internet Applications (RIAs). These web applications essays to fulfill user expectations in terms of usability, reliability, quality, maintainability and performance. In this paper, we are going to present a designing methodology that pursued as goal describing and developing User Interfaces of RIAs in a standardized way. The name of this ensemble of models and meta-descriptions is, TRIAD (Triad-based Rich internet Application Design).

Keywords

Rich internet applications, web engineering, Model driven engineering, USiXML

INTRODUCTION

Current trends in web development still are attached to the web page metaphor. The last generation of web applications is called, Rich Internet Application (RIA). These web applications break this paradigm and promise to fulfill user expectations in terms of usability, reliability, quality, maintainability and performance [18]. For this, a set of models based on UsiXML language (User Interface Description Language) supported by the CAMELEON Reference Framework are presented. Our work extends UsiXML User Interface Description Language with the purpose of designing UIs of Rich Internet Applications. This work presents these notations and the general method of design.

Indeed, software design includes a set of notations and models in order to specify different aspects of the software system. These notations specify applications in terms of various abstraction levels. The modeling of web applications has been treated in several works [18][19]. Roughly, we could classify web modeling in four categories: (1) hypertext models where organization and navigation is treated in a single model. [18](2) Data-driven models where exploitation of databases and a query language is translated into web applications [20]. (3) UI is defined in independent representations (e.g., UML) in a Model Driven Approach [21],[22] and Finally, (4) Task

based modeling, where web application is modeled in terms of tasks needed to complete the application goal. Our proposal takes a mixed approach since we use two categories: (3) and (4).

The rest of the paper is organized as follows: First, Core elements of the method are explained. Second, a running example shows the application of notations. And Third, Conclusions and Future work.

WHAT IS TRIAD?

In this paper, we are going to present a designing methodology that pursued as goal describing and developing User Interfaces of RIAs in a standardized way. The name of this ensemble of models and meta-descriptions is, TRIAD (Triad-based Rich Internet Application Design). TRIAD is a method for developing User Interfaces for RIAs. We start with an abstract definition of the UI. Then, in an iterative process more details are included until arriving to a concrete definition. Various features make TRIAD a viable choice: Extensibility (supports composition of structures), a set of Visual Patterns and the separation of concerns (data, logic and presentation).

CORE CONCEPTS OF THE METHOD

Now, we present the core concepts of our method which offers a scalable and model driven engineering approach [16] that is supported by the CAMELEON Reference Framework [5] and UsiXML language [4]. These concepts are: (1) Zoomable User Interfaces [15], which are applied to Task hierarchies (ZUIT concept) [13], (2) the triplet-based design concept and (3) the concept of location.

Zoomable User Interface Task Hierarchy (ZUIT)

TRIAD defines the application as a hierarchy of tasks. This hierarchy is composed by multiple levels where inner nodes are gathering elements and leaf nodes are atomic tasks. Tasks are connected by temporal operators of three types (Sequential, concurrent and choice). Nevertheless, there are some shortcomings in these models (discussed in [13]). For instance, the complexity is directly proportional to the size of the application and at some point, icons and texts become unintelligible. Another problem is that, models as ConcurTaskTrees notation (CTTs) (see Figure) which is discussed in [13], do not provide any semantic information through its structure

since the structure is simply replicated at all levels. For instance, in a minimal Sign in application (see, Figure a) we have two tasks (Get information and Submit). First you have to recollect user data before submit it. Then, tasks are related by a sequential operator ($[] \gg$). Get information is subdivided into two atomic tasks (Input User Name, Input Password) which are executed at the same time ($[[]]$). We use a treemap-like representation [14], instead of using an arborescence representation (Figure 1a) with an important variation: indeed, it is a Zoomable User Interface. The representation of the Sign in application is shown in Figure 1b. All tasks under the area of Login rectangle are her children which would be executed in sequential order (from left to right). Under the first child, **Get information** we have two other children. Here, for the sake of simplicity all the tasks are labeled but the only information visible at the beginning is the color (which is associated to operator and task types). We have chosen Piccolo toolkit [4] for taking advantage of its zooming predefined libraries. The benefits expand in three aspects. First, the navigation to inner hierarchies is done in a more intuitive way. Second, the visual overload is reduced since we are using coding schemes in order to identify task types and task relationships (Note: Here, for the sake of simplicity is not expressed). And third, more coding schemes could be integrated. For instance, weight metrics in order to adjust task area of the elements in the ZUIT.

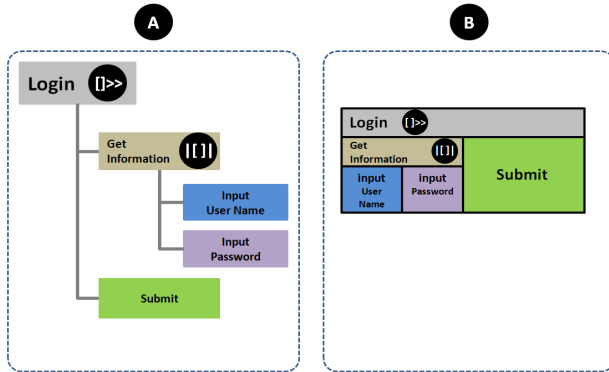


Figure 1. Login example in ZUIT format.

Triplet-based Design

The idea behind our method is simple: Keep designers focused on their tasks so that they do not become distracted when they go through the development process. Their objective is to define the sequence of tasks that is needed to accomplish the application goal. For each task T introduced by the designer, task triplets (triads) are introduced in the model. T is substituted by T' under which R (Robustness) and D (Decorative) set of tasks are added (Figure a). These groups exchange information between them and with the original task (which is part of the Utilitarian task set¹⁷). In order to update the model in

terms of validation and presentation. In Figure 2b, this structure is represented as a ZUIT. Note: the triad is defined by the task sets: $R(T)$, $U(T)$ and $D(T)$. That means that each set (R , U or D) is populated, with pertinent tasks, in function of the necessities of T .

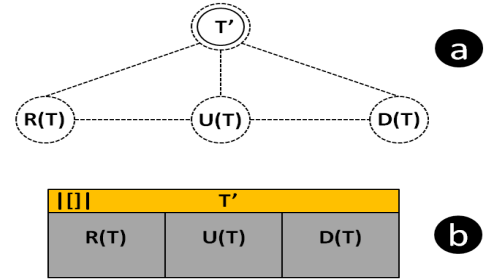


Figure 2. Triplet of task model.

Our model is loosely based in the triad of Vitruvius [8]. He stated that every building design should have three qualities: It must be strong, useful, and beautiful. We translate these features for UI design into: (1) Robustness tasks. This set agglutinates all tasks related to prevention, validation and recuperation. (2) Utilitarian tasks. This set includes Input, Output, Control and navigation tasks. (Besides CRUD operations to be applied on associated data [8]). That is, all the possible operations to be executed over tasks. (3) Decorative tasks. This set groups all tasks related to aesthetic aspects of the presentation of both, robustness and utilitarian tasks of UI. An overview of triads is shown in Tables 1, 2, and 3. Note that these are examples of the tasks in each category; it is not an exhaustive enumeration of them.

Task Operations	
	Description
Input	Entry of data
Output	Describes what information may be shown to the user
Control	Trigger a method of business logic
Navigation	Describes a container transition

Table 1. Utilitarian Tasks.

Prevention (task is affected a priori)	Validation (task is affected during interaction)	Recuperation (task is affected a posteriori)
Static User Help	Data Type check	Roll back
hidden tasks until needed (Displaying code at the last possible moment)	Order check	Error User help (Acknowledge)
	Auto-complete	Reset
	Dynamic User Help	

Table 2. Robustness tasks.

Decorative Tasks		
	Description	Example
Semantic	Affects semantic presentation, in general this means the value to be used.	A error signal icon is changed by other in term of internationalization

¹⁷ This set include CRUD and Task operations.

Syntactic	Affects order and presentation of task sequence to be done by the user	Change a text box for a date picker to simply the introduction of a date
Lexical	Affects order and presentation of lexical elements	A button could include a label and or icon or both
Alphabetic	Only affects alphabetic representation	Change from centimeter to millimeter values of a component
Physical	only affects physical appearance	Color, size

Table 3. Decorative Tasks.

Now, we are going to update the Login example (Figure 2). Each task in the Task Hierarchy is wrapped by the triad. This process is automated by XLST transformations in the UsiXML code.

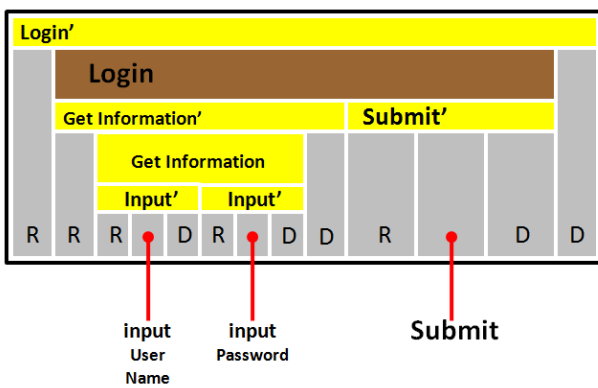


Figure 2. Inclusion of task Triad in Login Example.

The incorporation of the sense of location

It is time to focus in the Web Realm and for this, TRIAD introduces a pattern set in order to model **distributed nature** of RIAs. The original tasks are wrapped by these pattern structures which are added hierarchies that take into account location of tasks (see Table 4). This location is not related to physical places (e.g., client and server) at AUI level. The sense of location has two goals: (1) add hint labels to be used in the next step of the method and (2) populate the TH with tasks related to manipulation and/or system-oriented [17] (which are going to remain hidden until next step). An intuitive definition of location is that web applications could be divided into n distributed blocks. Nevertheless, we could generalize these arrangements into a two-block model. That is, some components of web application (presentation, logic and data) are located in the client (e.g., browser or local machine). This location is called Local Realm (see W1 in Figure 3b and c). The rest of components are assigned to the Foreign Realm (see Figure 3e). This last one includes server-side components, such as web services, web applications hosted in alternative places than user interaction point (usually his/her web browser). Also, it is possible to extend this definition to include even a local web server or application server. The logic to define the two blocks implies as local the application executed in the client host

environment, generally a web browser. It is worth notice that Figure3d describes a mixed realm task. This implies a compound task is internally composed by some elements in local and others in foreign realms. A concrete example is a submit operation which include client and server validation. Nevertheless, This type of task is included in foreign location in order to simplify the notation. Task types used in the Task Hierarchy level are wrapped up and divided in terms of their scope (local or foreign). In Table 3, it is described the set of labels to be added. Their work is adding hints or indicators in order to assist designers and the method in the translation into concrete web components. For instance, if the developer wishes to introduce an interaction task, for him/her the process is straightforward. But the real model will be updated with task triads that take into account the location.

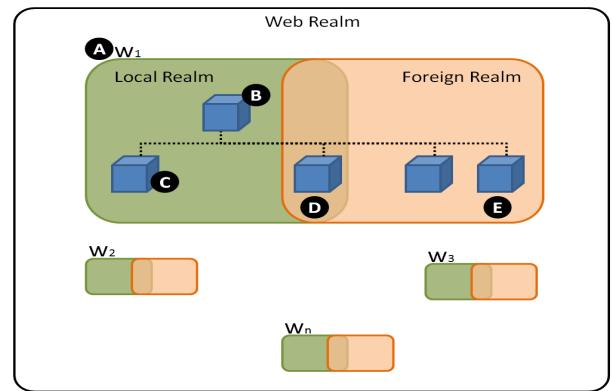


Figure 3. The Web Realm.

Description	Local Realm	Foreign Realm
Interaction task		
Application task		
Abstract task		
Link		

Table 4. Wrapper notation for RIAs.

GETTING STARTED

This section describes our method step by step modifying a case study in order to show all the updates.

Features of the Method

The development of web applications needs to be very flexible. Changes could appear in all the steps. In order to deal with this characteristic, our method includes two important features: First, the separation of concerns principle is integrated to the method and second is based in a Model Driven Approach.

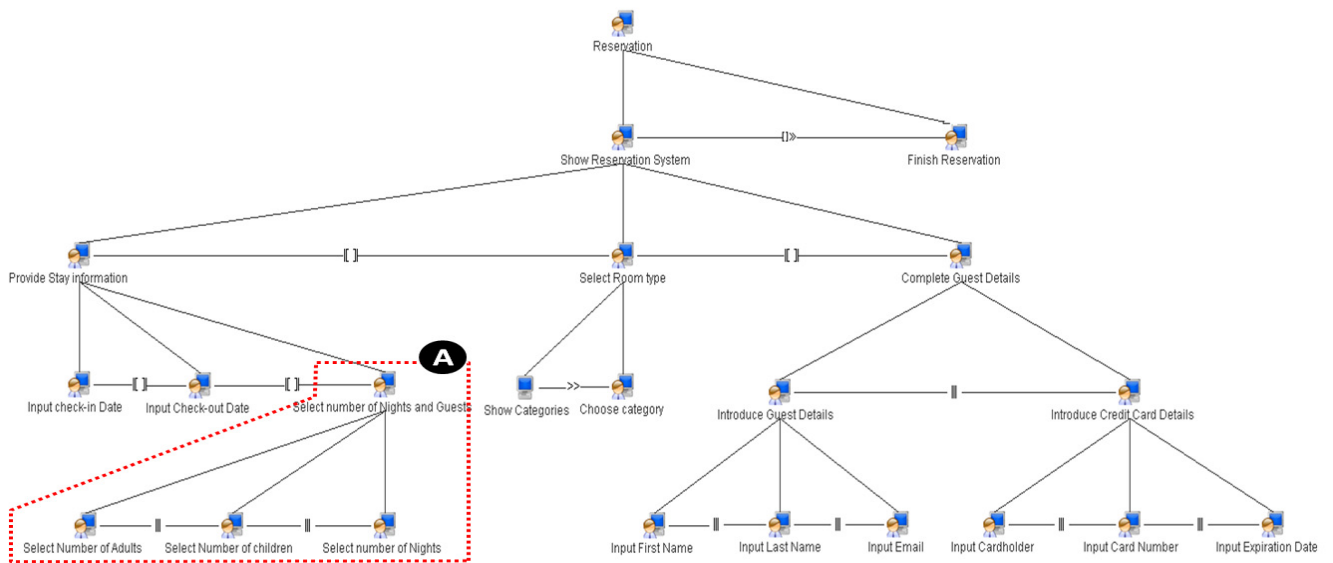


Figure 4. Example of an application for booking a hotel room in CTT model.

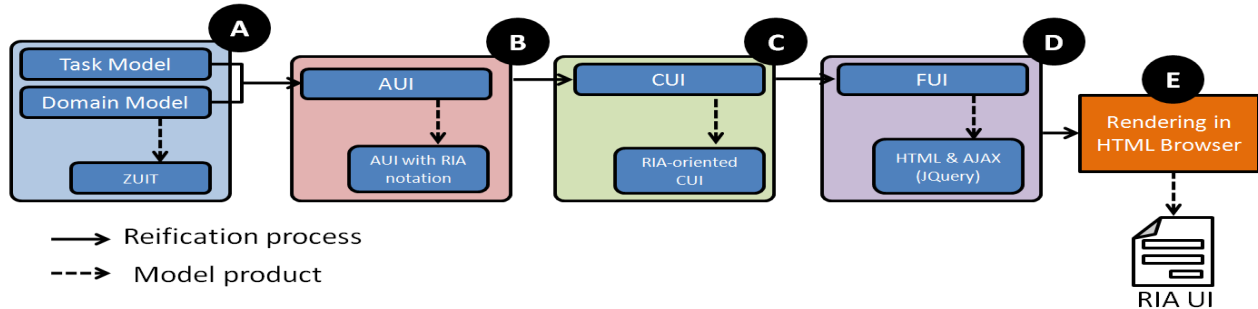


Figure 5. TRIAD Method.

The general Method

Our method is a process of translation abstract models (Figure 4) into more concrete ones (see Figure 5) [3]. The general process could be synthesized in the following steps: In the first phase, we create two models: a Task hierarchy model which describes User's goal and a Domain model to represent the data needed (T&D). The second phase implies the production of an Abstract User Interface (AUI) without any context or compromise with any technological platform. Then, in a Third phase a Concrete User Interface (CUI) is derived from the previous model. There, modality and platform widgets are decided. In the last phase, a Final User Interface (FUI) is obtained for a specific technology (For instance AJAX, .NET, LZX, SWF among others).

CASE STUDY: BOOK A ROOM

We use a Reservation System example. This simple UI allows users to book a hotel room. Check-in and Check-out dates, as well as, the room type and number of guests are selected. The UI is presented in a single interface over three sections. Validation is done during submission and during the recollection of values. This running example is named: Reservation System Case Study (RSC). We are

going to focus our attention on the models produced for the different phases of the method. A final remark is that model translation is based on XSLT in all steps.

Step 1: Creation of Task Hierarchy and Domain models

The recollection of User requirements are out of the scope of this work. They could be recollected by UML use cases and activity diagrams. These requirements are expressed in a hierarchy of tasks (TH) and Domain models (UML Class Diagrams).

Task Hierarchy model

The goal of the application is described by a TH. This TH is decomposed in successive levels until arriving to atomic tasks related to system manipulation. In RSC, the goal is booking a room. Then, this task could be decomposed in three subtasks (Pick a Date, Select Room Type and Complete Guest Details) which could be decomposed as well in other sub tasks. The result of this is presented in Figure 4. Also, the temporal relationships of tasks are modeled by temporal operators of three types (Sequential, concurrent and choice). For instance, in the first level of the TH, We have to reserve a room before confirm the transaction. This relationship is expressed with a sequen-

tial operator ([>>]). This representation (a variant of CTT) was created with the IDEALXML tool which the current tool used one in UsiXML [11]. Also we have in Figure 6 the proposed Domain model. Then, TH is translated into ZUIT format (see Figure 7) using XLST. In Figure 4a, we marked a fragment in order to show its UsiXML representation (Figure 8). This fragment allows presenting the code which will be refined in following sections.

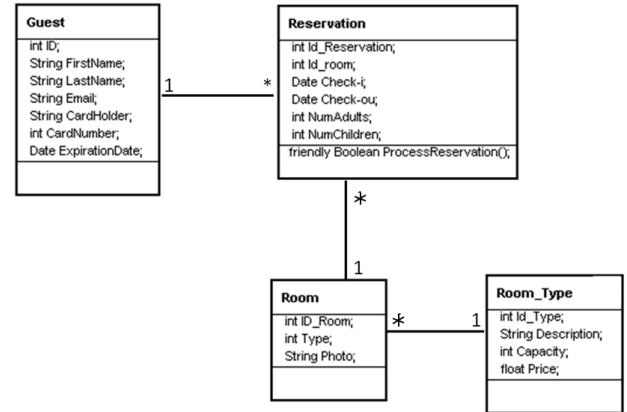


Figure 6. Domain model.

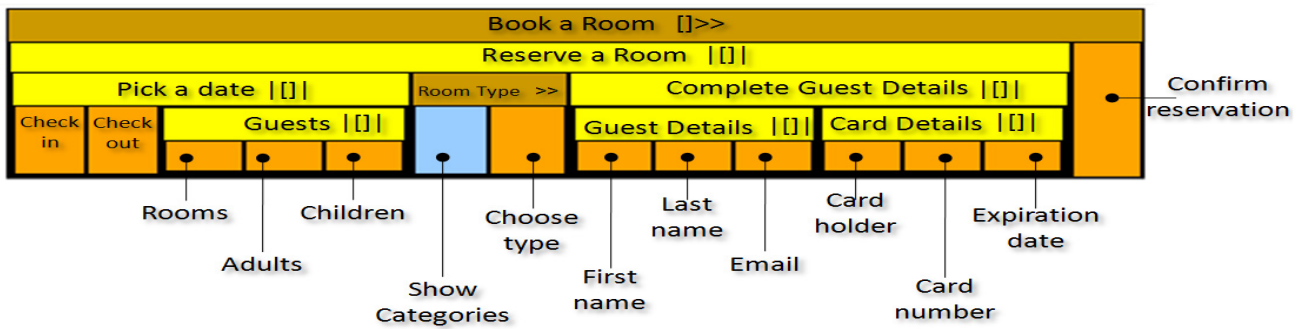


Figure 7. ZUIT representation of Task Hierarchy model.

```
<?xml version="1.0" encoding="UTF-8" ?>
<taskmodel>

...
<task id="st0task8" name="Select number of Nights and Guests" type="interaction">
  <task id="st0task19" name="Select Number of Adults" type="interaction" />
  <task id="st0task20" name="Select Number of children" type="interaction" />
  <task id="st0task21" name="Select number of Nights" type="interaction" />
</task>
<orderIndependence>
  <source sourceId="st0task19" />
  <target targetId="st0task20" />
</orderIndependence>
<orderIndependence>
  <source sourceId="st0task20" />
  <target targetId="st0task21" />
</orderIndependence>
...
</taskmodel>
```

Figure 8. Fragment of UsiXML code of TH.

Mapping Model

The models in TRIAD are connected through a mapping model that aids to define diverse relationships (this model is defined in UsiXML). These relationships allow features as derivation from models from one level to another, helping in the successive process of adding component features and selection of elements (depending on the level, abstract components or concrete ones), also for addressing context aspects (e.g., if the context defines a java-based platform which version of virtual machine is needed).

Available Relationships between domain model and other used models are: **triggers**, **observes**, **updates**, **isReified-**

By, **isAbstractedInto**, **isExecutedIn**, **isTranslatedInto**, **manipulates** and **hasContext** (see [7] for more information about them). In the case of RSC, a relationship of manipulates type is established between the Domain and Task Models. For instance, Figure 9 depicts “manipulates” relationships between task and domain model as pointed arrows. **Pick a Date** is mapped onto **Reservation** class. **Complete Guest Details** is mapped onto **Guest** class. **Show Categories** is mapped onto **Room_Type** class. And finally, **Confirm reservation** is mapped onto the method **ProcesReservation** of the class **Reservation**.

Manipulation of TRIAD tasks

The TH begins with the tasks needed to fulfill the goal of the application. The next step is manipulating triad tasks. That is, the developer should select decorative, robustness and utilitarian tasks, see in Figure 10a a mock up of the development environment that we are developing. Here, **FirstName** is selected to change its features. A possible visualization of these sets is shown in Figure 10b. In this case, instantiated attributes are colored in green (Figure c) when you move over Figure 10a. New features or modifications could be done by Triad Management Pattern inside a development environment (Figure 11). Note: We use this pattern to describe the *modus operandi* of the triad task repository.

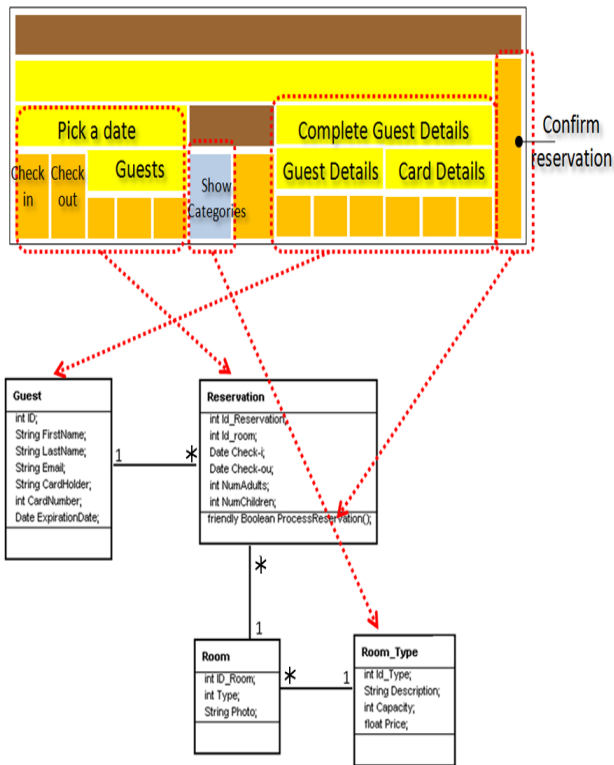


Figure 9. Mapping model.

Step 2: Building the Abstract User Interface

The next step is translating the TH into an AUI model. The process implies the introduction of more details and precisions. The AUI is improved with precisions about location that we introduce in the previous phase. According to an algorithm described in [7] and improved in [8], each inner node is interpreted as an Abstract Container (AC). In Figure 12a is depicted how this operation is done: rounded squares specify task grouping. The algorithm of [7] is straightforward: inner nodes are labeled and converted into Abstract Containers and leaf nodes in Abstract Interaction Components (Figure 13b).

A new visual notation is introduced for modeling AUI representation (see Figure 13). This one includes three main features: First, temporal operators are used in Figure 13a in order to explain the abstract dialogue inherited from the previous model. Second, in Figure 13b pointed arrows are used to express abstract adjacency (i.e., the spatial disposition of components). And Third, the RIA patterns are applied and used in each AC and AIC to indicate their application, Figure 13c.

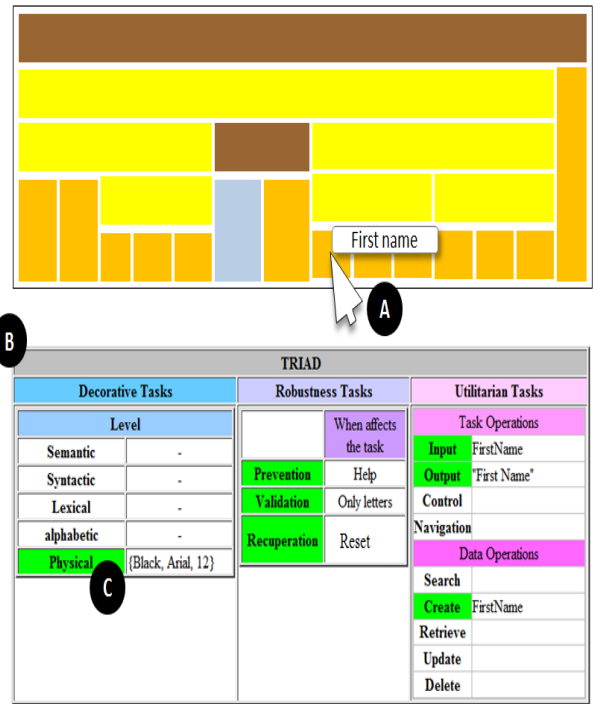


Figure 10. ZUIT representation of RSC with instantiated attributes (b).

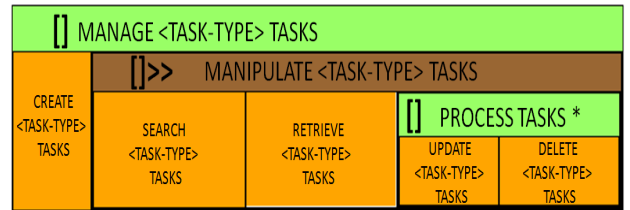


Figure 11. TRIAD instantiation pattern for selecting attributes of tasks.

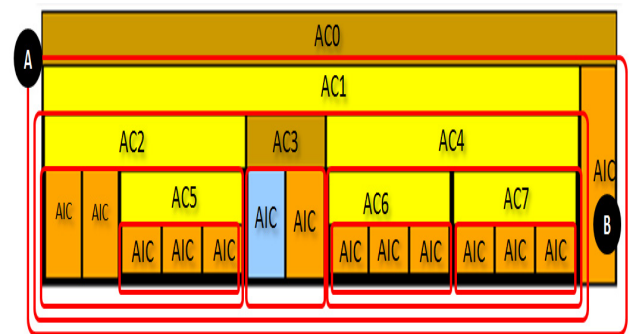


Figure 12. Visual explanation of recovery of Abstract components from ZUIT.

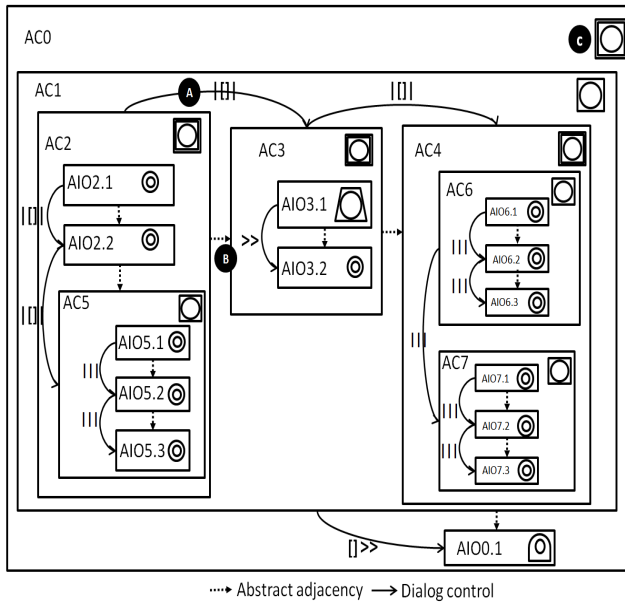


Figure 13. AUI representation of the RSC.

Step 3: Building the Concrete User Interface

A Concrete User Interface (CUI) model is composed of Concrete Interaction Objects (CIO) and concrete relationships. CUI model designates a specific modality (e.g., Graphic or speech). Therefore, behavior and UI elements are adjusted in order to fulfill requirements of this modality. The graphical modality could be described in terms of graphical input **GI** and graphical output **GO**, **GI** is defined by tuple (pointing device **P**, direct manipulation) where **P** usually is a mouse. **GO** is defined by tuple (Screen, Drawing language) where drawing language could be procedural, declarative, pixel or vector based [7].

At this level, compromise with any toolkit or platform is not yet done. Nevertheless, it is possible to define a CUI in term of valid elements of the modality. For instance, if the graphical modality is selected and the element to be reified is an **explicit container**. Then, the valid set of elements includes window, box, dialogue (defined in UsiXML). In the other hand, if a foreign interaction is associated to an AIC. Then, a RIA- CIO could be selected.

The process of choosing Concrete Interaction Components (CICs)

From the triad table (see Figure 10b) the developer has already chosen the different facets (input, output, control or navigation), data operation (CRUD), data types, and cardinalities among others. Nevertheless, the process has to be refined in order to select components related to the current modality. In Table 5, we can see possible derivations from AUI elements into CUI ones in RSC for **AC2**. The choice is driven by the selection of options (using a decision tree [8]) in the triad and also depending on developer preferences.

Abstract Interaction Component	Facet Specification or/and RIA label	Relevant Information to define which CIC will be selected	Possible Concrete Interaction Component to be used
Pick a date	Explicit container/ Output facet	The container is fixed	A box with a label (output) "Pick a Date"
Check in Date	Local input/ Output "Check in"	Data type, Domain value	A label "Check in" with a date picker
Check out Date	Local input/ Output "Check out"	Data type, Domain value	A label "Check in" with a date picker
Select Guest number and Type	Implicit container/ Output facet	The container is fixed	A box with a label "Select Guest number and Type"
Adult Number	Local input/ Output "Adult num"	Data type, Domain value	A label "Check in" with a dropdown list
Room Number	Local input/ Output "Room num"	Data type, Domain value	A label "Check in" with a dropdown list
Children Number	Local input/ Output "Children num"	Data type, Domain value	A label "Check in" with a dropdown list

Table 5. Translation between AIO types into CIO ones of AC2.

Modeling the behavior

The modeling of behavior has been treated until here in terms of rough granularity. That is, the dialogue in a sort of "Big Picture". Now, it is time to pay attention to details. And for this, we used the Abstract Data View notation or ADV-charts [12].

This is a notation for describing the behavior of interactive systems. Also it provides a way to define the flow control and the relationships between UI components and their events. ADV-charts are composed by ADVs, states, attributes and transitions. The representation of ADVs are treated as equivalent to Concrete Containers and CICs). States are depicted as rounded rectangles which contains the name of the state. More than one state is possible per ADV. States could be alone or in a cluster. States are linked to other with transitions (arrows) which are indexed and explained.

In Figure 14a, it is an example of ADV Book a room, in Figure 14b a state, Figure 14c a transition. And finally, Figure 14d represents a concurrent execution of components. It is needed to define each transition after defining ADV-charts. First, we describe its preconditions, then the event that triggers the transition and finally post conditions. Note: For the sake of simplicity, only a sub set of the available transitions are shown in Table 6.

#	Triggered by	Comments
1	Display	Active the application
2	Focus (produced by keyboard or mouse click)	-
3	Dynamic recovery of data view	Post conditions {screen = screen + ShowCategories}

Table 6. Some of the transitions to model the behavior of RSC

Now, with modality selected and Behavior more detailed, possible CUI representations are shown in Figure 15 (HTML- based code to have a visual representation) and Figure 16 (with GrafiXML tool [1]) both represents UisiXML CUI code (see Figure 17).

Step 4: Building the Final User Interface

In this step are produced Final User Interfaces (FUI). That is, UIs described by a specific platform (.NET, LZX, SWF and GWT among others). The CUI is translated again with XSLT templates and finally, we have code in the target language. In this way, native code is produced in order to be treated by interpreters, compilers, generators or converters of platform.

We translate CUI specifications into native widget sets available in the chosen platform. One of the advantages of the method is its capability of redirection. That is, CUI models could be oriented to different final platforms. This section shows the CUI definition of RSC into AJAX code (in particular, JQuery [5]) a possible final result is shown in Figure 18.

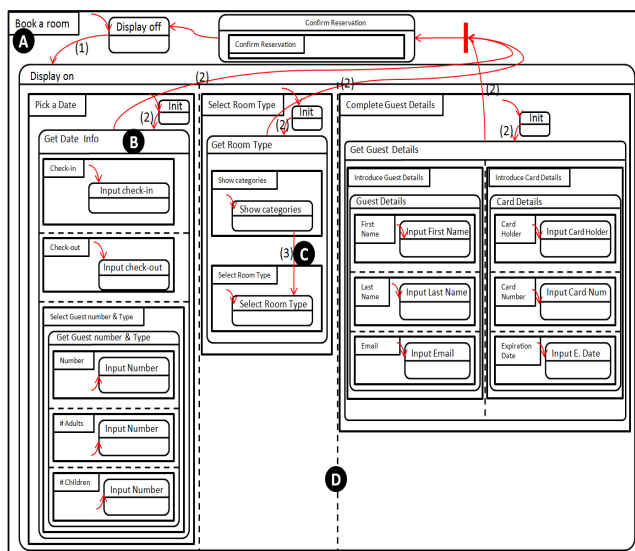


Figure 14. Updated Concrete dialogue of RSC

Figure 15. CUI representation of RSC.

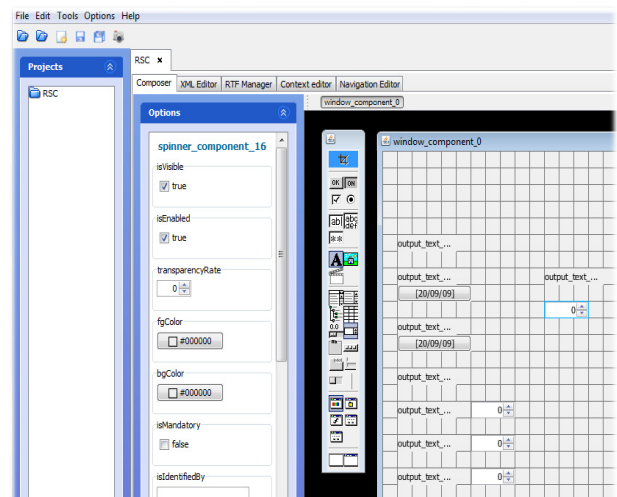


Figure 16. Screenshot of GrafiXML tool for CUI development.

```
<cuiModel id="RSC-cui_20" name="RSC-cui">
  <window id="window_component_0" name="window_component_0"
    width="626" height="498">
    <gridBagBox id="grid_bag_box_1" name="grid_bag_box_1"
      gridHeight="24" gridWidth="31">
      <constraint gridx="1" gridy="5" gridwidth="4"
        gridheight="1" weightx="1.0" weighty="1.0"
        fill="both" insets="0,0,0,0">
        <outputText id="output_text_component_2"
          name="output_text_component_2"
          tooltip="/uiModel/resourceModel/cioRef/cioId=output_text_component_2]/resource/@tooltip"
          defaultTooltip="Pick a Date"
          content="/uiModel/resourceModel/cioRef/cioId=output_text_component_2]/resource/@content"
          defaultContent="Pick a Date" isVisible="true"
          isEnabled="true" isBold="true" textColor="#000000"/>
      </constraint>
```

Figure 17. Fragment of CUI definition of RSC.

Figure 18. UI example with JQuery embedded in HTML

CONCLUSION AND DIRECTIONS FOR FUTURE WORK

In this paper, we have presented TRIAD, a MDA method for developing UIs of RIAs, which provides an ensemble of models in order to treat the complexity of RIA design. The first step includes a Zoomable representation of the UI that have been proposed to avoid many of the disadvantages of typical task representation in UsiXML, such as tree node explosion and lack of structure [13]. Piccolo framework [4] is used to implement ZUITs. The design method helps developers to focus in their work instead of dealing prematurely with tasks related to operation and security. This is done by the introduction of the triad concept as integral part of development. Task model has received special attention. Particularly, its container structure[9]. AUI representation has been enriched with the RIA notation, which is proposed to introduce, early in the development, information about RIA-oriented tasks. We used a minimalistic case study to show these concepts and their applicability. Therefore, we just give a glance of the real potential of the method to deal with RIAs. Finally, this approach is not only applicable to RIAs and it could be used for others types of applications.

Our main objective is to establish TRIAD as a plausible alternative method for UI-oriented development of RIA applications. Obviously, this is a first version. TRIAD approach will continue evolving. This process is due to updating current features: Our Repository of triad tasks is being updated with news tasks; Better behavior modeling; Adaptation to mobile technologies; as well as, a better understanding of the quality in the overall process; Better metrics in order to measure the task weights and exploit more semantic of ZUITs. Finally a Development environment is being developed in order to avoid manual process between models as in the current state.

ACKNOWLEDGEMENTS

This work is supported by the Programme AlBan, the European Union Programme of High Level Scholarships for Latin America, scholarship No. (E06D101371MX), The University of Zacatecas (UAZ) and the Belgian Computer Human Interaction Laboratory. The authors also acknowledge the support of the Région Wallonne for ITEA2 Call 3 UsiXML project, under reference 2008026.

REFERENCES

1. UsiXML. <http://www.usixml.org/> (May 10th, 2009)
2. Martínez-Ruiz, F.J., Muñoz Arteaga, J., Vanderdonckt, J., González-Calleros, J.M., *A First Draft of a Model-driven Method for Designing Graphical User Interfaces of Rich Internet Applications*, Proc. of 4th Latin American Web Congress LA-Web'2006 (Puebla Cholula, 25-27 October 2006), J.A. Sanchez (ed.), IEEE Computer Society Press, Los Alamitos, 2006, pp. 32-38.
3. OMG, <http://www.omg.org> , (May 10th, 2009)
4. <http://www.piccolo2d.org> (May 10th, 2009).
5. <http://jquery.com> (May 10th, 2009).
6. Luyten, K., Clerckx, T., Coninx, K., Vanderdonckt, J., *Derivation of a Dialog Model from a Task Model by Activity Chain Extraction*, Proc. of 10th Int. Conf. on Design, Specification, and Verification of Interactive Systems DSV-IS'2003 (Madeira, 4-6 June 2003), J. Jorge, N.J. Nunes, J. Cunha (eds.), Lecture Notes in Computer Science, Vol. 2844, Springer-Verlag, Berlin, 2003, pp. 203-217.
7. Limbourg, Q., Vanderdonckt, J., *UsiXML: A User Interface Description Language Supporting Multiple Levels of Independence*, in Matera, M., Comai, S. (Eds.), "Engineering Advanced Web Applications", Rinton Press, Paramus, 2004, pp. 325-338.
8. Albin, S. T. (2003). The art of software architecture. John Wiley and Sons.
9. Martínez, F., Vanderdonckt, J., Muñoz-Arteaga, J. Web user interface generation for multiple platforms. In *Proc. of 7th Int. Workshop on Web-Oriented Software Technologies IWWOST'2008* (Yorktown Heights, July 14th, 2008), L. Olsina, O. Pastor, D. Schwabe, G. Rossi, M. Winckler (eds.), CEUR Workshop Proceedings, Vol. 445, 2008, pp. 63-68. Accessible at <http://ceur-ws.org/Vol-445/02icwe2008ws-iwwost11-martinez-ruiz.pdf>
10. Bartalos, Peter & Bielíková, Mária: (S)CRUD pattern support for semantic web applications. Safarik University, Kosice, Slovakia (2008).
11. Montero Simarro, F.; Lopez Jaquero, V., IDE-ALXML: An Interaction Design Tool. Computer-Aided Design of User Interfaces V. Publisher Springer, 2007, pp.245-252.
12. Carneiro, L. M., Cowan, D. D., and Lucena, C. J. 1994. ADVcharts: a visual formalism for interactive systems. SIGCHI Bull.26, 2 (Apr. 1994), 74-77.

13. Martínez-Ruiz, F., Vanderdonckt, J., and González-Calleros, J.M. Model Driven Engineering of Rich Internet Applications Equipped with Zoomable User Interfaces. In *Proc. of Joint 4th Latin American Conference on Human-Computer Interaction-7th Latin American Web Congress LA-Web/CLIHC'2009* (Merida, November 9-11, 2009), E. Chavez, E. Furtado, A. Moran (Eds.), IEEE Computer Society Press, Los Alamitos, 2009, pp. 44-51.
14. Johnson, B. and Shneiderman, B. 1991. Tree-Maps: a space-filling approach to the visualization of hierarchical information structures. In *Proceedings of the 2nd Conference on Visualization '91* (San Diego, California, October 22 - 25, 1991). G. M. Nielson and L. Rosenblum, Eds. IEEE Visualization. IEEE Computer Society Press.
15. Pederiva, I., Vanderdonckt, J., España, S., Panach, I., and Pastor, O. The Beautification Process in Model-Driven Engineering of User Interfaces. In *Proc. of 11th IFIP TC 13 Int. Conf. on Human-Computer Interaction Interact'2007* (Rio de Janeiro, September 10-14, 2007), Lecture Notes in Computer Science, Vol. 4662, Springer-Verlag, Berlin, 2007, pp. 409-422.
16. Raskin, J. 2000. *The Humane Interface: New Directions for Designing Interactive Systems*. Addison-Wesley, Reading, Mass.
17. Lenorovitz, D.R.; Phillips, M.D.; Ardrey, R.S. & Kloster, G.V. "A taxonomic approach to characterizing human-computer interaction". In: G. Salvendy (Ed.), *Human-Computer Interaction*. Amsterdam: Elsevier Science Publishers, 1984, pp.111-116.
18. Linaje, M., Preciado, J.C., Sánchez-Figueroa, F., 2007. A Method for Model Based Design of Rich Internet Application Interactive User Interfaces. In *The Seventh Int. Conf. on Web Engineering (ICWE'07)*.
19. Nora Koch, Matthias Pigerl, Gefei Zhang and Tatiana Morozova. Patterns for the Model-based Development of RIAs. In *Proc. 9th Int. Conf. Web Engineering (ICWE'09)*, LNCS, volume 5648, pages 283-291. Springer, Berlin, June 2009.
20. Ceri, S., Fraternali, P., Matera, M., and Maurino, A.(2001). Designing Multie-Role, Collaborative Web Sites with WebML: a Conference Management System Case Study. *IWWOST'01*, Valencia, Spain, June 2001.
21. Matias Urbieto, Gustavo Rossi, Jeronimo Ginzburg, Daniel Schwabe. Designing the Interface of Rich Internet Applications. *Proc. 5th Latin American Web Congress (LAWeb' 07)*, pp.144-153, IEEE, 2007.
22. Valverde F., Pastor, O., Applying Interaction Patterns: Towards a Model-Driven Approach for Rich Internet Applications Development. . In *Proc. of 7th Int. Workshop on Web-Oriented Software Technologies IWWOST'2008* (Yorktown Heights, July 14th, 2008), L. Olsina, O. Pastor, D. Schwabe, G. Rossi, M. Winckler (eds.), CEUR Workshop Proceedings, Vol. 445, 2008.

OpenInterface: A Lightweight Open Source Platform for Rapid Prototyping of Multimodal Applications

Jean-Yves Lionel Lawson¹, Jean Vanderdonckt², Benoît Macq¹

¹Communications and Remote Sensing Laboratory (TELE)

²Belgian Lab. of Computer-Human Interaction (BCHI)

Université catholique de Louvain (UCL), Belgium

{jean-yves.lawson, jean.vanderdonckt, benoit.macq}@uclouvain.be

ABSTRACT

In this paper we present the *OpenInterface Kernel* as part of an open source platform for supporting the effective prototyping of multimodal interactive applications. Multimodal Interactive applications are based on the assembly of several components, namely, various and sometimes interchangeable modalities at the input, fusion-fission components and also several modalities at the output. Iterative design of such a system requires the easy integration, replacement or upgrade of components and the possibility to derive interaction properties from the component and basic ergonomic properties for the global system. We have designed a thin communication kernel able to manage this in an easy way by providing the research community a mean to solve a gap in the current support for multimodal applications implementation: *OpenInterface Kernel* is a practical light way to assemble various modalities with different implementation language, keeping a high level of performance of the assembled system. The effective connection of components requires their parsing in order to extract their communication features. Further parsing could be used to extract high level interaction properties. A running example illustrates the dynamic and extensible aspects of the platform. It runs on several operating systems and allows fast integration of interaction devices, signal processing tools, domain-independent data fusion, and dynamic runtime connection and re-configuration of interaction modalities.

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

General terms: Design, Experimentation, Verification.

Keywords: Prototyping, component-based architecture, reusable software component, multimodal interfaces, multimodal software architecture, integration techniques.

INTRODUCTION

Theoretical studies have highlighted the main issues at the development level of multimodal applications: content selection (“what to say”), modality allocation (“which modality to use to say it”), modality realization (“how to say

it in that modality”) and modality combination [22]. Criteria and requirements for the implementation of multimodal interactive systems are given in [34]; a precise way exists for describing, analyzing, and reasoning about multimodal systems prior to their implementation [29].

Currently, there is little research aimed at filling the gap between the design&specification stage and the implementation process of a functional system. There are existing solutions such as [9,13,28] for designing and implementing a multimodal system. They are limited, however, in the sense that they either present a small or hardly extensible number of input devices, they are platform and technology dependent or they do not provide a flexible prototyping environment for a large and heterogeneous number of research products, such as new devices prototype, new algorithms, etc. Prototyping is an important phase of multimodal application development process, as it allows designers to tackle the main issues presented above in an iterative fashion. A designer can *plug and play* with modalities, combine them, and quickly reuse the work done in a previous stage with little knowledge in low level programming. There is no software solution that presently provides this fast multimodal interaction prototyping feature.

In this paper we briefly present (Section 2) existing solutions for the design and implementation of multimodal applications by focusing on the dynamic and extensible aspects of the presented tools. Section 3 provides an overview of the *OpenInterface Platform*, an open source cross-platform software designed to support fast prototyping and implementation of interactive multimodal systems. Section 4 illustrates the platform’s dynamic features by realizing a simple multimodal application taking advantage of our platform functionalities.

RELATED WORK

There are several toolkits for investigating the design of multimodal applications. A few are listed here, and their main shortcomings are highlighted.

ICON

ICON [13] is an input toolkit that allows interactive applications to achieve a high level of input adaptability. It is implemented in Java and natively supports few input devices. Devices can be added to the toolkit, but it re-

quires a great amount of programming using JNI.

ICARE

ICARE [6] is also a component-based platform for building multimodal applications. This solution defines a new component model based on Java Beans, and requires all components to be in java. The platform is hard to extend, produces non-reusable components and also requires a lot of programming for integrating new devices or features.

CrossWeaver

CrossWeaver is a user interface designer's application for planning a multimodal application. It allows the designer to informally prototype multimodal user interfaces. This prototyping tool supports a limited number of input and output modalities and is not suitable for integrating additional software components.

QuickSet

QuickSet [9] mainly focuses on multimodality applied to the development of map-based systems. It uses speech and pen-based gesture as interaction modalities. The extension mechanism uses the Open Agent Architecture [10].

Max/MSP and PureData

Max/MSP [23] is a graphical environment for music, audio, and multimedia; PureData [30] is its open source counterpart. Both software provide a great number of design tools, are extensible and have a large community of users including performers, composers, artists, teachers, and students. The extension mechanisms, however, require low level programming in C and it is not possible to use the platforms without the graphical environment.

Most of the solutions listed above requires commitment to a given technology or supports a limited number of interaction modalities like voice, pen, text, and mouse, and are designed for specific interaction paradigms.

Our approach is different in the sense that we aim at providing a flexible solution for fast prototyping of multimodal applications by the easy and extensive reuse of existing software and technologies. The rationale for OpenInterface Platform Kernel (Figure 1) is therefore to be a generic platform for integrating heterogeneous code (device drivers, applications, algorithms, etc.) in a single software in a non intrusive fashion and with minimal programming effort. To support the design, implementation and reasoning on multimodal applications it will provide tools for composing, testing and debugging generated prototypes.

OPENINTERFACE PLATFORM

Requirements

Cross-platform, Heterogeneity, Reusability, Extensibility, and Performance

A major requirement of the platform is the ability to enhance and enforce the reusability of work previously achieved while implementing/prototyping new multimodal systems.

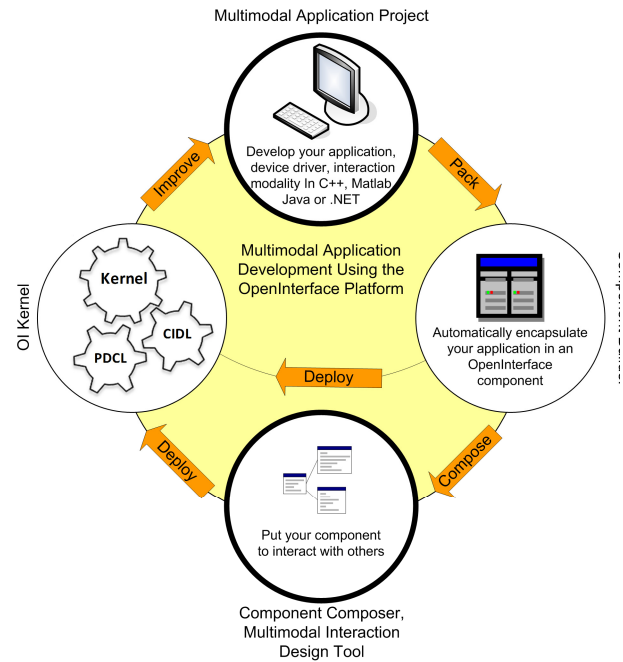


Figure 1. Multimodal Application Development Using the OpenInterface Platform.

Since the platform is intended for fast prototyping of new interaction models, it should also be easy to integrate and test immature signal processing research software like speech, gesture recognition – mostly implemented in high level prototyping languages such as Matlab, Python or scripting programming languages – as well as stable commercial devices provided with a low level API in C/C++ and graphical user interface software written in Java, Tcl/Tk, etc. To achieve good runtime performance of interactive systems, the platform must have good response time, despite the heterogeneous nature of the multimodal application being prototyped. One of the major differences between our approach and ICON is summarized by the heterogeneity requirement. It will be indeed a major advantage when it will come to rapidly integrate software and prototype interactions.

Multimodal Integration Support

Another important concept in multimodal application design and implementation is modality integration, or fusion. Two types of integration can be performed on modalities: feature fusion and semantic fusion. OpenInterface only implements domain independent fusion. Feature fusion is low level, domain-specific and performed between tightly coupled modalities (e.g. lips movement and speech), while semantic fusion is mostly high-level, preferably domain-dependent, and more related to the synchronization of time scale characteristics. Six basic types of cooperation between modalities are defined by [22]: Complementarity, Redundancy, Equivalence, Specialization, Concurrency and Transfer. A similar generic set of domain-independent combination of input modalities is

summarized in [12] where four roles are presented: Complementarity, Assignment, Redundancy, and Equivalence.

From a software point of view, all these generic roles can be expressed by a combination of simple data flow control operations. For instance, Complementarity can be implemented by temporal data synchronization, Assignment and Specialization by direct procedure call, Redundancy by connecting several modalities to the same functional module, etc. To facilitate the implementation of a functional multimodal system, the platform should embed a set of domain-independent fusion mechanisms but also offer a framework for adding new generic or tailored modality combination.

Plug and Play

Finally, our system should provide the ability to easily build a multimodal application by requiring as little programming effort as possible. Modalities, or software integrated within OpenInterface, should be easily assembled together in a plug and play fashion. This would allow for the creation of a network of components managing an advanced multimodal task, and provide the prototyping dimension to the system. Thus, the designers or programmers will then be able to quickly verify their model or specifications. The platform should therefore come with a significant set of application connectors, filters, debugging tools to ease the quick construction of an interactive multimodal system. An additional requirement is to not commit to a communication paradigm (event-based, procedure call, etc.), but rather provide tools capable of implementing the desired behavior. It will allow the platform to easily integrate with other software solutions.

Design

Heterogeneous Components

To achieve the reusability requirement, the platform adopts an extensible modular architecture where *components* are the base objects manipulated by the *OpenInterface Platform*. Our approach is very different from [6] in the sense that we do not define a new component model and we strive for minimal programming efforts when integrating new components. By not specifying an explicit model, it allows for flexibility by having the ability to implement a large set of models for multimodality. Additionally, at the implementation level, it enables communication with existing component models implementation like CORBA [11], EJB [15], and OSGi [28]. Components are unaware of the platform in which they are running; therefore, programmers use their preferred programming language and tools to build their piece of code, only declaring interfaces. Within our system, a *component* is only characterized by its interface and is defined as a *reusable and independent software unit with exported and imported Input/Output interfaces*. This definition is intentionally broad enough to encapsulate all models when implementing a multimodal system. Thus, a component must simply be software with the following mandatory attributes:

1. API (Application Programming Interface): there must be a way to communicate with the component services.
2. Installation/configuration: to facilitate the installation and configuration, the component should be packaged appropriately.
3. Documentation: the component must be well documented to enhance reusability.
4. No explicit dependencies with other components: a component must not make assumptions on the platform or other components existence or features. All required features must either be declared as imported, or packaged within the component.

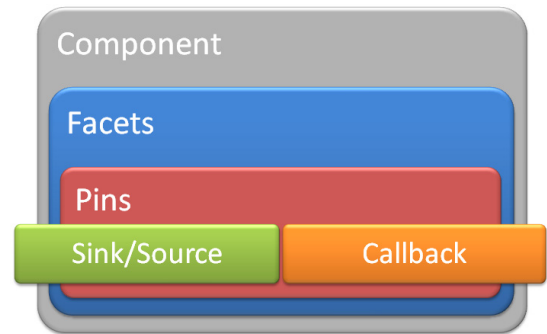


Figure 2. Component, OpenInterface view of any kind of external software.

OpenInterface provides tools to help producing such components. Figure 2 illustrates our view of a *component* as a bundled piece of software that provides a set of services/functionalities (*Facets*) which include input device drivers, signal-processing algorithms, fusion, fission, network communications, graphical interfaces, etc. These facets have well defined input/output interfaces called *Pins*. A pin can be a *Sink* (used to receive data), a *Source* (used to retrieve data from a component) or a *Callback* used to send data (and to import external functionalities into a component).

Pipelines

To build a running application, we introduce the concept of Pipeline as an interconnection and configuration of components as illustrated in Figure 3. It allows control over the components lifecycle and execution site (remote, local), and provides high level (threshold, filter, etc.) and low level (multicast, synchronization, etc.) data flow controls for building up a complex system.

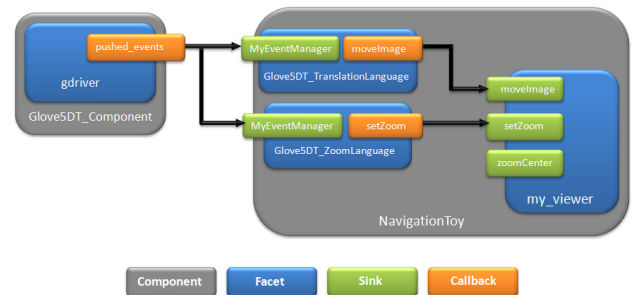


Figure 3. OpenInterface Pipeline.

Adapters/Connectors

Easily assembling components with a pipeline can only be done efficiently if the platform provides an extensible set of integrators which capture and mediate the communication between components. We use the concepts of Adapters/Connectors as entities that mediate interactions among components; that is, they establish the rules that govern component interaction and specify any auxiliary mechanism required [31]. Four categories fully describe the range of possible component interactions [24]: Communication, Coordination, Conversion and Facilitation.

In the platform, adapters/connectors are similar to components, with the difference being that their interface can either be generic or pre-defined. A generic interface can be connected to any components regardless of their interfaces, and Communication (TCP, RPC, etc.) and Coordination (Barrier, Synchronization, etc.) connectors fall into that category. Tailored connectors with well defined interfaces are called adapters in the platform, as they primarily serve the role of data conversion and facilitation.

Connectors are defined and designed so that they also include the definition of modality coordination, fusion. They are the basic tools for implementing semantic fusion (at the level defined by [22] and [12]) and communication paradigm (event-based, remote procedure call, pipe, etc.) within the platform.

Implementation

Heterogeneous Component Integration

Having components declare only their communication interface enforces the requirement for «independence». A component exports inputs/outputs to provide functionalities/services (display an image, device status, etc.) and imports inputs/outputs to request a feature provided by other components.

We have defined a description language called CIDL (Component Interface Description Language) that provides a way for components to declare their interface no matter the language in which they are written. The CIDL is semi-automatically generated from source code and is required by the OpenInterface Platform for manipulating a component. Figure 4 illustrates the syntax of the description language.

Once the CIDL description of the component has been made the platform actually generates code to encapsulate provided binaries into a well defined programming interface, Figure 5 illustrates that process. The encapsulated components can then be easily reused in any OpenInterface platform application in a plug and play fashion by using the pipeline description language presented in the following section.

Component Composition

A pipeline defines and configures connections between components and is described using the PDCL (Pipeline Description and Configuration Language [21]).

```

DirectXMouse.h
#ifndef _DirectInputMouse_H
#define _DirectInputMouse_H

#ifdef __cplusplus
extern "C"
{
#endif

#ifdef DLEXPORT
#define MYSPEC dllexport
#else
#define MYSPEC dllimport
#endif

__declspec( MYSPEC ) void startMouseCapture();
__declspec( MYSPEC ) void endMouseCapture();
__declspec( MYSPEC ) void setCb(void(*cb)(int dx,int dy,int screen_x,int screen_y,int bstate));

#ifdef __cplusplus
}
#endif

#endif

DirectXMouse.xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE Component PUBLIC "-//OpenInterface//DTD Component XML V0.1/EN"
"component.dtd">
<Component id="be.ac.ucl.tele.openinterface.component.win32.mouse.directx">
  <Name value="DirectXMouseComponent"/>
  <Language value="c++"/>
  <Container>
    <Name value="DirectXMouse"/>
    <Format value="so"/>
    <Location>DirectXMouse</Location>
  </Container>
  <IO>
    <Facet id="mdriver">
      <Bin>
        <CustomType type="cppclass" name=""
          def="DirectXMouse.h"/>
      </Bin>
      <Source id="pushed_events">
      </Source>
      <Sink id="StartMouseCapture">
      </Sink>
      <Sink id="StopMouseCapture">
      </Sink>
    </Facet>
  </IO>
</Component>

```

Figure 4. OpenInterface CIDL, Simplified Description (bottom) of a C/C++ Mouse Component (top).

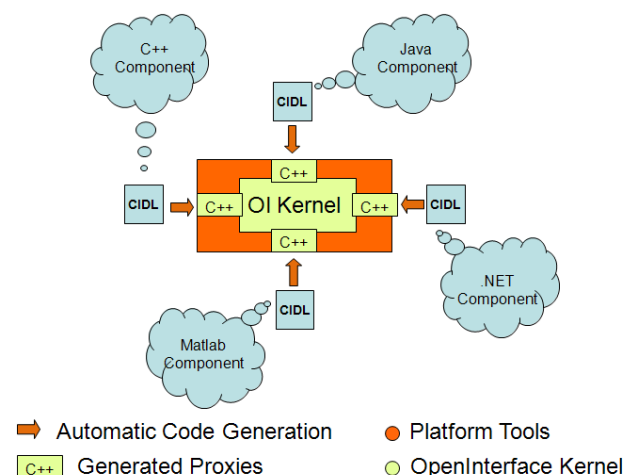


Figure 5. Heterogeneous Integration, Overview.

It provides simple embedded dataflow controls, such as direct function call and asynchronous call, but also ex-

poses a simple but powerful extension mechanism for extending the pipeline expressivity to simplify as much as possible intuitive interaction implementation. Currently, advanced flow control such as multicast (publisher/subscriber), Complementarity and Redundancy/Equivalence modalities fusion (temporal synchronization), data transformation like range filtering, rescaling, smoothing, etc. are distributed with the platform.

Overview

We have implemented the OpenInterface Kernel in C++ to achieve maximum performance, and also because most of the languages have C++ bindings. This allowed us to provide a portable platform capable of integrating heterogeneous software in a single application. An overview of how the platform works is illustrated by Figure 1, where each component is registered into the OpenInterface Platform using the Component Interface Description Language (CIDL). The platform then automatically generates proxies to perform the actual integration. Using an editor or the Kernel API, the user can edit the components properties and compose the execution pipeline (by connecting the components) of the multimodal application. This execution pipeline is either interpreted at once by the kernel or dynamically built by the application. More details on architecture and implementation can be found in [26]. According to the presented overview, the platform use can be decomposed into two main steps:

1. Integrate new modalities, devices, functional cores, i.e. components into the platform. The software can be provided in any of the supported programming languages (C/C++, Java, Matlab and .NET; more extensions can easily be added). This integration phase is well documented, and semi-automatic tools are provided to ease that process. It is usually performed by modality providers, i.e. programmers.
2. Use the pipeline interface to combine components and generate a running application. Currently there are two ways of using the pipeline:
 - Control the platform from the final application using the provided API and bindings. The API is in C++, but bindings for a large set of languages can be generated using the SWIG [4] wrapper. Java bindings are provided with the default installation as an example.
 - Use a development environment plugged to the platform. Currently there is one graphical editor, the OpenInterface Interaction Development Environment [16] (SKEMMI) which is a graphical tool, built on top of the platform, for the iterative design of a multimodal application. This tool is still in its alpha development phase, but is already available for download and evaluation.

This section presented the requirements, design, specification and implementation of the OpenInterface Kernel. We have stated that having such a tool as an implementa-

tion base for various models of multimodality shortens the development phase and provides better multimodal applications, thanks to a high rate of implementation refinement using prototypes. In the next section we demonstrate the dynamic functionalities of the OpenInterface platform with the construction of a simple multimodal music player prototype by reusing existing open source components.

DESIGNING A PROTOTYPE

In this part we prototype an application to provide the ability of navigating in a multimodal way through an MP3 collection and select a song for playback. Figure 6 illustrates a simple open source CD case image collection viewer [25] which has no other function than displaying a set of images in an aesthetically pleasing layout.

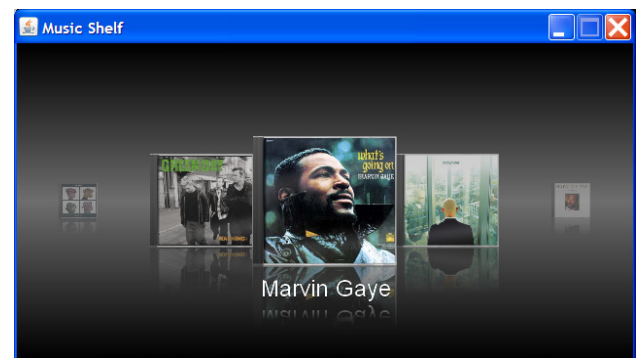


Figure 6. Java Music Shelf, a simple album art viewer.

For this application, we have integrated the following components inside the platform:

- A command-based MP3 player written in Java based on the JLayer library [18]
- A simple gauge level written in C# [8]

We also reused the following already integrated components:

- A Wii remote control (Wiimote) component which provides the device state (accelerometer, button, speaker and infrared points tracking) [35]

The integration process of existing code is fast and straight forward. It is thoroughly described in the platform online documentation.

First Iteration

In this first prototype, we build a pipeline as illustrated in Figure 7.

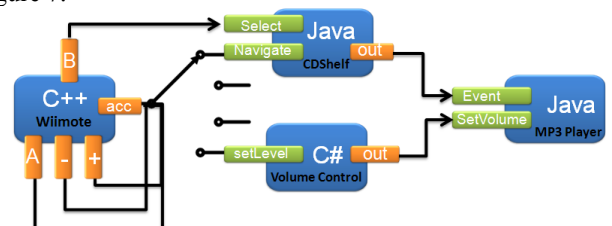


Figure 7. First Iteration of the Multimodal Music Player, Simple control with Wiimote.

The command-based Java mp3 player is the main application and has two interfaces for controlling the playback of a song (*Event*) and modifying the volume level (*SetVolume*). To provide an attractive graphical feedback to the user, the Music Shelf application (Figure 6) is used to display album art for each song, and a visual feedback for the volume level will be implemented by the meter level gauge (Figure 8). To control all of these visual components, we wanted to experiment with the Wii remote control. It is a powerful interaction device that comes with an integrated accelerometer, infra-red dots tracking capabilities, and several action buttons.

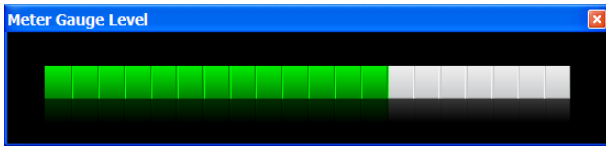


Figure 8. C# Meter Gauge Level for volume control visual feedback.

Task Model

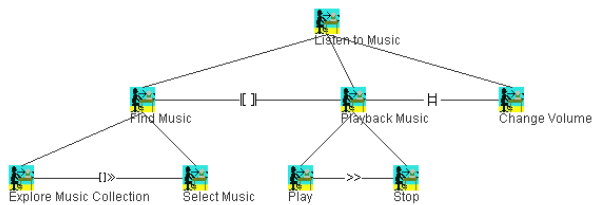


Figure 9. Multimodal Music Player Task Model.

Figure 9 illustrates the task model of our application where the user should be able to navigate through a collection of songs, select a song for playback, control the volume, and stop the playback.

Pipeline Implementation

Table 1 summarizes the *event-action* mapping we will perform to implement our task model.

Events	Actions
B Button pressed	Start a song playback
A Button pressed	Connect the Wiimote to the CD Shelf
+ Button pressed	Connect the Wiimote to the volume level control
- Button pressed	Disconnect the Wiimote
Acceleration (X)	Navigate through the songs collection or Increase/Decrease the volume

Table 1. Event-Action Mapping for the Multimodal MP3 Player

Applying that mapping at runtime, gives the user the option to disconnect the Wiimote from the application or connect it (Button A) to the CDSshelf component to navigate through the image or connect it to the volume level control (Button +). The SKEMMI can be used to generate the runtime configuration of our application, as illustrated by Figure 10.

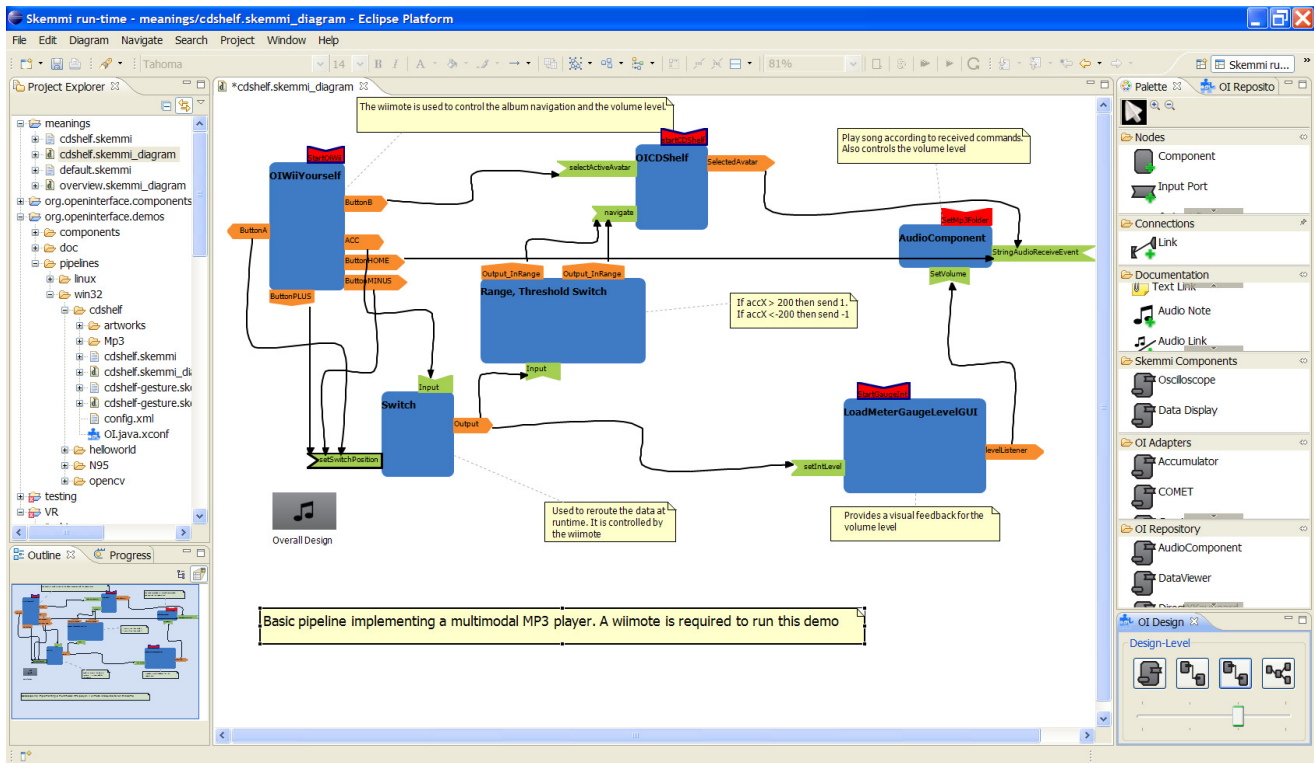
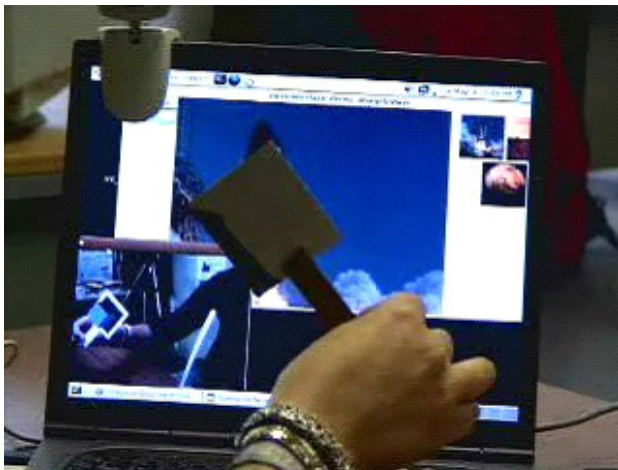
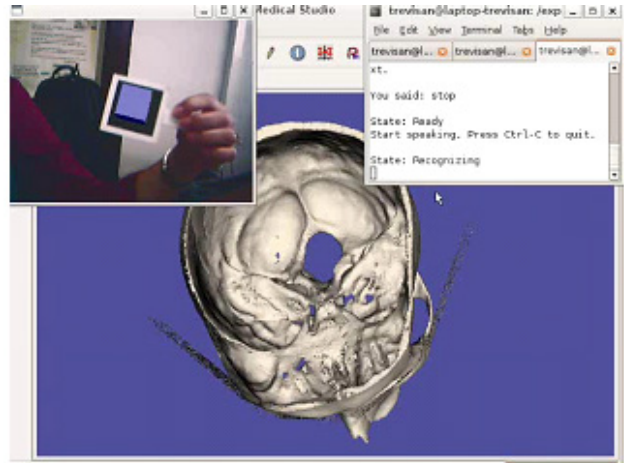


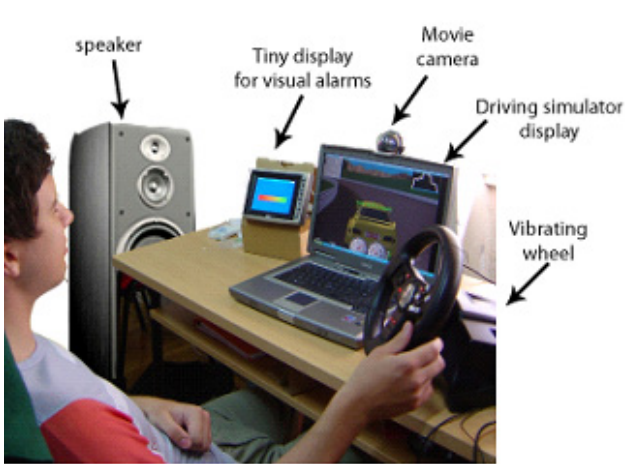
Figure 10. Design of the first iteration within SKEMMI.



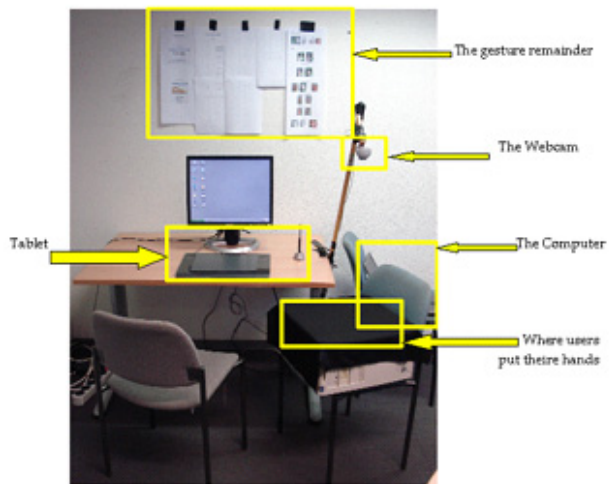
(a)



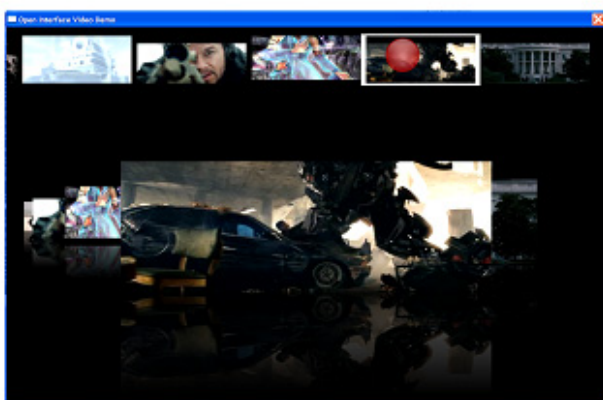
(b)



(c)



(d)



(e)



(f)

Figure 11. Selection of projects built with OpenInterface.

(a) First prototype, Image navigation; (b) Integration within MedicalStudio; (c) Hypovigilance state detection using a multimodal driver simulator; (d) InterpiXML integration; (f) 3D Game on Nokia N95 testbed.

The application can also be generated on the fly using the runtime API. This last solution is useful when the final prototype has to be a standalone application, therefore not relying on a graphical editor to compose connections.

In this first iteration, the accelerometer is too sensitive and triggers many events. To reduce the noise, we simply use a threshold connector between the Wiimote and the CDShelf component. The filter then only sends an event when its value (acceleration) is outside a predefined range; in this case we use $[-200, 200]$. This simple example has illustrated how easy it is to integrate modalities and combine them in a single application. The power of the OpenInterface Platform is demonstrated here in the sense that the Wiimote can easily be replaced by any combination of interaction devices and modalities.

In the next section, we describe some applications designed and implemented using the OpenInterface Platform, and give a short overview of the growing and easily extensible component database.

APPLICATIONS

The OpenInterface Platform has been used, and is being used, in several research projects for successfully designing, prototyping, and implementing multimodal applications. Some of them are described below:

Multimodal Driving Simulator

The OpenInterface platform was first used at eNTERFACE05 [14] a four-week workshop organized by the SIMILAR European Network of Excellence, aiming at establishing a tradition of collaborative, localized research and development work by gathering, in a single place, a group of senior project leaders, researchers, and (undergraduate) students, working together on a pre-specified list of challenges. The platform was still in its alpha state, being used as the backbone of a multimodal driver simulator [5], and working as the integration platform for combining the driver state based on video data (e.g., facial expression, head movement, eye tracking) and multimodal input. The setup of this application is illustrated by Figure 11 (c).

MedicalStudio

MedicalStudio [33] is a composable, open source and easily evolvable cross-platform framework that supports surgical planning and intra-operative guidance with augmented interactions. It is designed to integrate the whole computer aided surgery process to which both researchers and clinicians participate. OpenInterface has been used as an integration platform to study speech and marker detection interaction for 3D objects manipulation as illustrated by Figure 11 (b).

UsiXML Interpreter

UsiXML [35] (which stands for USeR Interface eXtensible Markup Language) is a XML-compliant markup language that describes the UI for multiple contexts of use such as Character User Interfaces (CUIs), Graphical User Interfaces (GUIs), Auditory User Interfaces, and Multi-

modal User Interfaces. InterpiXML is a runtime interpreter for UsiXML files and supported a limited number of modalities. The interpreter has been integrated as a component inside within OpenInterface so that it could use the platform as a runtime environment for controlling and combining a larger set of modalities. Figure 11 (d) illustrates a usability case study conducted using the described configuration.

Multimedia Information retrieval

The IRMA project [17] aims at designing and developing a modular, customized, efficient, protected and economically viable interface for multimodal searching and browsing in indexed audiovisual databases. The platform is being used for investigating multimodal interaction for navigating in a large set of multimedia data including annotated video, sound, and text. Figure 11 (e) represents a multimodal video viewer controlled with finger gesture and voice. The finger interaction modality has been implemented using infra red dot tracking.

OpenInterface Project

The OpenInterface Project [27] is an ongoing European IST Framework 6 Specific Targeted Research Project funded by the European Commission. It promises:

1. To ground the development of multimodal interactive systems in a scientific understanding of multimodal interaction. In our case, this will be achieved through reusable software components in the platform that are directly defined from theoretical results on multimodality.
2. To provide a tool for implementing a truly iterative user-centered design process.
3. To turn the results into industrial standards by way of the platform.

Several demonstrators have been implemented, including a multimodal map navigation and a mobile multimodal game. A typical setup for the latter is illustrated by Figure 11(e) where the game runs on a mobile phone (Nokia N95), while the different interaction modalities are running within the platform on a PC. This configuration has allowed Arcadia Design [3] designers to investigate different modalities to control their game: speech commands, 3D gesture commands with the Shake [36] and 3D gesture commands with ARToolkit [1] have been tested so far.

Component Database

Figure 12 synthesizes the current state of the platform component database and Figure 13 presents the database of connectors that can be used to compose components. The presented components are highly reusable and come from the various projects involved in OpenInterface. Tailored components, such as complete applications (games, functional cores, etc.) have not been listed because of their limited reusability in the field of application design.

	Connectors
Communication	•OSC •Bluetooth •TCP •Switch •Multicast
Operation	•Generic Filter •Smooth •Range Filter •Threshold •Rescale •Conditional •Delta
Fusion	•Complementarity •Redundancy/Equivalence

Figure 12. Overview of the OpenInterface Connector Database.

	Input	Output
Gestual	•Sketch Recognizer •Gesture Recognizer •Head Tracker (OpenCV) •Wiimote •Finger Tracking •Data Gloves (5DT, etc.) •TouchPad (Synaptics) •Face Tracker •Infra Red Tracking (OptiTrack) •Stylus •Hand Posture (HandVU) •Mouse, Keyboard •ARToolkit •...	•Wiimote •Shake
Visual	•Generic Pie Menu •Camera(Firewire, USB Webcam)	•VirtualMouse •Virtual Pointer
Vocal	•Speech Recognizer	

Figure 13. Example of components currently integrated and available from OpenInterface component database.

The database is already well furnished in input devices and a large number of combinations could be prototyped for studying new interactions. It also highlights the lack of output devices and interaction techniques; this provides us hints for future work, and directions to explore when designing innovative interaction modalities. This database is available from <https://forge.openinterface.org>. Specific licensing scheme may apply to each component, depending on the original binary used to perform the integration.

CONCLUSION AND FUTURE WORK

Going from a multimodal application design to a running implementation is not a simple task due to the lack of tools supporting this process. We have presented a runtime platform, the OpenInterface Kernel, which enables easy reuse of existing work to iteratively design and build a multimodal system with minimal programming effort. We have briefly illustrated the process of building a dynamic multimodal application – an MP3 player application controlled with the Wii remote control– by reusing existing open source software. The platform, as presented here, is available for download as an open source platform from <https://forge.openinterface.org>. Currently, there is one graphical front-end to the kernel, the SKEMMI – OpenInterface Interaction Development Environment – which focuses more on the designer view when developing multimodal application. However, the modular approach followed by the kernel allows it to be interfaced by any kind of front-end application which can output XML description. We have already successfully tested integration PureData and ICARE; in the near future we will investigate the integration of the platform within solutions like CrossWeaver, ICON and with existing open source interactive applications workflow editors such as *EyesWeb* [7] and *Clam Network Editor* [2]. Our focus is also directed toward enriching the platform with a comprehensive set of tools that will allow us to automatically generate multimodal interface prototype from high level description languages like UsiXML [35] or UIML [1] of interactions which is abstract from a particular implementation and allows to integrate basic rules of usability in the iterative design process.

ACKNOWLEDGMENTS

The work on the OpenInterface platform was partly funded by the European FP6 SIMILAR Network of Excellence (www.similar.cc) and is now mainly funded by the European FP6-35182 OpenInterface (www.oi-strep.com). J. Vanderdonckt also acknowledges the support of the Région Wallonne for ITEA2 Call 3 UsiXML project, under reference 2008026.

REFERENCES

1. Abrams, M., Phanouriou, C., Batongbacal, A.L., Williams, S.M., Shuster, J.E. UIML: An Appliance-Independent XML User Interface Language. *Computer Networks* 31,11-16 (1999), 1695–1708.
2. Amatriain, X., Arumi, P., Garcia, D. 2006. CLAM: a framework for efficient and rapid development of cross-platform audio applications. In *Proc. of ACM Multimedia'2006* (Oct. 23-27, 2006). ACM Press, New York, pp. 951–954.
3. Arcadia Design, www.arcadiadesign.it
4. Beazley, D.M. SWIG: an easy to use tool for integrating scripting languages with C and C++. In *Proc. of the 4th Conf. on USENIX Tcl/Tk Workshop* (July 10 - 13, 1996, Monterey, CA). USENIX Assoc., Berkeley, 1996, 15.

5. Benoit, A., Bonnaud, L., Caplier, A., Damousis, I., Tzovaras, D., Jourde, E., Nigay, L., Serrano, M., Lawson, J-Y. Multimodal Signal Processing and Interaction for a Driving Simulation: Component-based Architecture. *J. on Multimodal User Interfaces 1*, 1 (2007), 49–58.
6. Bouchet, J., Nigay, L. ICARE: A Component-Based Approach for the Design and Development of Multimodal Interfaces. In *Extended Proc. of CHI'04* (April 2004, Vienna). ACM, NY, 2004, pp. 1325–1328.
7. Camurri, A., Ricchetti, M., Trocca, R. EyesWeb - toward gesture and affect recognition in dance/music interactive systems. In *Proc. of IEEE Multimedia Systems '99* (June 1999, Firenze). IEEE Comp. Press, Los Alamitos, 1999.
8. CodeProject, www.codeproject.com
9. Cohen, P.R., Johnston, M., McGee, D., Oviatt, S., Pittman, J., Smith, I., Chen, L., Clow, J. QuickSet: multimodal interaction for distributed applications. In *Proc. of ACM Multimedia '97*, 1997, pp. 31–40.
10. Cohen, P.R., Cheyer, A.J., Wang, M., Baeg, S.C. An Open Agent Architecture. In *Proc. of AAAI Spring Symposium* (March 1994). AAAI (1994), pp. 1–8.
11. Common Object Request Broker Architecture, www.corba.org
12. Coutaz, J., Nigay, L., Salber, D., Blandford, A., May, J., Young, R. Four Easy Pieces for Assessing the Usability of Multimodal in Interaction the CARE Properties. In *Proc. of IFIP Interact'95*. IOS Press (1995) pp. 115–120.
13. Dragicevic P. and Fekete J-D. Input Device Selection and Interaction Configuration with ICON. In A. Blandford, J. Vanderdonkt, P. Gray (eds.), *Joint proceedings of IHM 2001 and HCI 2001 (IHM-HCI'01)*. Springer Verlag, London, 2001, pp 543-558.
14. eINTERFACE Workshops, <http://www.enterface.net/>
15. Enterprise JavaBeans Technology. <http://java.sun.com/products/ejb>
16. Gray, P., Ramsay, A., Serrano, M. (2007). A Demonstration of the OpenInterface Interaction Development Environment. In *Adjunct Proc. of UIST'07*. ACM Press, New York, pp. 39-40.
17. IRMA: Interface de Recherche Multimodale en contenu Audiovisuel, <http://www.irmaproject.net>
18. JLayer, MP3 library for the Java Platform. <http://www.javazoom.net/javalayer/javalayer.html>
19. Kato, H., Billinghurst, M. (1999) Marker Tracking and HMD Calibration for a video-based Augmented Reality Conferencing System. In *Proc. of the 2nd Int. Workshop on Augmented Reality IWAR '99* (October 1999, San Francisco)
20. Katsurada, K., Sato, K., Adachi, A., Yamada, H. and Nitta, T. 2005. A Rapid Prototyping Tool for Constructing Web-based MMI Applications. In *Proc. of INTERSPEECH2005*, pp.1861-1864.
21. Lawson, J-Y., 2006. Openinterface description languages specification. Technical report, TELE-UCL, 2006. Available online: <http://www.openinterface.org/platform/documentation>
22. Martin, J. C., TYCOON: Theoretical Framework and Software Tools for Multimodal Interfaces. In *Intelligence and Multimodality in Multimedia Interfaces*, AAAI Press, 1997.
23. Max/MSP. Available on-line at: <http://www.cycling74.com/products/maxmsp>
24. Mehta, N. R., Medvidovic, N., and Phadke, S. 2000. Towards a taxonomy of software connectors. In *Proc. of the 22nd Int. Conf. on Software Engineering ICSE '2000* (Limerick, June 4-11, 2000). ACM Press, New York, pp. 178-187.
25. Music Shelf in Java2D, Available on-line at: http://www.jroller.com/gfx/entry/a_music_shelf_in_java2d
26. OpenInterface Association, www.openinterface.org
27. OpenInterface European project. IST Framework 6 STREP funded by the European Commission (FP6-35182). www.oiproject.org.
28. OSGi Alliance. www.osgi.org
29. Palanque, Ph. and Schyn, A. 2003. A Model-Based Approach for Engineering Multimodal Interactive Systems. In *Proc. of IFIP TC 13 Int. Conf. on Human-Computer Interaction INTERACT'03* (Zurich, 2003) p. 543.
30. Puckette, M. 1996. Pure Data: another integrated computer music environment. In *Proc. the 2nd Inter-college Computer Music Concerts*, Tachikawa, pp. 37-41
31. Shaw, M. and Garlan, D. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice-Hall, 1996.
32. Sinha, A.K., Landay, J.A. Capturing user tests in a multimodal, multidevice informal prototyping tool. In *Proc. of the 5th Int. Conf. on Multimodal Interfaces ICMI'2003* (Vancouver, November 5-7, 2003). ACM Press, New York, pp. 117-124.
33. Trevisan, Daniela G.; Nicolas, Vincent; Macq, Benoit; NEDEL, Luciana P. MedicalStudio: a medical component-based framework. In *Proc. of Workshop de Imagens Médicas WIM'2007* (Recife, 2007). Anais do Workshop de Imagens Médicas (em CD), 2007.
34. UIMS Tool Developers Workshop. A Metamodel for the Runtime Architecture of an Interactive System. *SIGCHI Bulletin*, 24(1), January 1992, pp32-37.
35. Vanderdonckt, J. Model-Driven Engineering of User Interfaces: Promises, Successes, and Failures. In *Proc. of 5th Annual Romanian Conf. on Human-Computer Interaction ROCHI'2008* (Iasi, September 18-19, 2008), S. Buraga, I. Juvina (eds.). Matrix ROM, Bucarest, 2008, pp. 1–10.
36. Wiio, The wiimote C library. www.wiio.net
37. Williamson, J., Murray-Smith, R., and Hughes, S. 2007. Shoogle: Excitatory Multimodal Interaction on Mobile Devices. In *Proc CHI '07*. ACM Press, New York, pp. 121-124.

Pattern-Based Design of User Interfaces in UsiXML

Francisco Montero¹ and Jean Vanderdonckt²

¹Laboratory of User Interaction & Software Engineering (LoUISE)
University of Castilla-La Mancha, 02071 – Albacete (SPAIN)

fmontero@dsi.uclm.es

²Université catholique de Louvain, Louvain School of Management
Place des Doyens, 1 – B-1348 Louvain-la-Neuve (Belgium)

jean.vanderdonckt@uclouvain.be – www.isys.ucl.ac.be/bchi/members/jva

ABSTRACT

This paper introduces a knowledge-based method for developing user interfaces based on generative patterns instead of descriptive patterns. The knowledge base contains generative patterns from which portions of previously designed user interfaces could be identified and re-applied to a new design case study by generating code from its functional specifications. The method introduced in this paper is relying on models that are typically involved in user interface development such as task, domain, abstract user interface, concrete user interface, final user interface, context model, and mappings between them. In this way, any type of model can virtually be the source of a pattern and can be described, searched, matched, retrieved, and assembled together so as to create a new user interface. The method is supported by IDEALXML, a software that can be used for pattern-based design of user interface based on the USiXML user interface description language. In order to support pattern-based design, the USiXML language has been expanded with concepts addressing problems raised by pattern description and matching related to models pertaining to user interface development. This extension could also be considered as an extension of PLML (Pattern Language Markup Language) that has been introduced to uniformly represent user interface patterns.

Author keywords

Descriptive pattern, generative pattern, model-driven engineering.

INTRODUCTION

Since a more than two decades, design patterns [1, 3, 13, 30] have received much attention in various domains of the human activity, including software engineering [8], software development [5], and User Interface (UI) design [15, 33] with the conviction that parts or whole of any UI that has been designed for a past interactive application may be reused later in another, perhaps similar, application. In addition, design patterns are also frequently expressed as a comprehensive way to communicate pairs of (problem, solution) in a manner that remains largely applicable, and more general than usability guidelines [34]. In particular, usability guidelines were criticized for losing the context where they are applicable or for not providing this context explicitly for applying the guidelines [3].

The CHI'2003 workshop on UI Patterns [12] observed that many different, probably inconsistent, sources of UI design patterns exist today [10,11,29,33], thus raising the need for a common pattern language to express UI design patterns. This resulted into the Pattern Language Markup Language (PLML) [12] specification.

The main goal of PLML was to bring some structure and consistency to the many forms that have been used by pattern authors. PLML became more widely applied as several pattern collections have been translated into this format, thus facilitating comparison, reuse, and linking between various collections. But, PLML is a natural language-based way for writing patterns. Since any form of natural language always suffers from intrinsic problems like ambiguity, inconsistency, PLML did not escape from these problems that prevent this language from being directly used as a pattern format for a pattern-based UI design process [15] that is effectively and efficiently supported by software.

Lack of expressivity: several PLML tags express various aspects of a pattern that were believed of sufficient general interest, but some are missing. For instance, a tag describes the forces of a pattern, but no other tag describes the counter-forces. Flat definition: PLML is defined in a Document Type Definition (DTD) in a flat structure that does not allow structured pattern-matching and searching.

Lack of separation of concerns: the current PLML definition mixes the expression of several concepts together, thus violating the principle of separation of concerns where different aspects are captured in different independent models. For instance, the context definition is completely embedded in a general tag without being further refined. Therefore, it is not guaranteed that various patterns, if they are relevant to a same context, will be expressive enough to be found out for a similar context of use, unless the person who encodes the context description devotes a lot of attention.

Lack of structure: several tags are only defined in a general way (e.g., a string), with no further decomposition, thus leaving the definition very open and flexible (which is an advantage), but hindering a structured use of the tags by a software (which is a shortcoming for large and efficient use). Fig. 1 shows that this flat structure does not facilitate much rigorousness.


```

<!ELEMENT pattern (name?, alias*, illustration?, problem?,
context?, forces?, solution?, synopsis?, diagram?, evidence?,
confidence?, literature?, implementation?, related-patterns?, pat-
tern-link*, management?)>
<!ATTLIST pattern patternID CDATA #REQUIRED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT alias (#PCDATA)>
<!ELEMENT illustration ANY>
<!ELEMENT problem (#PCDATA)>
<!ELEMENT context ANY>
<!ELEMENT forces ANY>
<!ELEMENT solution ANY>
<!ELEMENT synopsis (#PCDATA)>
<!ELEMENT diagram ANY>
<!ELEMENT evidence (example*, rationale?)>
<!ELEMENT example ANY>
<!ELEMENT rationale ANY>
<!ELEMENT confidence (#PCDATA)>
<!ELEMENT literature ANY>
<!ELEMENT implementation ANY>
<!ELEMENT related-patterns ANY>
<!ELEMENT pattern-link EMPTY>
<!ATTLIST pattern-link type CDATA #REQUIRED
patternID CDATA #REQUIRED
collection CDATA #REQUIRED
label CDATA #REQUIRED>
<!ELEMENT management (author?, credits?, creation-date?,
lastmodified?, revision-number?)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT credits (#PCDATA)>
<!ELEMENT creation-date (#PCDATA)>
<!ELEMENT last-modified (#PCDATA)>
<!ELEMENT revision-number (#PCDATA)>

```

Figure 1. Document Type Definition of PLML.

In order to address these shortcomings, a method for developing a UI based on patterns is introduced:

1. A definition of models involved in UI design, which can then be mapped to a pattern.
2. A specification of these models and the pattern according to a single and consistent User Interface Description Language (UIDL).
3. A definition of the method steps with these models.
4. A software for supporting the method called IDE-ALXML (Interface Development Environment for Applications specified in UsiXML).

RELATED WORK

A pattern must be useful because this shows how having the pattern in mind may be transformed into an instance of the pattern in the real world [1], as something thing that adds value to our lives as developers and practitioners. A pattern must also be usable because this shows how a pattern described in literary form may be transformed into a pattern that we have in our mind. And a pattern must be used because this is how patterns that exist in the real world first became documented as patterns in literary form. In the next subsections, we discuss how UI patterns have been tried to become useful, usable, and used.

Patterns compilation

We can find many references where design patterns in Human-Computer Interaction (HCI) or interaction patterns appear. Compilations of those references can be found in, for instance, the interaction design patterns page [10], The pattern gallery [11], HCI patterns pages [4] or Interaction design patterns [33]. In those compilations, several ways of documenting the same type of contents can be identified from natural language to XML-based formats. Manipulating interaction patterns is very difficult and is necessary to provide additional assistance in order to use them in a (semi-automatically) way. In this sense, only a few proposals are available where designers can work using patterns. In software engineering, notations exist like UML and tools where design patterns [14] can be used together. Design patterns are documented using class diagrams from which guidance is provided to designs on how to use them.

These tools are not available in other fields like HCI because UI patterns in this field are difficult to use, to document, to compare, and to know. Using UI patterns typically requires assistance for identifying, selecting, adapting, and integrating them. These tasks should be supported by tools to become really usable. For this purpose, the pattern documentation should be improved prior to making them available in tools. Using only natural language is not enough in order to work efficiently with patterns. Other forms exist. Thus, there is no universal way to write a pattern.

Patterns are often referred to as being descriptive when they basically consist of a description of the pattern, its problem, the context in which the problem is posed, and the potential solutions that can be brought to solve the problem. Patterns are one form of establishing a mapping between the problem space and the design space. Descriptive patterns are intended to be used mainly by human such as project leaders, designers, analysts, and developers. Descriptive patterns usually seek to maximize descriptivity (i.e., the ability of a pattern to be described in details enough to become self-contained) and genericity (i.e., the ability of a pattern to be applicable to the widest problem space possible by interpreting the description for a particular context of use). As opposed to descriptive patterns, patterns are said to be generative when they subsume an object-oriented representation that can be automatically obtained in order to generate the final code.

Generative patterns are intended to be used by automata (e.g., algorithms, program analysis and synthesis techniques). Generative patterns usually seek to maximize expressivity (i.e., the ability of a pattern to be expressive enough so as to obtain a working system) and generativity (i.e., the ability of pattern to be expressed in a way that facilitates automated generation of code).

In our context, generative patterns tell us how to create a UI that can be observed in the resulting interactive system to be developed. Non-generative patterns describe recur-

ring phenomena without necessarily saying how to reproduce or to concretize them in a particular interactive application. We should therefore strive to document generative patterns because they not only show us the characteristics of good UIs (e.g., they could also convey information about usability [34]) that are appropriate in their context of use, but they also teach us how to build them or, in other words, to develop them. This does not mean that descriptively should be left out. We here argue for a UI pattern scheme that combines both the qualities of descriptive and generative patterns by defining a UI pattern template containing both descriptive and generative aspects as opposed to one single dimension at a time.

Therefore, we are expecting to introduce genuine patterns that will maximize all the four properties of purely descriptive and generative patterns. Indeed, descriptive patterns are usually estimated of high genericity and descriptivity, but weak in generativity and expressivity (Fig. 2). Generative patterns are located in an inverse situation: they are rich in generativity and expressivity, but weak in genericity and descriptivity. By combining the qualities of both families of patterns into the genuine patterns, we need to reach a high level for all four properties simultaneously (Fig. 2).

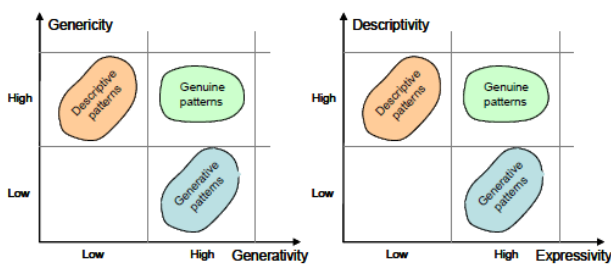


Figure 2. Classification of patterns compilations according to the four properties

Pattern software

Different software exists today for supporting the process of using UI patterns. Environments exist where patterns can be introduced [29], suggested [16], viewed [33] or used to develop prototypes [26].

CanonSketch [7] is a tool to describe user interfaces using the notation of Canonical Abstract Prototypes [9]. Introducing a UI using this notation which is independent of any technology represents a generative pattern since HTML code can be automatically generated from the description. However, no other information about the pattern is provided. The Montreal Online Usability Patterns Digital Library [29] is an Integrated Pattern Environment (IPE) that was originally designed with two major objectives: as a service to UI designers and software engineers for UI development and as a research forum for understanding how patterns are really discovered, validated, used and perceived.

MOUDIL consists of a pattern editor, a pattern navigator and a pattern viewer. In this way, it supports descriptive

patterns effectively, but needs to be connected with other tools to give rise to a running UI.

Greene [16] developed a software prototype to support pattern-assisted design and development. The software supports the pattern creating, browsing, viewing, and editing, but most importantly, it provides decision support to help filter and select patterns based on criteria or drivers specified by the pattern authors as relevant to particular patterns. Internally, patterns are stored as XML documents. Pattern elements are the fields or properties of the patterns (e.g., 'Name', 'Problem', 'Forces', 'Context', 'Solution', etc.). There is a default set of such properties, but, since there is of yet no accepted standard set of properties, this set is definable and extensible by the pattern language author.

One can define different pattern types with different fields and links between patterns may be user defined and typed, thus providing mechanisms that are adequate for making a true knowledge base of patterns.

One can search for patterns that contain specified strings in all, or any subset of the fields of the patterns. Although there are currently two decision support mechanisms embodied in the tool to identify appropriate patterns, it does not produce any running UI.

MESCA [18] consists of a knowledge base of UI elements that are considered as patterns. Its advantage relies in its case-based reasoning algorithm for finding out similar UI elements based on search criteria. Again, it does not produce any running UI. The PIM tool [25] probably represents the most advanced tool for UI patterns which are both descriptive and generative: it stores models in the XIML (www.ximl.org) and allows several degrees of pattern searching.

Methodologies

In the area of Model-Driven Engineering (MDE), several methodologies exist that support the development life cycle of interactive applications, such as UML-based methodologies. WISDOM [7] or IDEAS [23] are object-oriented, they use the UML to specify, visualize, and document the artifacts of the development project. They have been adapted to develop interactive applications because UML does not support UI design.

WISDOM and IDEAS evolve incrementally through an iterative process. Other approaches are task centered [26], pattern-oriented [17, 23, 27, 30], involve different techniques such as usability engineering [17], interaction templates [26], multiple design [27], and MDE [28,30]. They provide a methodological guidance on how to use patterns but, again, are not generative. A major observation is that a UI pattern may be informed by many different types of contents belonging to different models which are not all necessary at once, but which could be considered individually when needed.

Next, we introduce our representation of a UI pattern so that it is both descriptive and generative.



The PLML [12] language, resulting from a consensus obtained during the CHI'2003 workshop on patterns, is certainly a reference base to be considered for extension. Therefore, by relying on specifications as described in Fig. 1, we extended PLML into a UML Class Diagram for representing UI patterns that are both descriptive and generative (Fig. 3). We now justify why these extensions have been required.

But these forces should be counterbalanced with other dimensions which are typically found in the SWOT analysis, a tool for auditing an organization and its environments with four axes: strengths, weaknesses, opportunities, and threats.

control, adaptation, guidance, and error management [2]). By expressing which ergonomic criteria are respected (or addressed), we know in advance the quality of pattern and their purpose. If we want to maximize consistency, patterns related to consistency could be selected from the knowledge base.

- 5: two or more experiments support the pattern.
- 4: one experiment supports the pattern.
- 3: two or more studies support the pattern.
- 2: one study supports the pattern.
- 1: one or more observations and no other supporting evidence support the pattern.
- 0: no evidence supports the pattern.

- X *uses* Y in its solution,
- X *is a variant of* Pattern Y,
- X *has a similar problem as* Y,
- X *is related in the related patterns section to* Y,
- X *specializes* Y (in the sense of pattern inheritance),

- *X connects to Y as part of the sequence S*, in this case, the label includes S and a descriptive text that serves as the glue text in the sequence,
- *X mentions Y* in its context, this means that *Y was applied before Y*,
- *X and Y are members of the same class or family*,
- *X and Y involve a common participant P*, and
- *X and Y can be found in the same known context of use U*.

The problem provides a description of the problem space covered by the pattern while the space attribute describes the solution space ensured by the pattern. Another factor of confidence we can assign to a pattern comes from the bibliographic reference (*reference*) where it is defined: a pattern defined by an organization, an expert or a practitioner may widely differ in its scope and purpose. For instance, a pattern recommended by an official standardization body could be considered as stronger than a pattern provided by an individual person.

It is very important to document examples showing the application of a pattern so as to facilitate its interpretation and its application [34]. Therefore, example contains a description of a supportive example demonstrating the applicability, the non-applicability, or an exception of the pattern. Each example could be associated, if needed, to one or several domains of human activity (*humanActivity*) that characterize whether a pattern is generic or specific to a domain. In this way, it is also possible to search the knowledge base of patterns for patterns that are applicable to a particular domain, say for instance chemistry, medical record of patient, museum visits, etc. This concludes the upper part of Fig. 3 containing the descriptive explanatory power of a UI pattern. The below part of Fig. 3 represents the generative power as it relates the pattern to any combination of UI models involved in the Cameleon Reference Framework [6] for developing multi-target UIs, which is decomposed into four steps [6, 31, 32]:

1. *Task and domain modeling* (Computing Independent Model in MDA): a model is provided for the end user's task, the domain of activity and, if needed, the context of use (user, computing platform, and environment).
2. *Abstract User Interface modeling* (Platform Independent Model in MDA): this level describes potential UIs independently of any interaction modality and any implementation technology.
3. *Concrete User Interface modeling* (Platform Specific Model in MDA): this level describes a potential UI after a particular interaction modality has been selected (e.g., graphical, vocal, multimodal). This step is supported by several tools helping designers and developers to edit, build, or sketch a user interface.
4. *Final User Interface*: this level is reached when the UI code is produced from the previous levels. This code could be either interpreted (in this case, UI rendering is

ensured) or compiled (in case, various techniques such as generative programming, template-based approach, static code generation could be used).

Our methodology enables expressing and executing model transformation based on UIs viewpoints. For this purpose, the mapping model links the various models resulting from the above steps through mappings [6]:

- *Reification* is a transformation of a high-level requirement into a form that is appropriate for low level analysis or design.
- *Abstraction* is an extraction of high-level requirement from a set of low-level requirements artifacts or from code.
- *Translation* is a transformation a UI in consequence of a context of use change. The context of use is, here, defined as a triple of the form (U, P, E) where E is an possible or actual environment considered for a software system, P is a target platform, and U is a user category.
- *Reflection* is a transformation of the artifacts of any level onto artifacts of the same level of abstraction, but different constructs or various contents.

USING UI PATTERNS WITH IDEALXML

To support the usage of UI patterns as defined in Fig. 3, the IDEALXML software has been developed in Java code. It can exploit a knowledge base of UI patterns stored in UsiXML language [32] (www.usixml.org). This UIDL has been selected because it already covered the various models involved in the below part of Fig. 3.

The upper part has therefore been equally defined so that it could be expressed in a XML format that is compliant with UsiXML. In order to illustrate how this software can support the four-step method outlined above, let us consider an example related with web design and development: the Sedan-Bouillon web site (<http://www.sedan-bouillon.org/>) is a web site for providing tourists with location-aware information on the archeological site. Fig. 4 shows a screen shot where tourist guides could be ordered on-line. This web page is a form where the user can ask until three different catalogs related with tourist information of this French region. This request is considered as a transaction that a visitor (*participant*) establishes when he visits this website. This participant should be provided with additional information in order to receive these catalogs.

Finally the user should send his request pressing send button. In order to design our application at least three models should be considered: domain, task, and abstract UI models before reaching a final UI. To do that different elements are used: class diagrams for the domain model, ConcurTask-Tree notation [23] for the task model, and Abstract Interaction Objects (AIOs) for the abstract UI model. For doing this purpose, we can use patterns for each model. So, we can identify three classes in our diagram of classes: partici-

pant, transaction and catalogs. These classes and their relationships are gathered in a pattern recommended from [8, 22]. They are *role-transaction* pattern and *transaction-specificItem* pattern (Fig. 5).

Figure 4. Contact page on the Sedan-Bouillon site

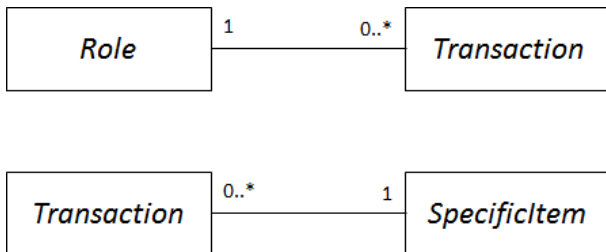


Figure 5. Domain model using collaboration patterns

In a similar way, a task model is specified according to the ConcurTaskTree notation [23]. Fig. 6 reproduces such a task model where different tasks related with the request filling where the user firstly selects a catalog, then provides personal information of contact and finally send his request

In Fig. 6, we can identify patterns that we can find in many task specifications, for instance, when the user must to provide data by using a form (form pattern in Fig. 7). Form pattern structure consists of two elements: an edition and a control part. With the first part user can provide his data, and in the second one user has the control; he can send, cancel or close this task. If user sends the message, he will have a confirmation.

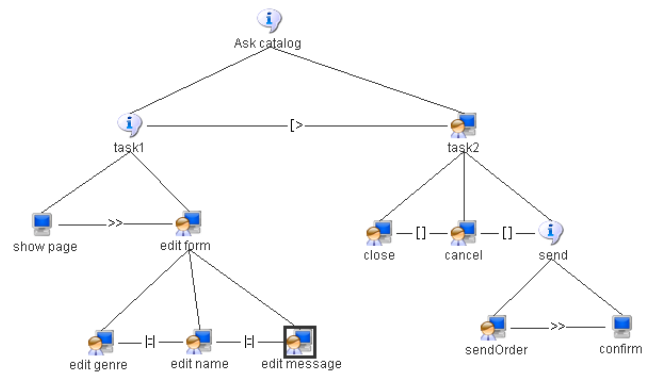


Figure 6. Task model using CTT notation.

After modeling task and domain, an AUI model is needed that represents a canonical expression of the renderings and manipulation of the domain concepts and functions in a way that is independent from any modality and computing platform. Such AIOs are composed of multiple facets, each facet describing a particular function to be assumed (input, output, navigation and control). IDEALXML provides an editor where an abstract representation can be specified using abstract containers, abstract individual components and facets.

Fig. 8 represents a simplified abstract UI: first a container for the request form and then several individual components were defined in order to specify catalogs and components of the form used in this example.

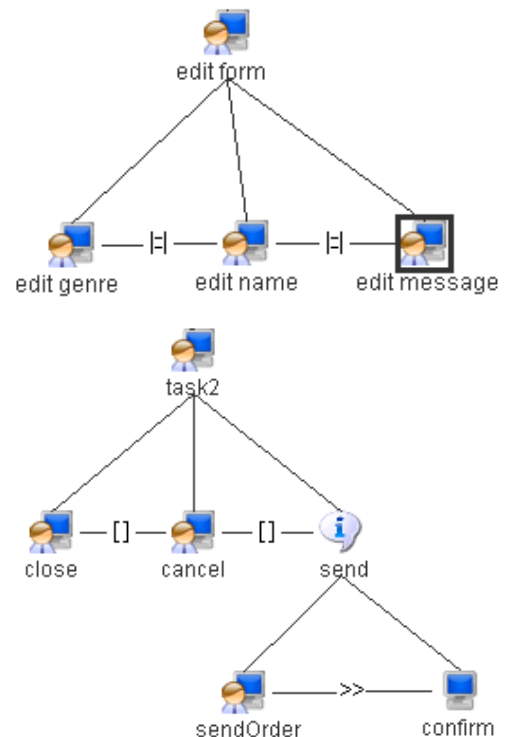


Figure 7. Examples of task patterns: edition form pattern and control form pattern

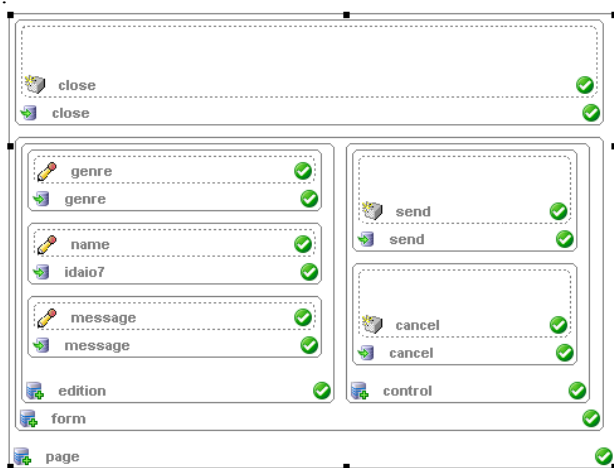


Figure 8. Abstract specification of Sedan-Bouillon form.

All these previous specifications can be done using IDEALXML where four editors (Fig. 9) are provided in order to model tasks, domain presentation and mappings between them. For example, Fig. 9 depicts a mapping between task, domain, and abstract UI, when the designer identifies tasks where the user invokes actions, these actions can include validation of information and then the action will be executed. Methods and attributes will be invoked too when these actions are done.

Finally, representing the task and the domain models, it is possible to link elements of these two models through the mapping model. Such mappings include: *triggers*, *observers*, *updates* (mappings between domain and task), *isReifiedBy*, *isAbstractedInto* (mappings between abstract and concrete UIs), *manipulates* (task and domain) and *isExecutedIn* (task and abstract UI).

In this sense, we can identify patterns between models (intra-model-patterns) as the mapping model (Fig. 9) contains a series of mappings between the related models. Therefore, if we have a domain model that represents a domain pattern and a task model that represents a task pattern, it is possible in IDEALXML to enter mappings between so as to create a task+domain pattern. This reasoning is similar for all subsequent models found in IDEALXML.

CONCLUSION

In this paper, we have introduced IDEALXML, a software that provide facilities for managing UI patterns according to the rules of model-based approach as defined in MDA. With IDEALXML, it is possible to specify task, domain, and UI models in a graphical way and to automatically generate specifications in UsiXML, a XML-based language used to specify UI.

Patterns can be expressed at any level (e.g., one model only) or declined at several levels (e.g., multiple models simultaneously). In addition, it is possible to link several different UI for a single task+domain depending on the context of use. In this case, the context determines the solution given in the UI pattern. The original aspect is that the patterns are generative (the UsiXML specifications initiate automated code generation) as opposed to only descriptive and contemplative.

ACKNOWLEDGEMENTS

The authors also acknowledge the support of the Région Wallonne for ITEA2 Call 3 UsiXML project, under reference 2008026.

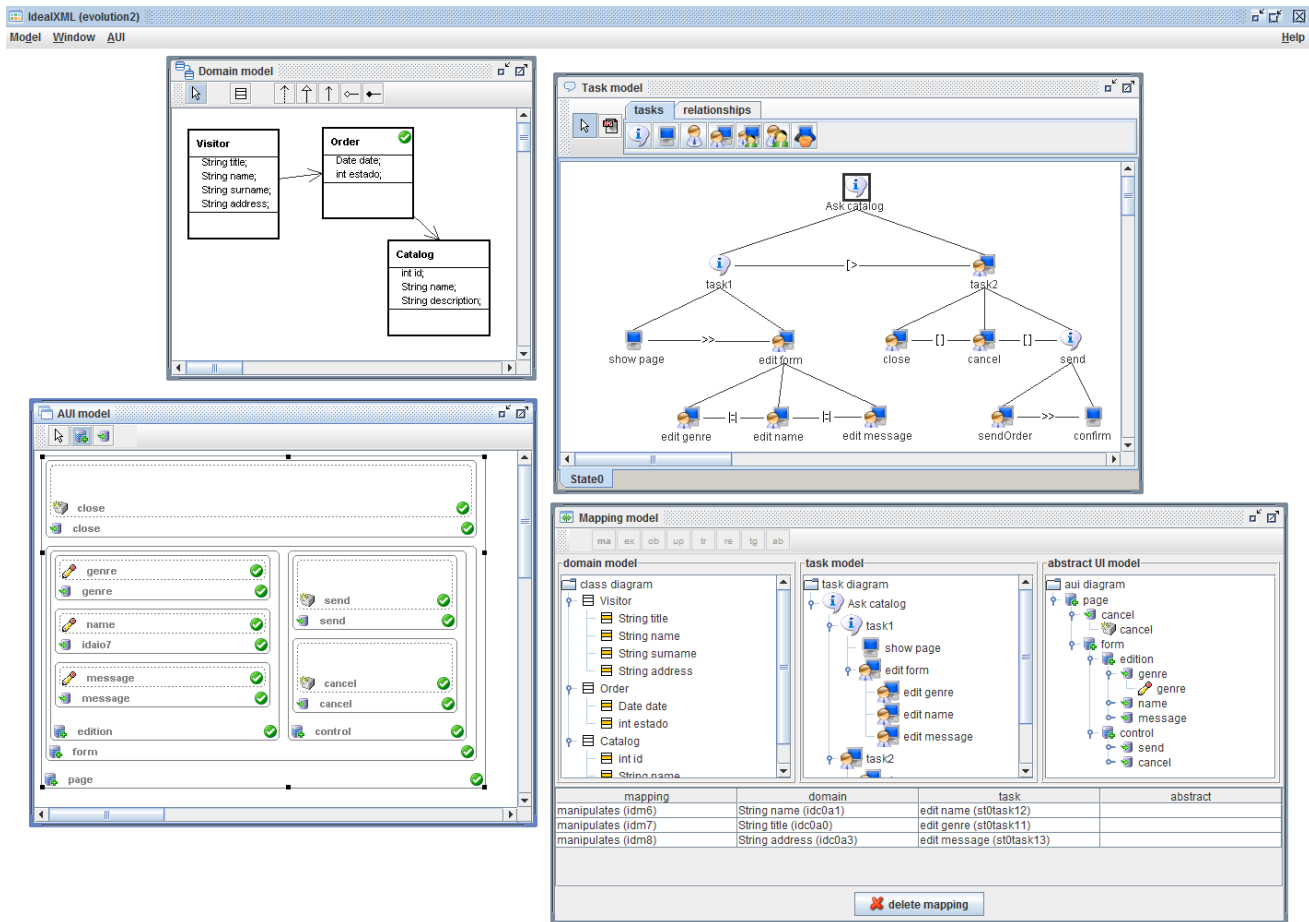


Figure 9. Several screens and tabs provided in IDEALXML for the various models in the UI pattern.

REFERENCES

- Alexander, C., The Timeless Way of Building, Oxford University Press, New York, 1979.
- Bastien, J.M.Ch. and Scapin, D.L. Evaluating a User Interface with Ergonomic Criteria. *Int. J. of Human-Computer Interaction*, 7(2), 1995, pp. 105–121.
- Bayle, E., Bellamy, R. Casaday, G., Erickson, T., Fincher, S., and Grinter, B., “Putting it all Together: Towards a Pattern Language for Interaction Design”, *SIGCHI Bulletin*, 30(1), 1998, pp. 17–24.
- Borchers, J., A Pattern Approach to Interaction Design, John Wiley & Sons, New York, 2001.
- Budinsky, F., Finnie, M., Vlissides, J., and Yu, P., Automatic Code Generation from Design Patterns, *IBM Systems Journal*, 35(2), 1996.
- Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J., A Unifying Reference Framework for Multi-Target User Interfaces, *Interacting with Computers* 15, 3 (2003), pp. 289–308.
- Campos, P. and Nunes, N.J. Towards useful and usable interaction design tools: CanonSketch. *Interacting with Computers* 19 (2007), pp. 597–613.
- Coad, P., Mayfield, M., and North, D., Object Models: Strategies, Patterns, and Applications, Prentice Hall, New Jersey, 1997.
- Constantine, L., “Canonical Abstract Prototypes for Abstract Visual and Interaction Design”, *Proc. of DSVIS’ 2003*, Springer, Berlin, 2003, pp. 1–15.
- Erickson, T., The Interaction Design Patterns Page, http://www.pliant.org/personal/Tom_Erickson/InteractionPatterns.html
- Fincher, S., The pattern gallery, http://www.pliant.org/personal/Tom_Erickson/InteractionPatterns.html
- Fincher, S., Finlay, J., Greene, Sh., Jones, L., Matchen, P., Thomas, J., and Molina, P.J., “Perspectives on HCI patterns: concepts and tools”, *Ext. Proc. of CHI’2003*, ACM Press, New York, 2003, pp. 1044–1045.
- Gaffar, A., Seffah, A., and Van der Poll, J., “HCI Patterns Semantics in XML: A Pragmatic Approach”, *Proc. of HSSE’2005*, ACM Press, New York, 2005.

14. Gamma, E., Helm, R., Johnson, R. and Vlissides, J., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Reading, 1994.
15. Granlund, Å., Lafrenière, D., and Carr, D.A., "A Pattern-Supported Approach to the User Interface Design Process", *Proc. of HCI Int'2001*.
16. Greene, S. and Matchen, P., "Tool-based decision support for pattern assisted development", *Proc. of CHI'2003 Workshop on User Interface Patterns*.
17. Javahery, H. and Seffah, A., "A Model for Usability Pattern-Oriented Design", *Proc. of Tamodia'2002*, Infosec Printing House, Bucharest, 2002, pp. 104–110.
18. Joshi, S.R., McWilliam, W.W., "Case-based reasoning approach to creating user interface components", *Proc. of CHI'96*, ACM Press, New York, 1996, pp. 81–82.
19. Kolodner, J., *Case-Based Reasoning*, Morgan Kaufman, San Mateo, 1993.
20. López-Jaquero, V., Montero, F., Molina, J.P., Fernández-Caballero, A., and González, P., "Model-Based Design of Adaptive User Interfaces through Connectors", *Proc. of DSV-IS'2003*, Springer, pp. 245–257.
21. Molina, P., Belenguer, J., and Pastor, O., "Describing Just-UI Concepts Using a Task Notation", *Proc. Of DSV-IS'2003*, Springer, Berlin, 2003.
22. Nicola, J., Mayfield, M., and Abney, M., *Streamlined Object Modeling*, Prentice Hall, New Jersey, 2001.
23. Paterno, F., *Model-based design and evaluation of interactive applications*, Springer, Berlin, 1999.
24. Pribeanu, C., Vanderdonckt, J. A Pattern-based Approach to User Interface Development. In *Proc. of 2nd Int. Conf. on Universal Access in Human-Computer Interaction UAHCI'2003* (Crete, 22-27 June 2003), Vol. 4, C. Stephanidis (ed.), Lawrence Erlbaum Associates, Mahwah, 2003, pp. 1524–1528.
25. Radeke, F., Forbrig, P., Seffah, A., Sinnig, D. PIM Tool: Support for Pattern-Driven and Model-Based UI Development, *Proc. of Tamodia'2006*, Springer.
26. Schneider, K. and Paquette, D., "Interaction Templates for Constructing User Interfaces from Task Models", *Proc. of CADUI'2004*, Kluwer, pp. 221–232.
27. Seffah, A. and Forbrig, P., "Multiple User Interfaces: Towards a Task-Driven and Patterns-Oriented Design Model", *Proc. of DSV-IS'2002*, Vol. 2545, Springer, Berlin, 2002, pp. 118–132.
28. Seffah, A. and Gaffar, A., Model-based user interface engineering with design patterns, *Journal of Systems and Software*, 80, 2007, pp. 1408–1422.
29. Seffah, A., MOUDIL: Montreal Online Usability Digital Library. <http://hci.cs.concordia.ca/moudil/>
30. Sinnig, D., Gaffar, A., Reichart, D., Forbrig, P., and Seffah, A., "Patterns in Model-Based Engineering", *Proc. of CADUI'2004*, Kluwer, pp. 195–208.
31. Vanderdonckt, J., Furtado, E., Furtado, V., Limbourg, Q., Silva, W., Rodrigues, D., Taddeo, L., *Multi-model and Multi-level Development of User Interfaces*, Chapter 10, in Seffah, A. & Javahery, H. (Eds.), "Multiple User Interfaces - Cross-Platform Applications and Context-Aware Interfaces", John Wiley & Sons, New York, November 2003, pp. 193–216..
32. Vanderdonckt, J. A MDA-Compliant Environment for Developing User Interfaces of Information Systems. In *Proc. of 17th Conf. on Advanced Information Systems Engineering CAiSE'05 (Porto, 13-17 June 2005)*, O. Pastor & J. Falcão e Cunha (eds.), Lecture Notes in Computer Science, Vol. 3520, Springer-Verlag, Berlin, 2005, pp. 16–31.
33. van Welie, M., *Patterns in Interaction Design*, 2004.
34. van Welie, M., van der Veer, G.C., and Eliens, A., "Patterns as Tools for User Interface Design", *Proc. Of TFWWG'2000*, Springer, London, 2000, pp. 313–324.

User Interface Composition with UsiXML

Sophie Lepreux^{1,2,3}, Jean Vanderdonckt⁴, Christophe Kolski^{1,2,3}

¹Univ Lille Nord de France, F-59000 Lille,

²UVHC, LAMIH, F-59313 Valenciennes,

³CNRS, FRE 3304, F-59313 Valenciennes,

France - +33 327 511 465

{sophie.lepreux, christophe.kolski}

@univ-valenciennes.fr

⁴Université catholique de Louvain

Louvain School of Management,

Place des Doyens, 1

B-1348 Louvain-la-Neuve (Belgium)

+ 32(0)10/47.85.25

jean.vanderdonckt@uclouvain.be

ABSTRACT

This paper presents novel ongoing works on user interfaces composition. These works have emerged with the problematic of component software composition transposed to the Human-Computer Interaction domain. Some software architectures indeed allow components assembling at the final design step. Our work, based on UsiXML, aims at proposing a composition/decomposition of user interfaces. These works begin with the concrete level of UsiXML dedicated to the graphical modality and continue with higher abstraction levels. This article provides a positioning of the proposal related to composition compared to the seven dimensions related to the "µ7" concept of UsiXML project.

Author Keywords

Composition, decomposition, interactive component, user interface adaptation, user interface extensible markup language, "µ7" concept.

General Terms

Design, Human Factors, Languages.

ACM Classification Keywords

D.2.2 [Software Engineering]: Design Tools and Techniques – *User interfaces*. H.5.2 [Information Interfaces and Presentation]: User Interfaces – *Graphical user interfaces*.

INTRODUCTION

The issue of software component reuse has given a new issue in Human-Computer Interaction (HCI) on the *User Interface (UI) Composition* [2,6,10,11,18]. If we take as hypothesis that a software component can be associated with interactive(s) component, then the composition of user interface components emerges. Many different terms have been used to refer to this problem, such as fusion [2], transparent composition [10], composable UIs [22], component-based adaptation [11], or pattern-based merging [7]. It is also interesting to note that UI compositions has been addressed at different levels of abstraction, ranging from code level [22], components [6,11], concrete UI [20], abstract UI [23], and task [1,3]. Different UI types have also been considered such as: web UI [1], cross-platform applications [2], web services [5,8,9], context-aware systems [13], tangible UIs [16,17], decision-support systems [18], and multimodal UIs [19]. Many different techniques have been used [7,8,32,33].

This article aims at presenting works ongoing on UI composition, in particular with respect to UsiXML [27]. The UsiXML language is XML-compliant language so the UI elements can be structured according to tree structure. The first section of introduction presents the tree algebra which is used to handle the model. The second section aims at giving the major principles of UsiXML which are useful to understand the paper. Then, the first part introduces the composition operators and algorithms which had proposed to compose user interfaces. These operators were initially proposed to be used on graphical user interfaces [20], modeled with UsiXML. Nevertheless, we can see in second part they can be used relatively to higher abstraction levels, i.e. abstract user interface and task level. Algorithms proposed specifically to the concrete graphical user interface are generalizable to the set of models proposed by UsiXML [27], but could be also applied to equivalent User Interface Description Languages (UIDLs) [12]. The last section shows how composition operators support UI adaptation according to the 7 dimensions introduced in the UsiXML.

Tree-algebra based UI Composition

Since the UI is represented in UsiXML terms and since it is a XML-compliant language, operations could be defined thanks to tree algebra. The composition operations could be logically defined on the XML tree and directly performed. Jagadish *et al.* define a data model [15]. A data tree is a rooted, ordered tree, such that each node carries data (its label) in the form of a set of attribute-value pairs. Each node has a special, single valued attribute called tag whose value indicates the type of element. A node may have a content attribute representing its atomic value. Each node has a virtual attribute called pedigree drawn from an ordered domain. The pedigree carries the history of "where it came from". Pedigree plays a central role in grouping, sorting and duplicate elimination. They define a pattern tree as a pair $P = (T, F)$, where $T = (V, E)$ is a node-labelled and edge-labelled tree such that:

- Each node in V has a distinct integer as its label ($\$i$).
- Each edge is either labelled pc (for parent-child) or ad (for ancestor-descendant).
- F is a formula, i.e. a Boolean combination of predicates applicable to nodes.

This pattern is used to define a database and to define the predicate used in the operations. As the initial notation is specific to the database, we had proposed a variant which is adapted to documents specific to interface. Indeed, in the HCI case, the most important is the structure and not the content. For example, it is more important to know that the window has a box as subitem than that the window has a height equal to 300. So the attributes are stored with the tag. A node is a tag with these attributes and their content. The pattern tree keeps coherent with the variant definition. Another point specific to the database is that the data are in several data trees so the operators use a collection of data trees in input and output. In the HCI case, the input is one (for the unary operators) or two (for the binary operators) XML documents so one or two data trees.

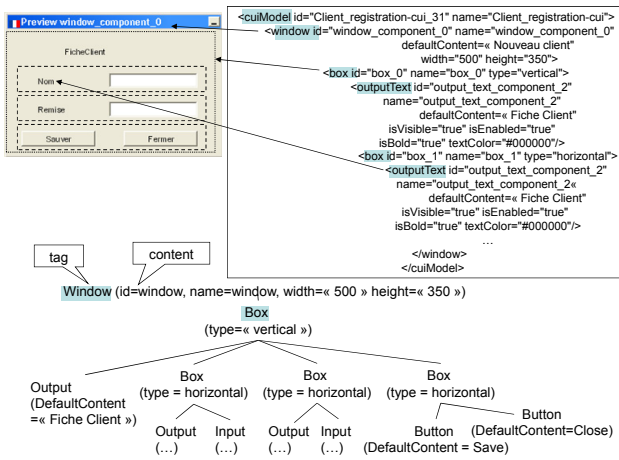


Figure 1. Corresponding between GUI-UsiXML/CUI - tree.

UsiXML framework

UsiXML is structured according to the four abstraction levels of the 'CAMELEON reference framework' [4] for multi-target UIs (Fig. 2). A *Final User Interface* (FUI) refers to an actual UI rendered either by interpretation (e.g., HTML) or by code compilation (e.g., Java). A *Concrete User Interface* (CUI) abstracts a FUI into a description independent of any programming or markup language in terms of Concrete Interaction Objects, layout, navigation [28], and behavior [29,30]. An *Abstract User Interface* (AUI) abstracts a CUI into a definition that is independent of any interaction modality (such as graphical, vocal or tactile). An AUI is populated by *abstract components* and *abstract containers*. Abstract components are composed of facets describing the type of interactive tasks they are able to support (i.e., input, output, control, navigation). The *Tasks & Concepts* level describes the interactive system specifications in terms of the user tasks to be carried out and the domain objects.

COMPOSITION/DECOMPOSITION OPERATORS

Several operators have been proposed [20]. They are distributed in two parts, the unitary operators which act on a single interface and the binary operators which take as ar-

guments two interfaces. A global view of these operators is presented in Fig. 3. The operators are presented in this part as well as several algorithms. They are illustrated with examples in the following part and according "μ7" concept of UsiXML project in the last part.

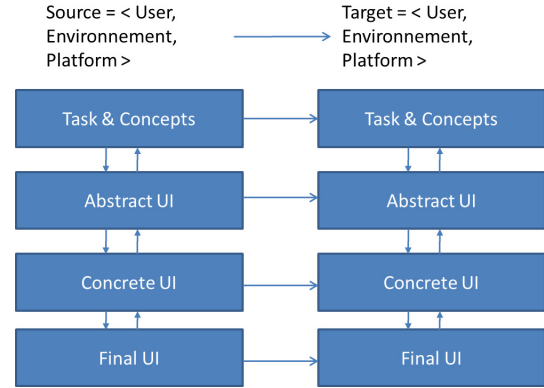


Figure 2. The four abstraction levels used in the Cameleon Reference Framework (CRF) [4].

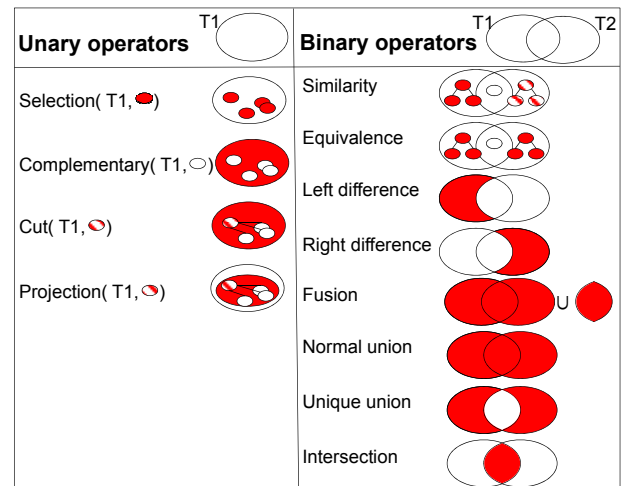


Figure 3. The set of composition operators.

Unary operators

In this part, all the operators are unary. The inputs are a tree T1 and an item or a set of items defined by a pattern tree. The output is a tree.

Selection operator ($\sigma(T1, P)$)

The selection operation applies on a tree and on a set of items defined by a pattern tree. The output tree contains the set of items corresponding to the pattern tree in input which exist in the tree T1. The items are individual components, as leaf nodes. The output tree has the same structure (layout) as the input tree T1. The algorithm is provided in [20].

Complementary operator

The Complementary operation applies on a tree and on a set of items defined by a pattern tree. The output tree is the input tree minus the set of items corresponding to the pattern tree. The algorithm for CUI model is presented in Figure 4.

Cut operator

The Cut operation applies on a tree and on an item. The output tree is the input tree minus the item (I). If the item (I) is not a leaf, all the items of the sub tree (I) are deleted.

Projection operator $((T1, P))$

The Projection operation applies on a tree and on an item. The output is the sub-tree defined from the item. The objective of this operator is not to provide an interface but rather to select a logical set of items. Thus, this operator can be used to capitalize a pattern from an existing interface.

```
Complementary (courant : tree, expression : item)
// Depth First Search
If (courant != NULL) {
    If (courant.attribut != NULL)
        {If (courant.attribut = expression)
            // delete item
            If (courant.child != null)
                { //a child became parent
                    courant.parent.child <- courant.child
                    neighbor <- courant.child.neighbor
                    While (neighbor.neighbor != NULL)
                        neighbor <- neighbor.neighbor
                    neighbor.neighbor <- courant.neighbor
                    Courant.neighbor.parent <- neighbor
                    Complementary (courant.child)
                }
            Else if (courant.neighbor != NULL)
                { courant.parent.child <- courant.neighbor
                  Courant.neighbor.parent <- courant.parent
                  Complementary (courant.neighbor);
                }
        }
    Else { //attribut != expression
        //i.e. expression is the complementary so it is not deleted
        Complementary (courant.child)
        // Depth First Search
        Complementary (courant.neighbor)
        // Breadth Search
    }
}
Else { //no attribut so it is not treated
    Complementary (courant.child)
    // Depth First Search
    Complementary (courant.neighbor)
    // Breadth Search
}
}
```

Figure 4. Complementary algorithm proposed to tree algebra for UI.

Binary Operators

In this part, all the operators are binary. They apply on a pair of trees T1 and T2. The Similarity and Equivalence operators provide a Boolean whereas the other operators provide a tree T3.

Similarity operator $(T1 \sim T2)$

The Similarity operation allows checking whether the two input trees are similar. A distance criterion is computed using the Levenshtein distance from the contents of the two

trees (not the structure) [20]. This distance informs about the similarity between two strings through an evaluation of the minimal number of char operations (deletion, insertion or replacement) to transform a string into another one. In this evaluation - as in the detection of common parts - we suppose the semantic alignment of the user interfaces.

Equivalence operator $(T1 \approx T2)$

The Equivalence operation allows knowing whether two trees are equivalent. Two interfaces are said to be equivalent if they are 100% similar.

Left (resp. right) difference operator $(T1 / (T1 \cap T2))$

The output tree of the Left (resp. right) operation is composed of the content of T1 (resp. T2) minus the redundant items between T1 and T2.

Fusion operator $(T1 + T2 = (T1 \cup T2) \cup (T1 \cap T2))$

The output tree of the Fusion operation is composed of the contents of both T1 and T2; the redundant items are placed twice. At the CUI level, particularly with the graphical modality, the order of the input argument is essential as the operation is not commutative. Furthermore, it is possible to precise whether the fusion (resp. Normal Union and Unique union) has to be horizontal or vertical. Such a parameter allows structuring the T3 output [20].

Normal union operator $(T1 \cup T2)$

The output tree of the Normal Union operation is composed of all contents of T1 and T2 but, contrarily to the Fusion operator, the redundant items are placed only once. An illustration of this operator with a horizontal parameter is shown in Fig. 5 and its tree representation in Fig. 6.

Unique union operator $((T1 \cup T2) / (T1 \cap T2))$

The output tree of the Unique Union operation is composed of contents of T1 and T2. In this union, the redundant items are deleted.

Intersection operator $(T1 \cap T2)$

The output tree of the Intersection operation is composed of the items contained in both T1 and T2; the non-redundant items are deleted. The algorithm proposed to the CUI level is presented in Fig. 7.

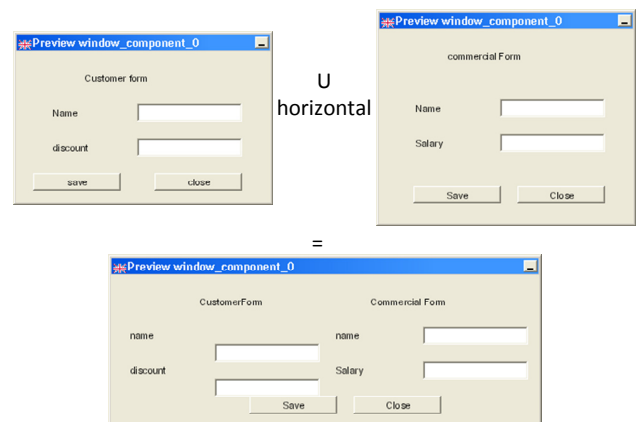


Figure 5. Normal Union operator applied on two UIs.

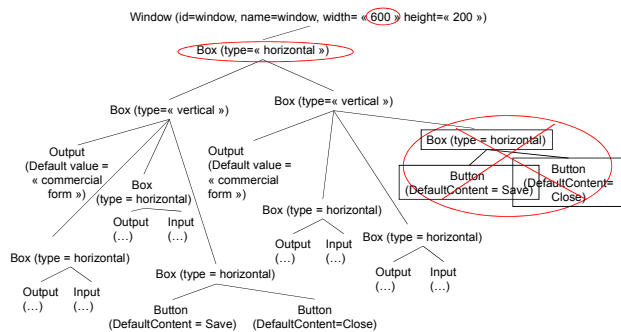


Figure 6. Normal Union operator – representation of result by tree.

```

For each component of T1,
  If {Component.defaultContent ∈ T2}
    // Hypothesis: Semantic alignment
    Add{Component, T3}
  Else {return false}
  If {Component.tag=InputText}
    // Component do not have a defaultContent
    // IsIdentifiedBy function have to be used to know its role
    {X = Component.getIdentifiedBy()}
    if {X ∈ T2} Add {Component, T3}
  Else return False
}
Return T3

```

Figure 7. Intersection on a tree algebra for CUI.

OPERATORS APPLIED ON USIXML LAYERS

This part is composed of three sections corresponding to the CUI layer, the AUI layer and the Task model layer. In each section, an illustration of some operators use is given.

Illustrations of operators use on a concrete UI

The above operators have been directly implemented in ComposiXML [20], a plug-in of GrafiXML, a graphical interface builder that automatically generates UsiXML specifications (Fig. 8) as opposed to final code for other builders. As they are an open source project regulated by Apache 2.0 open licence and available on SourceForge, they can be downloaded from <http://www.usixml.org>. GrafiXML is able to automatically generate code of a UI specified in UsiXML into (X)HTML or Java. For the purpose of the examples below, we will rely on the Java automated code generation.



Figure 8. User Interface of ComposiXML.

This part presents some illustrations of operators used at design time with ComposiXML. For example, they can be used by designers to create UIs which are developed for one or several applications within the same company, with the constraint of a corporate style guide. The designers can reuse some of the elements of the user interfaces thanks to

operators. The second use case is at run time. It is integrated in the reuse issue which has introduced the component idea. The first issue in this domain is the composition of the components. If we consider the business component as a component with UIs, one issue in the domain of HCI is to compose the user interfaces of several business components. If we consider that the user interfaces are specified with UsiXML, the Union operator is particularly interesting for such composition of business components. Illustrations of operators are given for example in the domain of tourism. In this domain, it frequently happens that some parts of the same information should be reproduced in different UI for different events (e.g., hotel information, tourist trip including hotel booking, booking a hotel, etc.).

Fig. 9 (a) reproduces a screenshot of the preview in Java (obtained by Java automated code generation) of a Concrete User Interface created with the GrafiXML editor. Fig. 9 (b) reproduces another UI for an event management application. The two UIs only differ from a few fields, here the event dates (Fig. 9-T1) and the comment (Fig. 9-T2).

The Selection is used on the tourist application UI with a set of elements as input to give a UI subset (Fig. 9(a)).

Likewise, the Projection operator is used to extract a set of items according to their type (i.e., structure). Given as an example, the Fig. 9 (b) shows the result of the Projection operator on the tourist application with two parameters: outputText and button. The preview is in French.

Therefore, if we want to identify the common part of these two UIs, the Intersection operator performs the operation as defined previously to identify common parts of both trees and then rebuilds a new tree with the identified elements. This operator re-generates new UsiXML specifications. This intersection is reproduced in Fig. 9(c).

Note in Fig. 9 (c) that the designer did not have anything to do: all common elements were identified, a new layout was produced so as to mimic the initial one and all objects have been laid out and aligned to preserve the initial constraints.

Illustrations of operators use on abstract user interface

The AUI level allows specifying user interfaces without to know neither the modality nor the platform. It can be developed with a tool provided by UsiXML project named IdealXML. A representation of the task “order a pizza” is given in Fig. 10 (a) and its corresponding UsiXML in Fig. 10 (b). We notice that *abstractIndividualComponent* as “Choose quantity” is placed in an *abstractContainer* corresponding by “Choose a pizza”. The structure of user interface composed by container, component and relation between elements is here again respected by this model. Therefore, the operators, as at the CUI level, can be applied using tree algebra. The tree corresponding to the specification view of Fig. 10 (a,b) is in Fig. 10 (c) [18]. An example of operators using at this level is shown in the following part, in the section dealing with the multi-modality [19].

Illustrations of operators used on task tree

The operators applying on task level are similar to other levels because the UsiXML language provides a XML structure and the structure of UI follows also the set theory. However, the operators have to be adapted to take into account the relationship between tasks in the cases of fusion and Unions. Given as an example, Fig. 10 provides an illustration of the Union operator use on task trees.

Conclusion on the (de)composition operators

In this part, illustrations of operators used were given at each abstraction level according to the adaptation needs. Some works are not again completely and must be extended in the future but the illustrations give an idea of the interests to use operators to adapt and the possibility of using them at each abstraction level. The illustrations are given to describe the use of one or several operators on each level. In the following part, the adaptation needs and the use of associated operators are illustrated according to the 7 dimensions referred by UsiXML.

COMPOSE TO SUPPORT THE 7 DIMENSIONS

This part aims at illustrating the use of (de)composition operators to adapt the user interfaces according to the 7 dimensions: Multi-device Usage, Multi-User interface, Multilinguality, Multi-organization, Multi-context usage, Multi-Modality usage, Multi-platform Usage.

Multi-device Usage

The simultaneous use of several devices sharing a same user interface implies the need to create FUI for each device.

They are noted FUI'1 to n in Figure 11. The devices can possibly be distributed. In order to transform the initial user interface (i.e. source) into several UI parts, duplication or a new composition of UI parts are necessary. In order to decompose UI or to extract several components to be reused on other devices, the unary operators such as Selection or Projection are well suited. These operators can be applied at the CUI level (Figure 11). The decomposition may be use the task model in order to choose the part to extract/duplicate etc. At the end of this first step, le CUI' 1 to n are defined following each device. If the interface of one device must integrate elements provided by several services, then the Fusion or Union operators will be appropriated. For instance, Figure 11 illustrates the examples of a user who may control a music player running on a media center using a remote control on a handheld device, perhaps coupled with a Wii for volume control. This example is composed of three tasks: the Music controller task, the Sound player task and the volume controller task and Three devices: the media center which is used to realize the Music controller task and the Sound player task. The handheld device to make a remote control (sound controller and Music controller) and the Wii for the volume control. The source CUI is an existing UI dedicated to the Music Player task. Elements are extracted are distributed to the several devices described. We have used the CUI level to use operators because it is sufficient to this dimension but they could be also applied at the AUI level or at the task level if the UI repartition corresponds to the task(s).

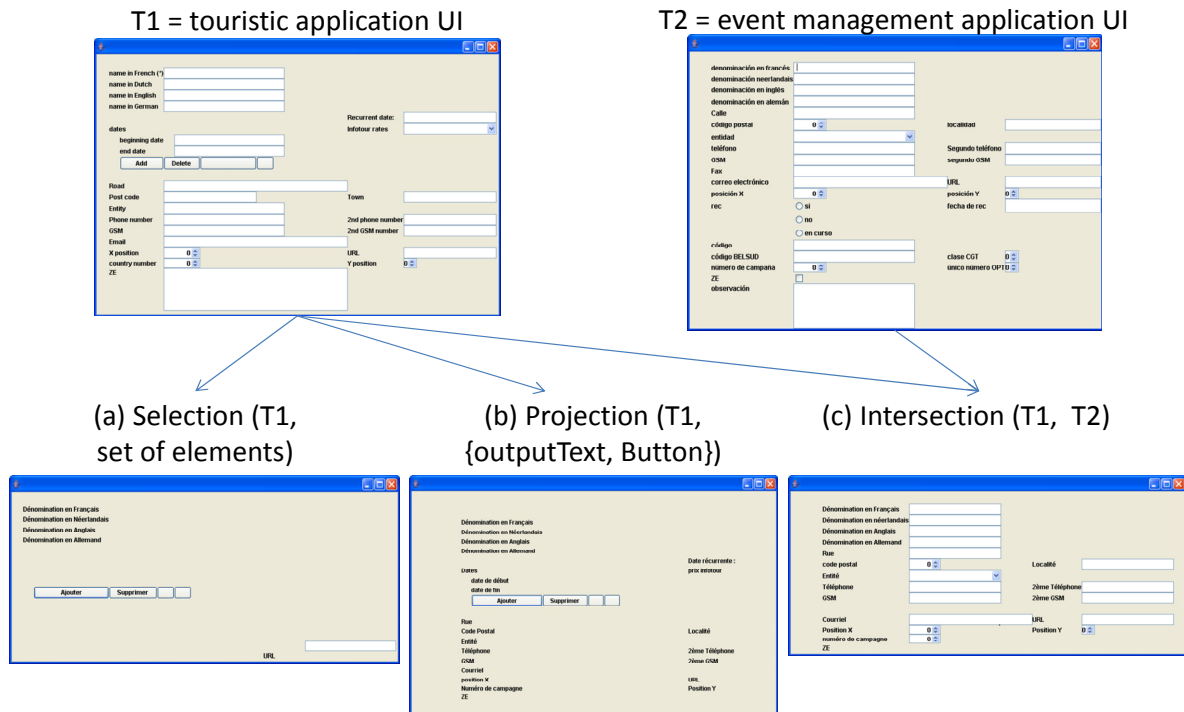
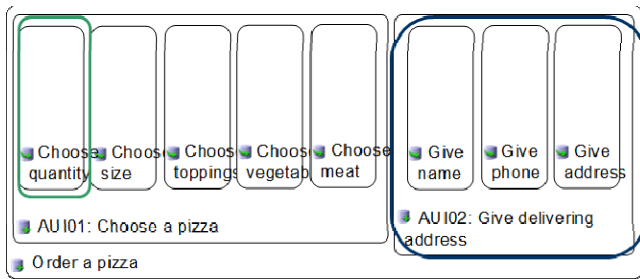


Figure 9. Illustrations of operators on CUI models.



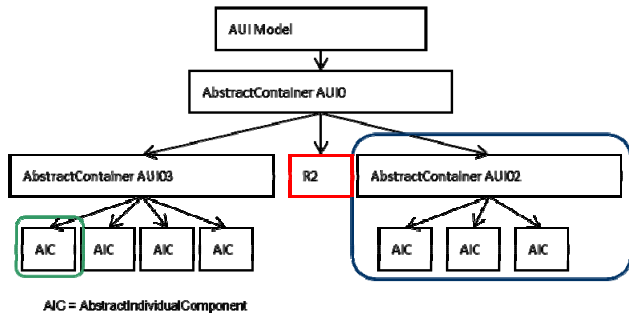
(a) Graphical representation developed with IdealXML

```

<auiamodel>
<abstractContainer id="idaio00" name="Order a pizza">
<abstractContainer id="idaio01" name="idaio01">
  <abstractIndividualComponent id="idaio02"
    name="Choose quantity">
  </abstractIndividualComponent>
  <abstractIndividualComponent id="idaio03"
    name="Choose size">
  </abstractIndividualComponent>
  ...
  <abstractIndividualComponent id="idaio06"
    name="[Choose meat]">
  </abstractIndividualComponent>
</abstractContainer>
<abstractContainer id="idaio07" name="idaio07">
  <abstractIndividualComponent id="idaio08"
    name="Give name">
  </abstractIndividualComponent>
  ...
</abstractContainer>
</auiamodel>

```

(b) UsiXML specification given by IdealXML



(c) Representation of the AUI model by tree

Figure 10. Three representations of the AUI model, developed with IdealXML, concerning a pizza order.

The tasks trees in the illustrations are created with the CTTE tool (<http://giove.isti.cnr.it/tools/ctte/>) [26]. Incidentally, they proposed to use CTT in order to adapt the UI following the device in [26]. Note that [7] propose composition-oriented user interface design patterns (CUIDP) with the same goal to reuse the UI knowledge. This work is also based on model-driven architecture and is based on UMLi to specify the UI.

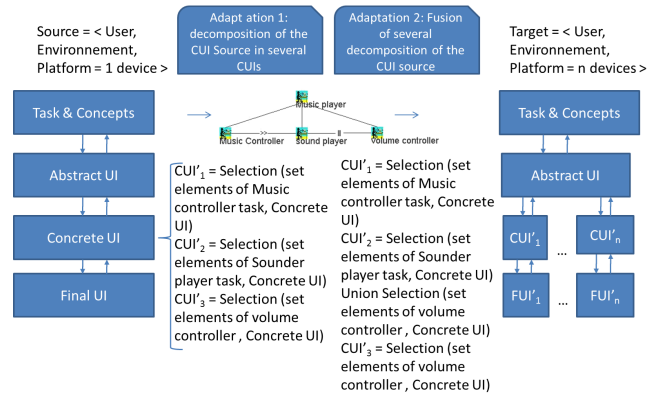


Figure 11. Definition of operators for Multi-device Usage.

Multi-User interface

Following this dimension, two adaptation propositions are possible. If the shared interface is co-localized, the users can simultaneously interact on one interface (for example a tactile multi-touch interface or tangible objects used [16]). In this case, it seems not necessary to adapt the interface by composition even if it should be adapted to the user (profiles, preferences, etc.). But, the problem is different if a user arrives in the collective work situation (or leaves it): the user needs a work space, with personal data and/or functions, and in a case of an interactive table for instance, the multi-user interface has to be adapted again, using composition/decomposition operators [17]. If the shared interface is distant, each user has a duplicate of this interface. In this case, duplication has to be carried out. The duplication operator is not proposed in this article (it can be viewed in [20] with application on a complete UI tree) but the Projection operator with the root node item as input allows to duplicate the whole arborescence. If a part only of UI must be duplicated, the input item of the Projection operator will be the item of the output root node. In parallel, the problem of the user rights or roles has also to be carefully considered. For instance in a brainwriting [14] session at a distance, the UI of the brainwriting moderator will be composed differently as the UI of the other participants.

Multi-linguality

Language is defined here as spoken language (not programming language). Several languages should not be used simultaneously within a same interface. Thus in many cases it seems not necessary to compose several parts of user interfaces. Nevertheless, if two similar interfaces are modeled at the CUI level (or higher) with specifications given for one (or more) language and another interface modeled with others languages, it can be possible to compose these interfaces to obtain one interface modeled for all these languages. Then, the UsiXML structure allows generating the final user interface in the selected language. But is it also important to consider translation considerations leading to necessities about composition/decomposition due to socio-cultural aspects (i.e. problems of designing UI for international use [20]): from a UI using a source language which

is read from left to right (and downwards; for instance, English), the problem is not trivial if the target UI has to use a language which is read from right to left (and downwards; for instance, Arabic), or upwards (for instance, Chinese). Indeed in several cases the different zones of the UI have to be composed differently.

Multi-organization

According to this dimension, the organization and task notions are essential. A task may be common to several organizations. The shared task must be associated to one interface. The Intersection operator allows detecting the common tasks whereas the Normal Union operator allows merging the two tasks without repetition of common parts (Fig. 12). These operators may be used at the task level of the UsiXML architecture [22]. We can note that Bihler *et al.* [2] implemented cross-application dynamic UI fusion in order to realize a task on a platform with a restricted graphical space. It is also important to recall that several organization models exist and are described in the literature (for instance in social sciences or multi-agent system domain) and lead to very different work methods. If several types of organizations are concerned, the composition/decomposition process may have to follow adapted rules or principles. An important research work has to be done on this subject.

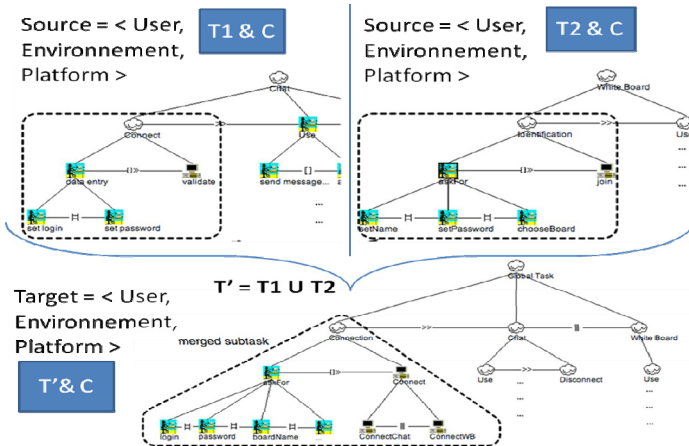


Figure 12. Normal Union operator for Multi-organization.

Multi-context usage

This dimension is integrated to UsiXML architecture since the structure with 4 levels and the transformation following the context change allow adapting user interfaces, whatever the abstraction level (Figure). A development methodology based on patterns, business component and learning, proposed in [13], is complementary to this architecture to generate and adapt the user interfaces according to the context evolution [13]. Another work focuses on the integration of the task model in the business component in order to compose complete application and to facilitate the co-evolution of the system [3]. A context modification can

generate evolutions at each abstraction level. For example, a lighting change acts on the luminosity of the device (FUI) whereas the background noise change acts (or not) on the choice of the vocal modality. A modification of the user work brings modification of tasks and/or on the tasks planning.

Multi-Modality usage

The AUI layer allows specifying the UI independently of the modality (i.e., vocal, graphical, multimodal). According to needs brought by context change, a UI composition at this level (or at a higher level) allows merging interfaces which will be concretized by different modality in the lower level. For example, a food order can be realized with several modalities. Some tasks can be vocal in the case of phoning order or graphical in the case of an Internet connected device (ubiquitous or not) [19]. In this example, the Normal Union operator is used merging two applications existing partly. On one hand, in the application « order a pizza » which is illustrated above Fig. 10, the sub-task « Choose a pizza » is multimodal (see CUI model and FUI (XHTML+VoiceXML) in Fig. 13). On the other hand, the « Chinese food order » application is only graphically developed. The normal Union of these two applications gives a result which can be multimodal. The Normal Union avoids to repeat the sub-task « delivering address » which is common to the two applications (Fig. 14).

```
<outputText id="Ask for pizza quantity"
  name="Ask for pizza quantity" defaultContent="Quantity:"/>
<inputText id="pizzaQuantity" name="quantity"
  defaultContent="1"
  voice_prompt="How many pizzas would you like?"
  voice_event_help="Say a number between one and twenty."
  voice_event_nomatch="Sorry I did not understand you."
  voice_event_noinput="You have to pronounce a quantity of
  pizza."/>
<voice_quantity:grammar>
  <![CDATA[
    #USGF V1.0;
    grammar pizza_quantity;
    public <quantity> = 1 | 2 | 3 | 4 | 5 | 6 | 7 |
    8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20
    ;
  ]]>
</voice_quantity:grammar>
</inputText>
<outputText id="Ask for pizza size" name="Ask for pizza size"
  defaultContent="Size:"/>
<group voice_prompt="What size would you like?">
  <radioButton id="radiobuttonSize1"
    name="radiobuttonSize" defaultContent="small" />
  <radioButton id="radiobuttonSize2"
    name="radiobuttonSize" defaultContent="medium"/>
  <radioButton id="radiobuttonSize3"
    name="radiobuttonSize" defaultContent="large"/>
  <voice_quantity:grammar>
    <![CDATA[
      #USGF V1.0;
      grammar pizza_size;
      public <size> = small | medium | large ;
    ]]>
  </voice_quantity:grammar>
</group>
```

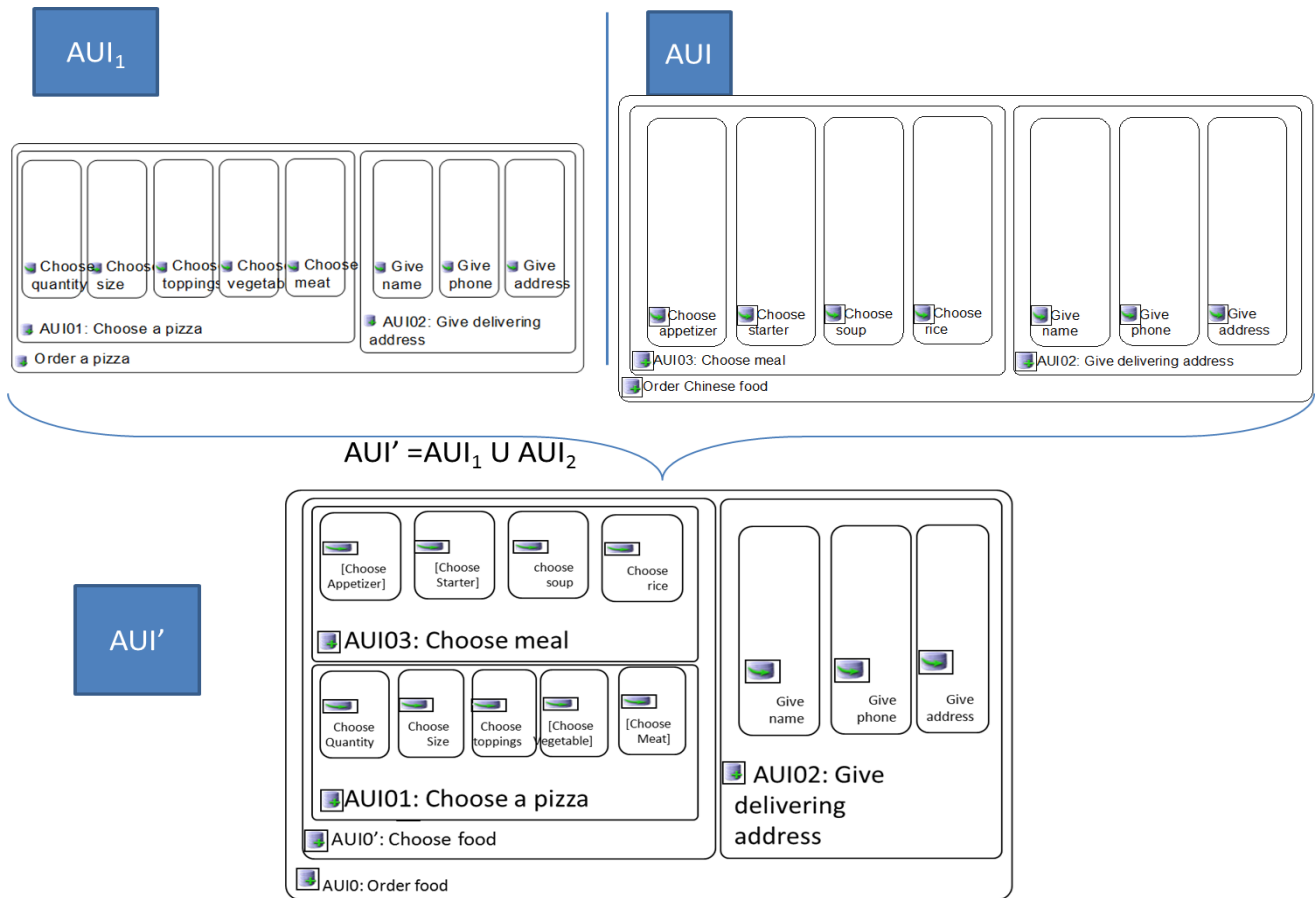



Figure 14. Normal Union (AUI (order a pizza), AUI (Order Chinese food))

The figure shows a multimodal user interface for a pizza ordering task. It consists of a 'Concrete UI' (top) and a 'Final UI' (bottom). The 'Final UI' is a form with the following elements: a 'Quantity' input field, a 'Size' selection with radio buttons for 'Small 12"', 'Medium 16"', and 'Large 22"', a 'Toppings' section with a checkbox for 'Extra Cheese', a 'Vegetable Toppings' section with checkboxes for 'Olives', 'Mushrooms', 'Onions', and 'Peppers', and a 'Meat Toppings' section with checkboxes for 'Bacon', 'Chicken', 'Ham', 'Meatball', 'Sausage', and 'Pepperoni'. A 'Submit Pizza Order' button is located at the bottom of the form.

Figure 13. A Multimodal CUI and FUI corresponding to the “Choose a pizza” sub-task.

Multi-platform Usage

The *multi-device usage* dimension often integrates the multi-platform dimension. So this part focuses on the adaptation of user interfaces centered on the platform only (not on the device). An example of multi-platform language is the java language. It can be executed on several platforms without changing model or implements. This dimension aims at doing the same from the user interfaces point of view. The UsiXML project allows for example to generate code from CUI model to FUI model. For example, GraphiXML (a graphical editor) allows generating java or XHTML code. In this case, we can say that the graphical user interface which is modeled with GraphiXML is multi-platform. This principle has to be generalized. According to this analysis, the composition operators are not needed into this process.

Conclusion on the support of the 7 dimensions of UsiXML

In the part here above, the adaptation of user interfaces following each of the dimensions has been explored using (de-)composition operators. Some dimensions do not require the use of (de-)composition operators whereas others have interest to use them. The analysis is realized for each dimension separately even if the 7 dimensions are complementary. Adaptations to criterion (dimension) can be done

to the detriment of the others. As a result, the adaptation must be global i.e. the adaptation has to take into account all the criteria simultaneously and not considered one by one. To go deeper, it is possible to do an analogy with McCall and his colleagues in 1977 [24] in software quality domain, considering that compromises between quality criteria are necessary from an application domain to another (for instance: efficiency is not always compatible with testability and portability). So we think that according to an application domain with its characteristics (real time or not, centralized or distributed, connectivity aspects, screen size, characteristics of the users well known or not (application for the general public), use frequency, ...), compromises have to be made between the seven dimensions for insuring the best quality as possible of the final UI. More, research questions are open concerning (1) the one by one or simultaneous consideration of these dimensions, (2) the order in taking them into consideration.

CONCLUSION

On one hand, the article has presented operators which have been developed initially to compose at design time and at the CUI level. Afterwards, these operators have been used to each abstraction level of user interfaces. Likewise, some examples have shown that the operators could be used at the runtime. These operators are a support to the user interface adaptation and their use was illustrated specifically for each dimension; several research ways have been also suggested. As another perspective to this work, we propose an adaptation engine which takes into account several criteria simultaneously in order to perform this adaptation.

ACKNOWLEDGMENTS

We gratefully acknowledge the support of the ITEA2 Call 3 UsiXML project under reference 20080026 supported by the European Commission and Région Wallonne. This current research work has been partially supported by CISIT, the Nord-Pas-de-Calais Region, the European Community, the Regional Delegation for Research and Technology, the Ministry of Higher Education and Research, and the National Center for Scientific Research. The authors gratefully acknowledge the support of these institutions.

REFERENCES

1. Betermieux, S. and Bomsdorf, B. Task-Driven Composition of Web User Interfaces. In *Proc. of 6th Int. Conf. of Computer-Aided Design of User Interfaces CA-DUI'2006* (Bucharest, June 6-8, 2006). Information Systems Series, Springer-Verlag, Berlin (2007), pp. 233–244.
2. Bihler, P. and Kniesel, G. Seamless Cross-Application Workflow Support by User Interface Fusion. In *Multiple and Ubiquitous Interaction*, S. Boedker, C. Brodersen, C.N. Klokmoose (eds.). DAIMI PB-581, University of Aarhus, 2007.
3. Bourguin, G., Lewandowski, A., and Tarby J.C. Defining Task Oriented Components. In *Proc. of the 6th Int. workshop on Task MOdels and DIagrams TAMODIA'2007* (Toulouse, November 7-9, 2007). Lecture Notes in Computer Science, Vol. 4849. Springer-Verlag, Berlin (2007), pp. 170–183.
4. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J. A Unifying Reference Framework for Multi-Target User Interfaces. *Interacting with Computers* 15(3), 2003, pp. 289–308.
5. Caramel, B., Joffroy, C., and Laguerre, M. De la composition de services à la composition d'interfaces homme-machine. In *Proc. of the 21st Int. Conf. on Association Francophone d'Interaction Homme-Machine IHM'2009* (Grenoble, October 13-16, 2009). ACM Int. Conf. Proc. Series, ACM Press, New York (2009), pp. 65–74.
6. Dery-Pinna, A.-M., Fierstone, J., and Picard, E. Component Model and Programming: a First Step to Manage Human-Computer Interaction Adaptation. In *Proc. of 5th Int. Symposium on Human-Computer Interaction with Mobile Devices and Services MobileHCI'2003* (Udine, September 8-11, 2003). Lecture Notes in Computer Science, Vol. 2795. Springer-Verlag, Berlin (2003), pp. 456–460.
7. Feng, S. and Wan, J. Multi-device User Interface Development with Composition-oriented User Interface Design Patterns. In *Proc. of the 8th ACIS Int. Conf. on Software Engineering, Artificial intelligence, Networking, and Parallel/Distributed Computing SNPDP'2007* (July 30 - August 1, 2007). Volume 3. IEEE Computer Society, Washington (2007), pp. 605–610.
8. Gabillon, Y., Calvary, G., and Fiorino, H. Composing interactive systems by planning. In *Proc. of the 4th French-speaking Conference on Mobility and ubiquity computing UbiMob'2008* (Saint-Malo, May 28-30, 2008). ACM International Conference Proceeding Series, Vol. 277. ACM Press, New York (2008), pp. 37–40.
9. Gabillon, Y., Calvary, G., Mandran, N., and Fiorino, H. Composition dynamique d'interfaces homme-machine: besoin utilisateur ou défi de chercheur? In *Proc. of the 21st Int. Conf. on Association Francophone d'Interaction Homme-Machine IHM'2009* (Grenoble, October 13-16, 2009). ACM Int. Conf. Proc. Series, ACM Press, New York (2009), pp. 61–64.
10. Ginzburg, J., Rossi, G., Urbietta, M., and Distant, D. Transparent Interface Composition in Web Applications. In *Proc. of Int. Conf on Web Engineering ICWE'2007* (Como, July 16-20, 2000). Lecture Notes in Computer Science, Vol. 4607, Springer-Verlag, Berlin (2007), pp. 152–166.
11. Grundy, J.C. and Hosking, J.G. Developing Adaptable User Interfaces for Component-based Systems. *Interacting with Computers* 14(3), 2002, pp. 175–194.

12. Guerrero García, J., González Calleros, J.M., Vanderdonckt, J., and Muñoz Arteaga, J. A Theoretical Survey of User Interface Description Languages: Preliminary Results. In *Proc. of Joint 4th Latin American Conference on Human-Computer Interaction-7th Latin American Web Congress LA-Web/CLIHIC'2009* (Merida, November 9-11, 2009), E. Chavez, E. Furtado, A. Moran (eds.). IEEE Computer Society Press, Los Alamitos (2009), pp. 36–43.
13. Hariri, M-A., Tabary, D., Lepreux, S., and Kolski C. Context aware Business adaptation toward User Interface adaptation. *Communications of SIWN* 3, 2008, pp. 46–52.
14. Heslin, P.A. Better than brainstorming? Potential contextual boundary conditions to brainwriting for idea generation in organizations. *Journal of Occupational and Organizational Psychology* 82(1), March 2009, pp. 129–145.
15. Jagadish, H.V., Lakshmanan, L.V.S., Srivastava, D., and Thompson, K. TAX : A Tree Algebra for XML. In *Proc. of 8th Int. Workshop on Database Programming Language DBPL'2001* (Frascati, September 8-10, 2001), G. Ghelli, G. Grahne (eds.). Lecture Notes in Computer Science, Vol. 2397, Springer-Verlag, Berlin (2001), pp. 149–164.
16. Kubicki, S., Lepreux, S., Lebrun, Y., Dos Santos, P., Kolski, C., and Caelen, J. New Human-Computer Interactions Using Tangible Objects: Application on a Digital Tabletop with RFID Technology. In *Proc. of the 13th Int. Conf. on Human-Computer Interaction HCI'International'2009* (San Diego, 19-24 July 2009), J.A. Jacko (ed.), Part III. Lecture Notes in Computer Science, Vol. 5612. Springer-Verlag, Berlin (2009), pp. 446–455.
17. Kubicki, S., Lepreux, S., Kolski, C., and Caelen, J. Towards New Human-Machine Systems in contexts involving interactive table and tangible objects. In *Proc. of 11th IFAC/IFIP/IFORS/IEA Symposium on Analysis, Design, and Evaluation of Human-Machine Systems* (Valenciennes, August 31 - September 3, 2010).
18. Lepreux, S., Kolski, C., and Abed, M. IHM et SIAD : vers une composition d'outils interactifs pour l'aide à la décision. In *Proc. of 16th Conf. on Association Francophone d'Interaction Homme-Machine IHM'2004* (Namur, August 30-September 3, 2004). ACM Int. Conf. Proc. Series, ACM Press, New York (2004), pp. 227–230.
19. Lepreux, S., Hariri, A., Rouillard, J., Tabary, D., Tarby, J.-C., and Kolski, Ch. Towards Multimodal User Interfaces Composition based on UsiXML and MBD principles. In *Proc. of 12th Int. Conf. on Human-Computer Interaction HCI International'2007* (Beijing, July 22-27, 2007), J.A Jacko (ed.), Part III. Lecture Notes in Computer Science, Vol. 4552. Springer-verlag, Berlin (2007), pp. 134–143.
20. Lepreux, S., Vanderdonckt, J., and Michotte, B. Visual Design of User Interfaces by (De)composition. In *Proc. of 13th Int. Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'2006* (Dublin, July 26-28, 2006), G. Doherty, A. Blandford (eds.). Lecture Notes in Computer Science, Vol. 4323. Springer-Verlag, Berlin (2006), pp. 157–170.
21. Levenshtein, V., Efficient reconstruction of sequences from their subsequences or supersequences. *Journal of Combin. Theory, Ser. A*, 93(2), 2001, pp. 310–332.
22. Leventhal, E., and Grubis, Al. Composable User Interfaces. The MITRE Corporation, Bedford, 2004.
23. Lewandowski, A., Lepreux, S., and Bourguin G. Tasks models merging for high-level component composition. In *Proc. of 12th Int. Conf. on Human-Computer Interaction HCI International'2007* (Beijing, July 22-27, 2007). lecture Notes in Computer Science, Vol. 4550. Springer-Verlag, Berlin (2007), pp. 1129–1138.
24. McCall, J.A., Richards, P.K., and Walters J.F. *Factors in Software Quality*, 3 volumes. RADC-TR-77-369, 1977.
25. Nielsen, J. Designing User Interfaces for International Use. Elsevier Science Publishers, 1990.
26. Paternò, F., Mancini, C., and Meniconi, S. ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. In *Proc. of the IFIP TC13 Int. Conf. on Human-Computer interaction Interact'97* (Sydney, July 14-18, 1997). S. Howard, J. Hammond, G. Lindgaard (eds.). IFIP Conference Proceedings, vol. 96. Chapman & Hall Ltd., London (1997), pp. 362–369.
27. Vanderdonckt, J. Model-Driven Engineering of User Interfaces: Promises, Successes, and Failures. In *Proc. of 5th Annual Romanian Conf. on Human-Computer Interaction ROCHI'2008* (Iasi, September 18-19, 2008), S. Buraga, I. Juvina (eds.). Matrix ROM, Bucarest, 2008, pp. 1–10.
28. Vanderdonckt, J., Limbourg, Q., Florins, M. Deriving the Navigational Structure of a User Interface. In *Proc. of 9th IFIP TC 13 Int. Conf. on Human-Computer Interaction INTERACT'2003* (Zurich, 1-5 September 2003). IOS Press, Amsterdam, 2003, pp. 455-462.
29. Vanderdonckt, J. Model-Driven Engineering of User Interfaces: Promises, Successes, and Failures. *Romanian Journal of Human-Computer Interaction* (Revista Romana de Interactiune Om-Calculator), Vol. 1, 2008, pp. 1–10.
30. Vanderdonckt, J., Coyette, A. Modèles, méthodes et outils de support au prototypage multi-fidélité des interfaces graphiques. *Revue d'Interaction Homme-Machine* 8, 1 (2007), pp. 91-123.
31. Vanderdonckt, J., Coutaz, J., Calvary, G., and Stanculescu, A. Multimodality for Plastic User Interfaces: Models, Methods, and Principles. Chapter 4, in “Mul-

timodal user interfaces: signals and communication technology”, D. Tzovaras (ed.), Lecture Notes in Electrical Engineering, Springer-Verlag, Berlin, 2007, pp. 61-84.

32. Vanderdonckt, J., Furtado, E., Furtado, V., Limbourg, Q., Silva, W., Rodrigues, D., Taddeo, L.. Multi-model

and Multi-level Development of User Interfaces. Chapter 10, in Seffah, A. & Javahery, H. (Eds.), “Multiple User Interfaces - Cross-Platform Applications and Context-Aware Interfaces”, John Wiley & Sons, New York, November 2003, pp. 193-216.

Rule Based Conversion of UsiXML-based GUI Description

Plamen Paskalev, Ilka Serafimova

Department of Computer Systems and Technologies

Technical University

Varna, Bulgaria

+359 52 612 367

{ppaskalev, iserafimova}@eurorisksystems.com

ABSTRACT

The paper considers a realization of an approach for converting a UsiXML-based GUI definition to a proprietary format of a WEB-based application. The conversion is realized as a rule based solution using CLIPS engine. The advantages of the selected approach and the technical realisation are discussed. The results from implementing the solution are debated. The integration with an existing GUI handling solution, storing the definition in a database is discussed.

Keywords

GUI description, UIDL, UsiXML, Rule based, CLIPS, Computer Systems and Technologies

ACM Classification Keywords

D.2.2 [Software Engineering]: Design Tools and Techniques – *User interfaces*. H.5.2 [Information Interfaces and Presentation]: User Interfaces – *Graphical user interfaces*.

INTRODUCTION

The user interface is an essential part of every application. The development of GUI has to answer many requirements – from reasonable and robust handling of the information shown or received to user-friendly, predictable and consistent design. Implementation of the main obligatory parts of the GUI (such as showing or hiding controls or groups of controls, making fields read-only, etc.) leads to program code which is hard to read and support. GUI design is especially complicated in case of complex client-server oriented systems, where large amounts of data have to be shown. The programmer tasks were simplified with introduction of high level programming languages (e.g. Java, C# etc.) and technologies for web based application development (e.g. JSF, ASP.NET, etc.). Nevertheless, the design is still a challenge in case of complex applications.

The problems like different types of hardware on client machines, different resolutions and multi-language support are combined with problems of grouping the data in correspondent logical hierarchy, implementing the logic behind the dialog layer and controlling the input. The approach of development of a separate solution for every special case is not an option due to the enormous amount of work. To answer these problems, the concept of UIDL (User Interface Definition Language) has been introduced. With its foun-

dations in User Interface Management Systems (UIMS), introduced in 1980s [16], the UIDL concept allows designers to describe the interactive behaviour in a high-level form, which gives a higher level of abstraction over the input – output devices.

Another aspect, which adds complication to the user interface handling, is inclusion of some aspects of Intelligent User Interfaces (IUI) in the application logic. The attempts to trace user behaviour, to determine the user expertise level, to maintain a knowledge base with most common user activities, action sequences and failures and, based on this knowledge, to undertake some dynamic changes in the GUI in order to help the user in his interaction with the application make the GUI design even more difficult and complex.

The basic theoretical matters and related works will be covered by the end of the introduction. The technologies used in the proposed solution will be discussed in the next two chapters. Further, the prototype tasks and goals will be defined. A detailed module description will follow. And finally, conclusions about the research will be made and future works will be stated.

Graphic user interface generation

The concept of user interface description language (UIDL) allows designers to abstract the description of the GUI and thus separate it from the application business logic. This abstraction gives the possibility to use the description in different manners. It can be converted during application release building to a static user interface implementation or it can be interpreted at runtime. The implementation of the user interface management systems (UIMS) based on the used UIDL is crucial for the application design. The former lightens the designers work, but also allows easy GUI consistency support across the application [19].

However, problems like isolating the designer from the control of the low level details in the visualization and other UIMS usage problems, as well as the standardization of user interface components did not allow a wide acceptance of the UIDL concept [15].

In the next years the concept of UIDL solved some of the above mentioned issues like different screen resolution, multi-platform applications, multi-language support with the introduction of new interaction technologies and devices.

The objectives of UIDL implementations are [12]:

- To capture the requirements for a user interface as an abstract definition that remains stable across a variety of platforms.
- To enable the creation of a single UI design for multiple devices and platforms.
- To improve the reusability of a user interface.
- To support evolution, extensibility and adaptability of a user interface.
- To enable automated generation of user interface code.

Related work

The usage of XML as a UIDL base is very widespread. There are numerous suggestions such as UIML [6], XML [5], TERESA XML [15], etc. The solutions [20] using data base description are not universal but tightly task-dependant and are used in applications with many different users. The usage of database description of GUI was discussed in [18]. A solution for generation of unified GUI generation, based on modular components is proposed in [7].

GUI DEFINITION USING USIXML

One of the UIDL solutions and frameworks developed in recent years is UsiXML [8,9,10]. It is an XML-based context-dependent UIDL which provides various levels of abstraction: context-independence, platform-independence, etc. UsiXML allows specifying of multiple models involved in user interface design such as: task, domain, presentation, dialog, and context of use, which is in turn decomposed into user, platform, and environment. These models are structured according to the four layers of the Cameleon Reference Framework (CRF) [1]: task and concepts, abstract user interface (AUI), concrete user interface (CUI), and final user interface (FUI). Intermodel mapping is used to support relationships between these models [8].

A FUI lies at the bottom of the Cameleon framework. It is platform dependent and is either interpreted (e.g., through a Web browser) or executed (e.g., after compilation of code in an interactive development environment). A CUI abstracts the UI definition as independent from any computing platform but it is environment dependent. A CUI is also considered a reification of an AUI at the upper level and an abstraction of the FUI with respect to the platform.

An AUI abstracts the UI definition as independent of any modality of interaction (e.g., graphical interaction, vocal interaction, speech synthesis, etc.). An AUI is considered as an abstraction of a CUI with respect to modality. At the top of the framework is the Task and Concepts level where the interactive task carried out by the end user is defined according to their viewpoint. Task and Concepts are con-

sidered class instances representing the concepts manipulated [8,9].

UsiXML is a preferred technology for the research of this article because it is an XML based UIDL with a wide range of possibilities and several GUI abstraction levels. It has a number of user-friendly and easy-to-use editors (such as GraphiXML [13] and SketchiXML [2]) thus covering one of the main problems of the previous solution, discussed in [17, 18].

For the purpose of this experiment CUI description is used since the solution realizes a graphic user interface which is environment dependent. The concrete JSF interpreter is used as a testing tool for the generated or converted GUI descriptions.

USER INTERFACE DYNAMIC MODIFICATION

Applications which interact with people allow the introduction of intelligent and dynamic behaviour which improves the communication between the former and the latter. The details of user input can be gathered and afterwards examined using intelligence engine. The user interface is then rearranged on the bases of that analysis. The enhancements concern:

- Dynamic control of complicated application views with large amount of heterogeneous data to be shown (e.g. health information systems, financial software, complex industry controlling software, etc.) to reduce the level of complexity for easier development and testing.
- Automatic construction of intuitive user interfaces according to the user experience to facilitate their interaction with the system and decrease the necessity of regular on-line help and application description usage.
- Adaptation of the user interface to changing user expertise level to provide detailed information depending on the user knowledge and goals.

An approach, used for the realization of the GUI of a WEB based application which realizes some aspects of IUI was presented in [16]. The main goal of that project was to create an environment, which would allow an easy extension, manipulation and dynamic modification of the GUI, based on the

- a) data to be displayed,
- b) investigation of the behaviour of the user.

The kernel of the project was realized as a rule-based solution (Figure 1, [18]) based on a CLIPS [4] engine. The advantages of the CLIPS engine usage were discussed in [17, 18]. Several layers of rules were developed for this purpose. That prototype solution was based on GUI description, located in a database.

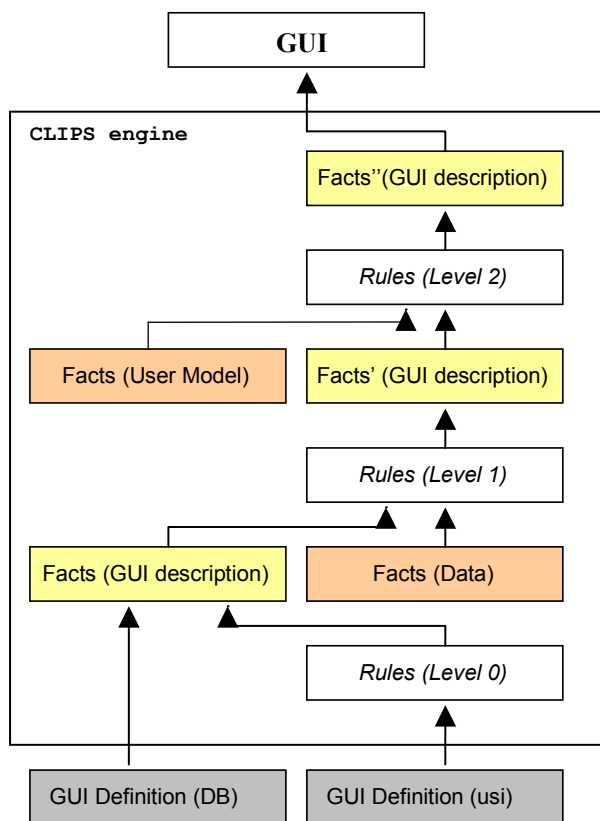


Figure 1. GUI Dynamic Modification project.

Although the results from the tests with the application, built using that approach, were positive, it became clear, that some extensions are necessary. The problems which were identified during the tests can be grouped in the following two categories:

- *Development problems* – during development of the test user interfaces it became clear, that the development of the modules without graphical tool is a time and effort consuming task. The absence of validation for the GUI description was also reported as a problem during the development stage.
- *Run time limitations* – Keeping of the GUI definition in the database, despite the advantages, contains some drawbacks, especially in cases, when the application has more simple structure and is not using database itself.

The current article discusses the attempt of the authors to solve these problems using UsiXML and implementing a rule-based converter between the GUI descriptions, built using the two approaches. The input and the output of this module have identical structure thus it is transparent to the GUI interpreter.

The conversion between both descriptions can be realized in different ways. The rule-based approach is the natural solution because it provides flexibility; ease of integration

with the existing solution and ability to cooperate with the other layers of GUI manipulation (Figure 1) which are developed in the same way. The conversion with the rule-based engine and the usage of the same internal representation in both cases (UsiXML and database description) makes the form of the structure transparent. In this way the application logic, which uses and manipulates the GUI description, remains unchanged.

TASKS AND GOALS

Several aims are defined for the research application prototype of this article. They arose naturally as an evolution of a previous development [17, 18]. The solution should use UsiXML as a GUI description language as well as previously used database description. The application has to provide a converting mechanism which can transform UsiXML to database GUI description and vice versa. The following advantages are pursued while designing the prototype:

- The discussed solution attempts to provide a graphic environment (a UsiXML editor) for GUI description definition in order to improve the usability of the proposed interpreter. The approach should allow designers who are not familiar with the underlying technologies to describe GUI fast and easy. The tool should be used to define new or rearrange existing application views without the need to recompile the application for them to take place in the system.
- The XML schemas should provide validation of the description at an initial level to prevent propagation of problems in the system.
- The solution, discussed in [17, 18] uses GUI definition, stored in a database. In some cases the usage of a GUI description, located in XML files on the server is the preferable solution. For some applications all the additional tasks related to the “GUI in DB” approach, including the database access, administration and user restrictions, are too complex and therefore must be avoided. The application developers will be able to fulfil this requirement using the suggested solution.
- The discussed tool should provide easy-to-use export/import functionality. UsiXML description gives the opportunity to realize the latter since the XML standards are widely used in case of server-server communication and in case of web services.

However, the proposed interpreter should combine the abovementioned advantages with the ones of a description of the GUI placed in a database. The most essential of them are:

- The usage of the database as a description container will enhance the reusability of groups of logically connected controls or sub-groups among different views in the application avoiding duplication. The groups can

be used as building elements for complex structures thus improving readability.

- The database organization allows the definition of so called exclusion rules for hiding single controls or groups in a specific hierarchy of components thus implying *polymorphism*.
- The database supplies straightforward utilities of user authorization policy.

Both UsiXML and database descriptions are platform independent. As far as both realize the UIDL concept they isolate the GUI description from the domain business logic. They are flexible, robust and the parsing of a description written in any of the former is not time consuming. The conversion mechanism between the two provides the interpreter with the combination of the UsiXML and database description advantages.

SOLUTION OVERVIEW

Application structure

The approach discussed in this article consists of (Figure 2):

- XML description – with respect to UsiXML schema of CUI description. In the current solution GrafiXML [13] editor is used to generate it;
- User interface description structure in a database;
- Converter, which translates UsiXML to database description and vice versa – an intelligence engine realized in CLIPS;

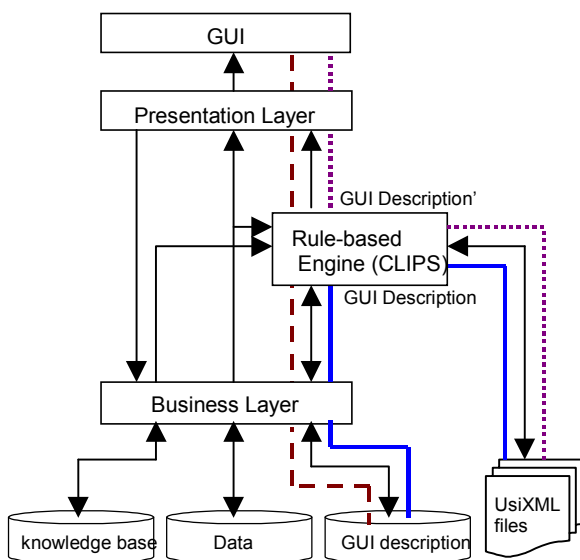


Figure 2. GUI generation process.

- Intelligence module for GUI enhancement [17].
- Description interpreter realized in Java [18].

The solution is integrated in a server-side WEB application, designed to work with financial information (financial instruments, environments, etc.). The description is organized into groups of components, which facilitate their reuse in visualization of different instruments. The discussed approach is applicable in any domain where complex data is arranged in reusable logical units.

Figure 2 presents the process of the GUI generation. First the GUI description is built. It can be extracted from the database directly using SQL statements over the description tables. On the other hand the former can be obtained from a UsiXML description. The controls from the latter are loaded into the CLIPS engine as facts and after some rules have been applied to them the modified facts are converted back and proceeded to the business layer.

On the next step additional information is extracted from the database: information about the currently selected instrument, which is also hierarchically organized (instrument, legs, cash flows, etc.), and information about the current user activities. All these three, including the description, are passed to the CLIPS engine once more for the GUI to be enhanced by the application of other rules. The format of Java objects, used before and after the conversion to CLIPS facts is identical, so the inclusion of the rule-based approach does not need any additional changes in the data representation and logic on the application server. Finally, the description is translated to a JSF component tree and displayed to the user.

The separation between the two passes in the CLIPS engine allows a background conversion between UsiXML and database formats. Thus a UsiXML description can be imported into the database without a graphical representation, so does the export of a database description into a UsiXML file.

Business objects

The GUI description hierarchy consists of (Figure 3):

- Controls (leaves in the hierarchy) which describe simple GUI controls like inputs and selects;
- Groups which contain controls and/or other groups (sub-groups). The groups on the highest level have special interpretation. For example, part of them represents tab control in a common tab holder of a form;
- Forms, defined with their identifiers, describing the views of the application;
- An instrument defines the financial instrument type to be shown on the GUI.

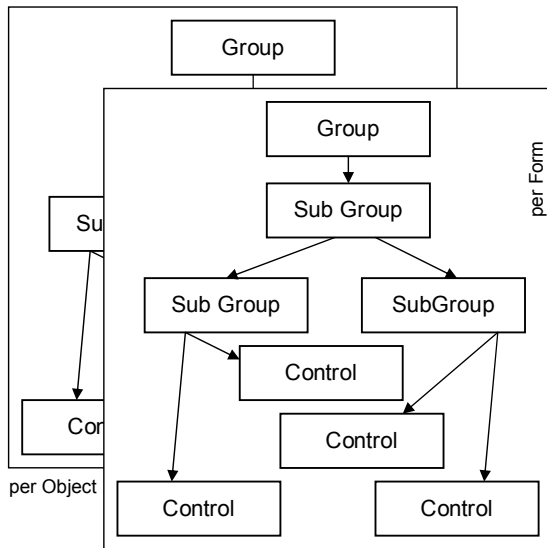


Figure 3. GUI description structure.

Description element belongings such as control-group, group-group or group-instrument relations are also defined along with the (X; Y) coordinates of the element in its parent. Other components of the element description are control type, data binding, validation and presentation information, etc. The discussed solution supports complex control types like tables and charts.

SOLUTION KERNEL

XML Description

The XML schema used in the discussed solution is developed by the UsiXML team [14]. GrafiXML [13] was used to generate the example for this article (Figure 4). Using the above mentioned editor a simple CUI description was generated. It was edited by adding custom attributes to its elements supplying specific information for the JSF description like validator, converter, etc. A fragment of the result .usi file is shown in Figure 5.

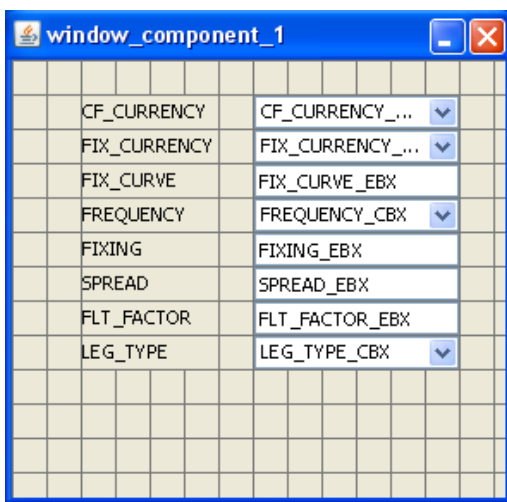


Figure 4. GrafiXML editor.

```
<cuiModel id="p-test-v6-cui_10" name="p-test-v6-cui">
  <window id="window_component_1" name="window_component_1"
    width="297" height="290">
    <gridBagBox id="RATES_018" name="RATES"
      gridHeight="14" gridWidth="14">
      <constraint gridx="2" gridy="5" gridwidth="4"
        gridheight="1" weightx="1.0" weighty="1.0"
        fill="both" insets="0,0,0,0">
        <outputText id="FIXING_LBL_000"
          name="FIXING" isVisible="true"
          isEnabled="true" isBold="true" textColor="#000000"/>
      </constraint>
      <constraint gridx="7" gridy="5" gridwidth="6"
        gridheight="1" weightx="1.0" weighty="1.0"
        fill="both" insets="0,0,0,0">
        <inputText id="FIXING_EBX_032"
          name="FIXING_EBX" isVisible="true"
          isEnabled="true" textColor="#000000"
          maxLength="50" numberOfColumns="15" isEditable="true"
          istyle="text-align: right" itype="RW"
          iAccessRule="" iDataBind=".fixing" iValidatorID=""
          iConverterID="javax.faces.Number;pattern=# ##0.00%"/>
      </constraint>
    </gridBagBox>
  </window>
</cuiModel>
```

Figure 5. .usi file example fragment.

```
(defrule read_only_type
  ?ro_control<-(control_template
    (type ?itype&"false" | "null"))
=>
  (modify ?ro_control (type "RO"))
)

(defrule read_write_type
  ?rw_control<-(control_template
    (type "true"))
=>
  (modify ?rw_control (type "RW"))
)

(defrule map_controls
  ?lbl_control<-(control_template (controlType "Label") (type "RO"))
  (label ?n1)(addPosition ?ap1) (labelStyle ?ls1))
  ?obj_control<-(control_template (controlType ~"Label") (addPosition ?ap2))
=>
  (if (= ?ap1 ?ap2) then
    (modify ?obj_control
      (label ?n1)
      (labelStyle ?ls1)
      (addPosition 0))
    (retract ?lbl_control)
  )
)
```

Figure 6. Description converter example rule.

GUI Description Converter

The main module of the discussed solution is the converter that translates a description in one format to another format. If a description is defined in a UsiXML graphic editor it is read from the .usi file to a Java tree and the tree is processed to the intelligence CLIPS engine as facts. After that, some rules are applied to the latter altering the knowledge base (Figure 6).

Next a new tree with the same structure as the input one is formed based on the final facts. The resultant Java tree can then be stored as a database description to the aforementioned database tables or it can be directly interpreted to a JSF component and then displayed in the user browser.

The mechanism can work in the opposite direction. A description might be read from the database, then processed to the CLIPS engine and finally stored as a .usi file or interpreted as HTML.

The rules applied to the CLIPS facts adjust the structure of the tree nodes. For example, in the database descriptions a control element contains information (Figure 7) about the control and its adjacent label (e.g., the label “Market price” with the value of 100.00) but the in UsiXML description the label and the control are specified as separate XML elements.

The presentations in all the levels (JSF component tree, Java internal representation, CLIPS template definitions) are the same, regardless of the type of the used source. After applying the rules on the facts, defining the UsiXML components, they are converted to facts, fully compatible with the ones, created based on the database source.

GUI DESCRIPTION INTERPRETER

After being loaded the GUI description is represented as a Java tree in spite of the source (database or XML). Next, it is converted to a JSF component tree. Java Server Faces (JSF) is a Java based WEB technology which was chosen for a variety of reason like support from any Java container (e.g. Tomcat, Apache, etc.); capacity; easy extensibility [3], etc. The interpreter is realized as a custom component which simulates a tab control. The latter renders only the necessary data according to the selected tab.

The generated tree has the following structure:

root - attached to the standard JSF view tree;
 `- *1st level* - tab views;
 `- *2nd level* - layout tables;
 `- ... nested layout tables;
 `- *leaves* - simple or complex controls.

The JSF component tree consists of:

- layout tables which arrange elements on the page;
- pairs of label and input control for scalar values;
- data tables presenting collections;
- action components which invoke actions from the model.

This GUI generation approach creates only the necessary controls during the rendering phase. Thus the discussed solution improves the readability of the view [18]. The visualization of the group from Figure 4, Figure 5 can be seen on Figure 8.

ID	GROUP_ID	STYLE	CONTROL	DATA_BIND	VALIDATOR_TYPE	LABEL_ID	CONVERTOR_TYPE
24	15 (null)		DropDown	position	com.ers.pm1j.ex.web....	POSITION	com.ers.pm1j.ex.web....
25	19 text-align:...		Edit	volume	(null)	VOLUME	javax.faces.Number;p...
26	19 text-align:...		Edit	nominal	(null)	NOMINAL	javax.faces.Number;p...

Figure 7. Description in the database (fragment).

Rates	
CF currency	EUR
Fix currency	EUR
Fix curve	
Frequency	Semi-annually
Fixing	4.00 %
Spread	0.01 %
FLT factor	100.00 %

Figure 8. Interpreted description result.

DATABASE GUI DESCRIPTION

Figure 9 shows the relationships between the GUI description tables. The latter are as follows:

- *INSTRUMENT_TYPE* (nomenclature table) – contains the definition of all instrument types, the GUI is capable to handle. They have a type-subtype pair and a unique identifier.

- *FORM_TYPE* (nomenclature table) – contains the definition of the views, the application consists of. Every form has a unique identifier.
- *GROUP_DATA* – contains groups definition: identification information (unique identifier, group type) and visualization information (relative positions in both directions, group style, label, etc.).
- *CONTROL_DATA* – contains controls definition: control type, relative position, identifier of the control, validating, type of the conversion of the input, access (read-only, read/write), etc.
- *GROUP_GROUP_REL* (relational table) – defines the hierarchy of the groups.
- *INSTRUMENT_GROUP_REL*, *INSTRUMENT_CONTROL_REL*, *INSTRUMENT_CONTROL_REL_OPP* (relational tables) – give additional constraints and rules for displaying or hiding groups and controls per instrument respectively.

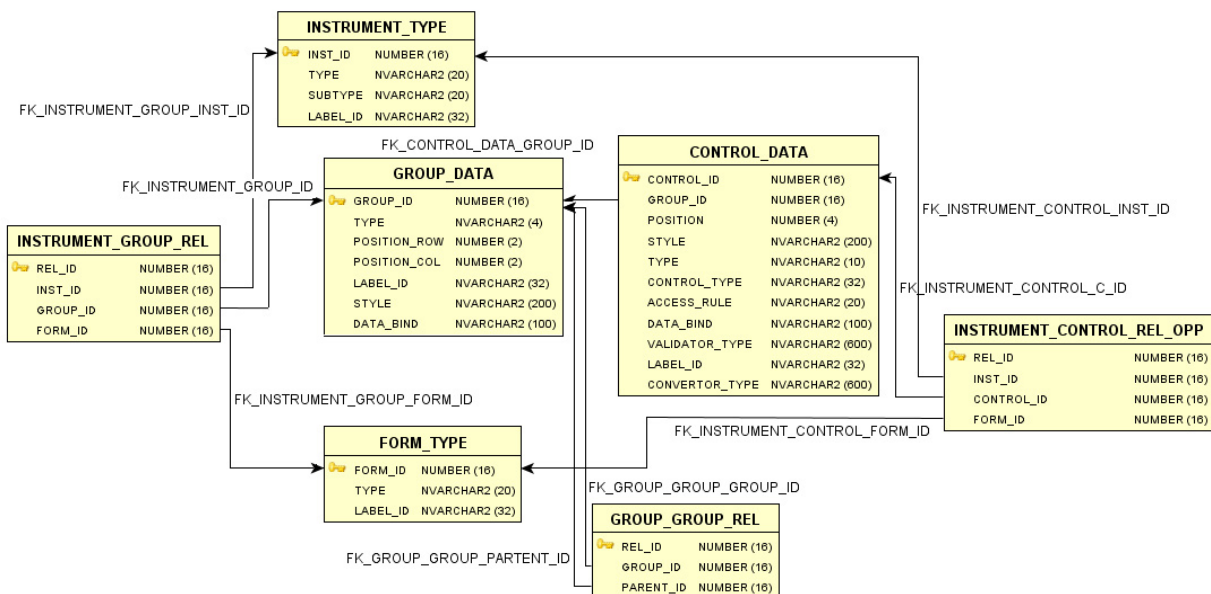


Figure 9. Database description tables.

CONCLUSION AND FUTURE WORK

The discussed solution was developed and tested using sample data. Some results can be seen on Figure 8. The following conclusions were made:

- The development of a dynamic interface is more time consuming, compared to direct hard coding the GUI. However this investment will pay itself back in

later changes and adaptations of the system because of its higher flexibility.

- No delays were encountered when processing a description in either of the formats.
- Both UIDLs are more flexible and simple than using the standard rendering checks.

- The database-oriented approach benefits from the easier maintenance and higher availability of the data centre.
- It allows reuse of groups of components with a single add. Changes in the group take place in all referencing hierarchies (one change per all hierarchies);
- Using XML for description definition allows easy and intuitive input validation.
- Graphic editors are much more intuitive than any 3rd party database tool or text editor.
- Tree structured XML files are easier to understand and update in a text editor unlike the database records.
- The latter however provide recursive references to specific definition unlike XML tree structure.
- The rule-based approach provides an easy-to-extend framework, where other cases can be handled by adding new rules.

Advantages and Disadvantages of the Approach

The implemented solution combines the advantages of both XML and database approaches. They are both platform independent and cross-technology (they can be processed by Java as well as .NET). Changes in any of the formats take place immediately in the application without the need of recompilation and/or redeployment. Also, access to the GUI description is fast and robust supported by numerous libraries providing easy treatment.

Using a database description improves the reusability of the group and/or control definitions. If used in a desktop application with a centralized database changes are applied simultaneously to all the database users. However, this type of description does not allow a built-in validation of the input structure as opposed to XML schemas. Moreover, XML description, unlike database one, can be changed in any simple text editor. Furthermore, UsiXML provides ready to use graphic editors which eases the GUI definition even more.

On the other hand, being separately developed languages the database description UIDL and the UsiXML specification are not entirely overlapping. This fact limits the possible elements used to define a GUI description. The database UIDL is JSF compliant which is too specific for the UsiXML concept but at the same time increases variety of components used for GUI definition which is very important for the needs of the application using the discussed solution.

REFERENCES

1. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J., A Unifying Reference Framework for Multi-Target User Interfaces. *Interacting with Computers* 15, 3 (2003), 289–308.
2. Coyette, A., Kieffer, S., and Vanderdonckt, J. Multi-Fidelity Prototyping of User Interfaces. In *Proc. of 11th IFIP TC 13 Int. Conf. on Human-Computer Interaction Interact'2007* (Rio de Janeiro, September 10-14, 2007), Lecture Notes in Computer Science, Vol. 4662, Springer-Verlag, Berlin, 2007, pp. 149-162
3. Geary D., Horstmann C. S, Core Javasever Faces [Jsf], *Prentice Hall PTR*, June 2004
4. Giarratano J. Expert Systems: Principles And Programming, 4Ed, ISBN: 053438447, CourseTec Ltd. 2004
5. Eisenstein, J., Vanderdonckt, J., Puerta, A., *Model-Based User-Interface Development Techniques for Mobile Computing*, Proc. of 5th ACM Int. Conf. on Intelligent User Interfaces IUI'2001 (Santa Fe, 14-17 January 2001), J. Lester (ed.), ACM Press, New York, 2001, pp. 69-76.
6. Helms, J., Schaefer, R., Luyten, K., Vermeulen, J., Abrams, M., Coyette, A., Vanderdonckt, J., *Human-Centered Engineering with the User Interface Markup Language*, in Seffah, A., Vanderdonckt, J., Desmarais, M. (eds.), "Human-Centered Software Engineering", Chapter 7, HCI Series, Springer, London, 2009, pp. 141-173.
7. Hoiem, D., Lovell, M., Method and apparatus for a unified user interface", US patent 7,512,899, 31 March 2009, Microsoft Corporation, <http://www.freepatentsonline.com/7512899.htm>
8. Limbourg, Q., Vanderdonckt, J., *UsiXML: A User Interface Description Language Supporting Multiple Levels of Independence*, in Matera, M., Comai, S. (Eds.), "Engineering Advanced Web Applications", Rinton Press, Paramus, 2004, pp. 325-338.
9. Limbourg, Q., Vanderdonckt, J. Multi-Path Transformational Development of User Interfaces with Graph Transformations. In Seffah, A., Vanderdonckt, J., Desmarais, M. (eds.), "Human-Centered Software Engineering", Chapter 6, HCI Series, Springer, London, 2009, pp. 109-140.
10. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Florins, M., Trevisan, D., *UsiXML: A User Interface Description Language for Context-Sensitive User Interfaces*, in Proc. of the ACM AVI'2004 Workshop "Developing User Interfaces with XML: Advances on User Interface Description Languages" (Gallipoli, May 25, 2004), K. Luyten, M. Abrams, Q. Limbourg, J. Vanderdonckt (eds.), Gallipoli, 2004, pp. 55-62. Accessible at <http://www.edm.luc.ac.be/uixml2004/index.php?selected=program>
11. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Lopez, V. UsiXML: a Language Sup-

- porting Multi-Path Development of User Interfaces. In *Proc. of 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems EHCI-DSVIS'2004* (Hamburg, July 11-13, 2004). Lecture Notes in Computer Science, Vol. 3425, Springer-Verlag, Berlin, 2005, pp. 200-220.
12. Luyten, K., Abrams, M., Limbourg, Q., Vanderdonck, J., *Proceedings of the ACM AVI'2004 Workshop "Developing User Interfaces with XML: Advances on User Interface Description Languages"* UIXML'04 (Gallipoli, May 25, 2004), Gallipoli, 2004. Accessible at <http://www.edm.luc.ac.be/-uixml2004/index.php?selected=program>.
 13. Michotte, B., Vanderdonck, J., *GrafiXML, A Multi-Target User Interface Builder based on UsiXML*, Proc. of 4th International Conference on Autonomic and Autonomous Systems ICAS'2008 (Gosier, 16-21 March 2008), D. Greenwood, M. Grottke, H. Lutfiyya, M. Popescu (eds.), IEEE Computer Society Press, Los Alamitos, 2008, pp. 15-22.
 14. Michotte, B., et al. UsiXML CUI schema specification team, <http://www.usixml.org/index.php?mod=pages&id=5>, 2006
 15. Mori, G., Paternò, F. and Santoro, C., Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions, *IEEE Transactions on Software Engineering*, vol. 30, issue 8, 2004.
 16. Myers B., Hudson S. E., Pausch R., Past, Present, and Future of User Interface Software Tools, *ACM Transactions on Computer-Human Interaction*, Vol. 7, No. 1, March 2000.
 17. Paskalev, P., Rule based GUI modification and adaptation. In *Proceedings of International Conference CompSysTech'09*, Bulgaria, June 2009
 18. Paskalev, P., Serafimova, I., Runtime Generation of an User Interface, Described in a Database. In *Proceedings of International Conference CompSysTech'09*, June 2009
 19. Shaer O., Green M., Jacob R., Luyten K., User Interface Description Languages for Next Generation User Interfaces Workshop. In *Proceedings of CHI 2008*, Florence, Italy, 2008.
 20. van de Vegte, J., Som de Cerff, W.J., A Dynamic GUI for accessing the Netherlands SCIAMACHY Data Center, Montreal, Canada, May 2000.

Adapting UsiXML User Interfaces to Cultural Background

Iyad Khaddam

Consulting Company for Computers
& Communications (4C) - Consultant
Damascus, Syria
+963 944 266 475
iyadkh@gmail.com

Jean Vanderdonckt

Louvain School of Management, UCL
Place des Doyens, 1
B-1348 Louvain-la-Neuve (Belgium)
+32 10 478525
jean.vanderdonckt@uclouvain.be

ABSTRACT

Adapting a user interface to the end user's cultural background today remains an open challenge since many underlying issues are not yet solved. This paper addresses this challenge by reporting on a selected series of these issues, by structuring them according to Nielsen's linguistic model of interaction, and by discussing how each issue can be supported by incorporating its solution into a User Interface Description Language, such as User Interface eXtensible Markup Language (UsiXML), at the level of a Concrete User Interface (CUI). In particular, the problem of right-to-left (RTL) versus left-to-right languages (LTR) languages is discussed through a series of adaptations of algorithms and techniques that support the automated generation of Arabain graphical user interfaces based on UsiXML.

Categories and Subject Descriptors

H.1.2 [Information Systems]: Models and Principles – *User/Machine Systems*. H5.2 [Information interfaces and presentation]: User Interfaces – *Prototyping; user-centered design; user interface management systems (UIMS)*.

General Terms

Design.

Author Keywords

Arabization, Cultural background, Globalisation, Localisation, LTR, RTL, UsiXML.

INTRODUCTION

Right To Left (RTL) languages are languages that are written from Right to Left, like: Arabic, Farsi, Urdu and Hebrew. They use a different set of letters than Latin counterparts. Similarly, some languages are read from bottom to top, as opposed to Latin languages that are read from top to bottom. We hereby define a RTL UI as a Graphical User Interface that accommodates the requirements posed by languages that are read from right to left. A RTL UI is the UI that meets the RTL language demands. RTL UI is an underestimated concept in the HCI community. At first glance, it may look like support for another set of languages, and sometimes it may be assumed as a localization of the product UI. Designing a RTL UI cannot be assimilated to designing a LTR UI and mirroring it by symmetry in order to obtain a corresponding RTL UI. Indeed, reading paths are different, location of widgets cannot be simply

mirrored because of labels lengths, explanation and character sets. Existing algorithms for UI automated layout do not consider these aspects at all.

RTL UI has two aspects that should be treated together: the localization and the orientation (mirroring). If we fail to address one of them, the resulting UI will not be acceptable to users who are RTL language speakers. UI *localization* is addressed in multiple works, like in [20]. RTL is a common property among a set of written languages. This implies that each of these languages needs to be localized separately. Another less common property among the RTL languages is that most of them use the Arabic alphabet but add some extended letters (Hebrew has its own alphabet). Arabic can represent all the RTL languages as it has many features which do not exist in the rest of the RTL languages. Therefore, support for Arabic will enable support for the rest of the RTL languages that are similar in principle.

Figure 1 shows the characteristics of RTL UI versus LTR UI. We can easily notice the mirroring effect. The form header is mirrored, and components order is reversed. The Close, Maximize, Minimize, Caption and Icon controls are all rearranged from right to left. The same effect applies to the menu and the tabs. Inside the tab, we can see that the table also is mirrored. The first column (that contains the labels: Name, Last name ...) is positioned to the right. Table columns now flow from right to left which is the mirror of the LTR table layout. The vertical scroll is on the right, "Save" and "Revert" buttons positions are mirrored.

RTL affects controls too. We note the change of writing direction inside the text boxes. The combo box rendering is also mirrored (the "Gender"), the drop down image is positioned to the right of the combo box in the RTL version. The check box caption is also mirrored. Menus are also affected. In Figure 2, we show the effect of RTL on menus and sub menus, they expand to the right in the RTL version. The triangle image before the menu item "Transform..." is mirrored, and also is the arrow after.

An interesting difference to note is the horizontal bar. Although a horizontal bar is horizontally symmetrical, but the scroll bar starts from the left in LTR UI and from the right in the RTL UI. This is important to note in case we shrink a RTL window; the right view (controls on right) should always be visible and no need to scroll to the right.

Another interesting difference can be seen by looking at the "Notes" text box control. This control allows multiple

lines, Pressing “Enter” key takes the cursor to the beginning of new line (\n + \r). While “beginning a new line” means in LTR “the left of the line”, RTL redefines this meaning as “the right of the line”. Writing direction switching may occur in a RTL text. The “Notes” field (in the RTL version) provides an example. The direction is switched using a combination of keyboard keys. In this example, the user wrote in Arabic, switched direction to LTR then back forth to RTL to continue his phrase. This can occur many times, and should not affect the readability of text.

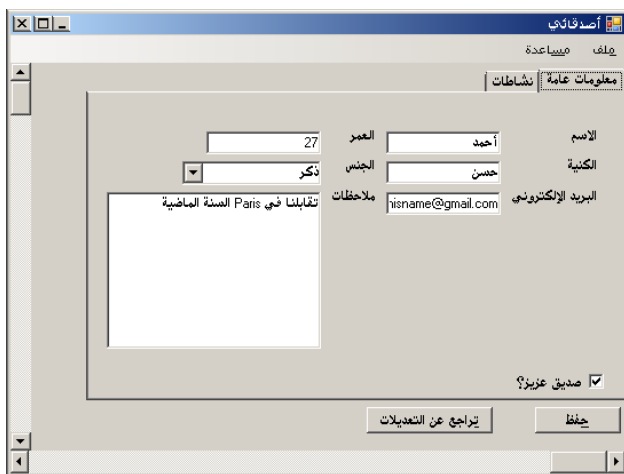
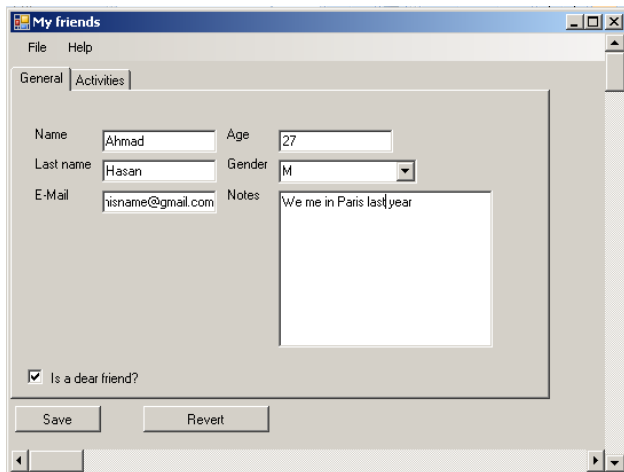


Figure 1. a sample English UI and the localized RTL version (in Arabic Language)

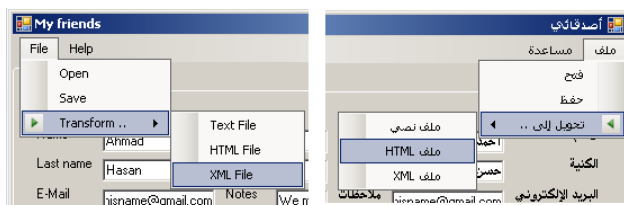


Figure 2. A sample LTR and the localized RTL version (in Arabic language).

“Tab” behavior is also affected with RTL (Tab to move focus to next enabled element). In the LTR version, pressing

“tab” moves focus between controls in sequence: “Name”, “Last name”, “E-mail”, “Age”, “Gender” then “Notes”. The same behavior is expected in the RTL version. If the “Tab” key behavior is related to element positioning, it won’t work in the RTL version (unless positioning considers RTL). The last thing to notice is the shortcuts. As RTL languages use non-Latin alphabet, keyboard shortcuts will be different in both versions. The RTL version will use keyboard shortcuts from its own language alphabet. This problem is related to Localization more than to RTL.

From the above example, we summarize RTL effect on UI in 2 ways:

- The orientation (the mirroring)
- The localization

The TRL localization has special characteristics that don’t exist in Latin-alphabet languages, which makes the localization a challenge. We identify from the examples before the following characteristics for RTL localization:

- Text localization
 - o Text localization: language encoding and character set (alphabet).
 - o Direction switching: direction of text writing.
- Graphics localization
 - o RTL sensitive graphics (non-horizontally symmetrical)
 - o Images with text inside
 - o Other localizable images (country flag...)
- Control localization
 - o Control rendering: ex: label control should support writing from right to left.
 - o Control behavior: controls should be aware of special behavior for special keys (like pressing “enter” key in a text area)

In the following sections, we discuss the support provided by the current version of UsiXML and explain our contribution to provide a full support for RTL.

RELATED WORK

As we mentioned before, the RTL issue is underestimated in the CHI research area. While localization is discussed in many works, rare effort was spent on RTL languages localization.

In the market, we can find well-arabized products and most of them are built on Microsoft Windows Operating System (As it was the pioneer in providing support for RTL languages) or on the web. ERP products were forced to provide RTL support due to market pressure. Hau in [7] shows the awareness of the RTL issue in the ERP industry.

Writers [5] explain the Arabic language characteristics and explain the challenges of the language in the context of providing OS support for Arabic. They discuss the issues related to encoding, character shaping and the “cursive” or “handwritten” style of writing in Arabic, vowels, numbers shapes and the mirroring effect on visual screens.

Rejmer et al. [20] discussed internationalization for a product as a case study. They came up with a set of guidelines to help internationalization/localization of a product. Their work handled the case of western languages (Latin alphabet) and didn’t discuss the case of RTL nor non-Latin alphabet languages. Their guidelines need reviewing to be adapted for RTL languages.

Other UI languages address the RTL in different ways. XUL [27] (XML User Interface Language) is the UI language used by Mozilla to create feature-rich cross platform applications. Firefox UI is built using this language.

XUL supports RTL UI by providing the “dir” property at the UI element which is the base of all elements. The “dir” property can have one of two values: normal, reverse. The “normal” means: “position elements in the container according to their order in the xml file”. The “reverse” value means: “position elements in the container according to their *reverse* order in the xml file”. Figure 3 shows the difference. XUL does not directly address the RTL concept, but their concept of “reverse” fixes the orientation issue. On the other side, it doesn’t address the control localization nor provide support to it. When the dir property is “reverse”, this doesn’t imply that the final control to be used is a control that supports RTL. XUL depends on the rendering framework to determine the final control (localized version). Thus it provides a localized version of Firefox for each language. The problem we note with XUL approach is that each localized version of the product will have a localized version of the design (the XUL file is copied for each language). This imposes a maintenance/update problem.

XAML is a markup system that underlies user interface components of Microsoft’s .NET Framework 3.0 and above [26]. XAML supports RTL by adding a “FlowDirection” property to the containers and UI elements that takes one of the values: “LeftToRight”, “RightToLeft”. This causes the expected effect of RTL to be applied on the container and/or the element. Figure 4 gives an example.

The FlowDirection property is inherited by all the elements in the objects hierarchy; all the elements in a container inherit its container’s property. Thus, we only need to set this property on the window level and all the inside elements will inherit it. Note the effect on the combo box, it becomes RTL in response to the window FlowDirection property value.

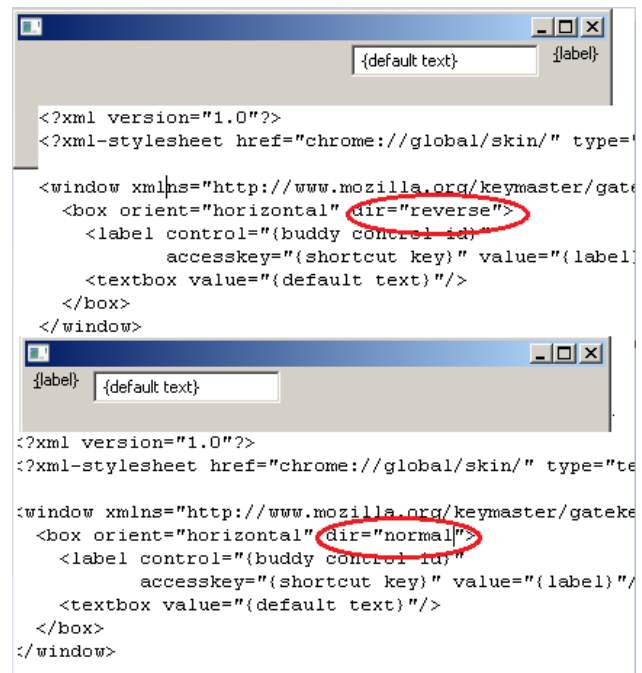


Figure 3. XUL property dir and effect of the 2 different values: normal and reverse.



Figure 4. XAML property FlowDirection and effect of the 2 different values: LeftToRight and RightToLeft.

The FlowDirection property can be overridden on the element’s level, this enables the designer to give a certain element a different direction than the container. This can be helpful in UI where a mixed content of LTR and RTL languages are used.

The problem XAML faces is that when we have a set of RTL interfaces (windows), then each must be set individually. This is because each file defines a window only. The other problem is that we need to copy the design (the XAML file) to obtain the two UIs (RTL and LTR) as the FlowDirection property value is static.

CURRENT RTL SUPPORT IN UsiXML

Architectural overview

UsiXML is MDA-compliant. The UsiXML development process [23] aligns the UsiXML models with the MDA models as follows:

MDA	UsiXML model
Computing Independent Model (PIM)	Task, Domain
Platform-Independent Model (PIM)	Abstract User Interface (AUI)
Platform-Specific Model (PSM)	Concrete User Interface (CUI)
Code	Final User Interface (FUI)

Table 1. UsiXML compliance to MDA.

RTL is a platform-specific issue. This implies that supporting RTL in UsiXML should be considered in the CUI model. RTL should not affect the Task, Domain or AUI models to keep UsiXML MDA-compliant.

The orientation

Current version of UsiXML doesn't make any difference between LTR and RTL, and is assumed to be LTR. UsiXML supports element positioning and alignment (Right, Left and Center). Positioning is supported by the employment of 2dGraphicalContainer objects. Positioning is different from orientation. Figure 5 shows this difference on a table.

1	2	3	3	2	1
4	5	6	6	5	4
7	8	9	9	8	7

LTR Table

RTL Table

Figure 5. The difference between positioning and orientation.

RTL Localization in UsiXML

Localization is supported in UsiXML using the resource model, which provides limited support for RTL languages localization. The problems with current implementation of localization in UsiXML are as follows:

Text Translation

UsiXML supports text localization. The same technique is applicable for RTL languages. Anyway, the issue in RTL text translation is in the storage of local text inside the xml file. The encoding used to save the xml file needs to support the RTL language. Using Unicode can be a solution for this issue.

Direction switching

UsiXML does not sense the change of writing direction in a text. It relies on the input/output FUI objects to do the rendering. This is a concern in case the target platform doesn't support RTL, or if output FUI objects don't support RTL. Enabling writing direction-change sensing would provide information to transformers (from CUI to FUI) to

enhance the generated UI in case FUI controls don't support RTL (for example: a transformer may render the RTL language text as an image instead of text).

Graphics Localization:

All the issues we mentioned before regarding graphics localization can be solved using the current UsiXML localization support. If we have an English UI resource file with n images, the localized UI resource for another LTR language (say French) will contain n images also but with m ($m \leq n$) localised images (ex. flag images and images with text). In the case of localization for an RTL language, the number of localized images may be larger than m . This is due to the non-horizontally symmetrical images that may exist in the original design (ex. Horizontal arrows need to be RTL localized). We will address this issue in the next section.

Control Localization:

UsiXML does not support any kind of control RTL localization. All CUI controls are direction insensitive. This is also left for the FUI objects. The problems with leaving them to the FUI are:

- 1- The control's direction property is hidden in the transformer. Reverse engineering to CUI will fail to identify the control's direction.
- 2- No transformer can generate a UI that contains both LTR and RTL FUI controls (Figure 1 gives an example: the e-mail text box must be LTR in both versions).

In the next section, we will provide enhancements to UsiXML to overcome the shortage in the current version.

UPDATING UsiXML DESIGN TO SUPPORT RTL UI

UsiXML has a unique characteristic over other UI models, which is the employment of multiple models in one. UsiXML designers design the UI of a concept, while in other UI models, they focus on the design of a single interface (window). One of the important models in our case is the context model.

Updating UsiXML class design

RTL is a platform property, and the context model describes the platform. This direct mapping gives an intuitive solution to the problem: extend the Platform class to contain a new property: "dir". The "dir" is an optional property that can have one of two values: "LTR" (the default) or "RTL". The other update to the design is adding a new property to the CUI class: 2DgraphicalCio. This will support RTL in all graphical user interface elements (containers or individual components). The "dir" property for the 2DgraphicCIO is inherited from the containing container. Root container (the window) inherits from the platform "dir" property.

Using the "dir" property

From a designer perspective:

The designer creates a new context for the RTL UI, where he can localize the messages and other resources. He also

sets the “dir” property of the platform to the value “RTL”. There is no need to set the “dir” property for containers and elements as the default value for them will be that in the context. Wherever there is an exception (i.e: the GUI is orientation-independent), a fixed value needs to be set for that gui element’s “dir” property. In Figure 1, we can see an example: the text box of the e-mail; e-mails are always written in English, so the text box should always be LTR.

From a transformer perspective

The transformer checks the “dir” property for each 2d graphical CIO. As we discussed before, the “dir” property is either context-dependent or static value. In both cases, it is resolved according to current employed context. Transformers have the choice now to render FUI correctly and to use the correct individual controls that support the CIO orientation.

Optional design modifications

In the section before, we mentioned the issue of non-horizontally symmetrical images. Although this issue doesn’t affect support for RTL in UsiXML (as support is provided using resource files), but an enhancement to reduce the number of images in the RTL UI localized resource may save designer’s time. Image rotation can be achieved programmatically. Transformers can produce the RTL version of these images when applied. All we need to do is to denote the images that are not horizontally symmetrical. We suggest adding a new property to the ImageComponent class; the “ImageDir” property. This property accepts one of the values: “RTL”, “LTR”, “empty”. If the physical image is for the LTR UI, then the value is LTR and vice versa. The transformer decision to rotate the image is explained in Table 2.

		ImageDir		
		LTR	RTL	empty
dir	LTR		Rotate	
	RTL	Rotate		

Table 2. The decision table used by transformers to rotate the image is based on the two properties of the Image.

The rotated version of the image can be saved and handled by the transformer internally and the transformation process is re-entrant. Direction switching can be handled by adding a direction tag separator when a switching occurs. We can use two tags: <LTR> and <RTL>. A string like:

3 2 1
 تقابلنا في | Paris | السنة الماضية

Can be rewritten to represent direction switching as follows:

<RTL>تقابلنا في</RTL><LTR>Paris</LTR><RTL>السنة الماضية</RTL>

This way, a transformer can have a choice to handle cases where an FUI control doesn’t support RTL to be rendered correctly. A valid approach to render the above text appropriately is to generate 3 labels in that the left-most will contain the 3rd part of the string and so on. In the case where the OS doesn’t support the alphabet, a transformer can produce an image of the above text. If RTL is well supported in a platform, the transformer can ignore the direction tags.

RTL AND USABILITY

Many resources can be found regarding usability tips and practices. In this paper, we present some of them and show the effect of RTL on known usability (RTL usability) tips and practices. We do not intend to provide a full guide, but just to give a sense of the differences. Some useful usability tips exist : <http://www.ambyssoft.com/essays/user-InterfaceDesign.html>. We excerpt two of them to discuss.

Tip 1: Align fields effectively: When a screen has more than one editing field, you want to organize the fields in a way that is both visually appealing and efficient. I have always found the best way to do so is to left-justify edit fields: in other words, make the left-hand side of each edit field line up in a straight line, one over the other. The corresponding labels should be right-justified and placed immediately beside the field. This is a clean and efficient way to organize the fields on a screen.

Our comment: This is an LTR thinking. To make this tip applicable in RTL UI, we need to replace the left words with right and vice versa.

Tip 2: Justify data appropriately: For columns of data, common practice is to right-justify integers, decimal align floating-point numbers, and to left-justify strings.

Our comment: this is a cultural preference. In Arabic, the more preferred way is to center-align numbers and right-justify strings.

TOWARDS CULTURALLY-AWARE UsiXML UIs

The last examples suggest that transforming UsiXML-based UIs for Arabic languages and culture is certainly a matter of revisiting existing techniques for selecting and placing widgets in a GUI. In this section, we briefly consider two techniques used for automating the production of GUIs: one technique that automatically select widgets depending on domain parameters and one technique for automated layout of these widgets.

Automatic selection of widgets

[14] provides a decision-tree based technique in order to automatically select widgets (e.g., check boxes, radio buttons, list boxes, combination boxes) based on parameters coming from the domain model (e.g., data type, number of possible values, number of values to choose). While the cognitive principles and usability guidelines that have been considered in order to build this technique remain universal in principle, there are still some adjustments that are not considered for Arabian languages because the technique assumes that the GUI is based on Western properties that are not necessarily applicable or valid for non-Western countries and cultures. For instance, Table 3 depicts some

adjustments that have been introduced in order to take into account RTL languages: the rightmost columns indicates the Concrete Interaction Object (CIO) to be selected depending on the following parameters from the domain model: for a simple choice in a known domain, number of secondary values (Nsv), expandable domain (Exp), Continuous domain (Cont), Number of possible values (Npo), precision, and orientation. Most of the time, the CIOs remain the same, but their right alignment is preferred and the combination of widgets is arranged so that the reading order is correct.

Automatic laying out of widgets

[15] describes an algorithm that automatically positions selected widgets resulting from the previous step in a container according to a technique called "Right-Bottom". This dynamic strategy is explained to be more flexible than a static strategy [16]. Let S_i denotes the CIO placed at time i . S_{i+1} is the next CIO to be placed. The idea of the right/bottom strategy consists of following the visual continuity principle by either placing S_{i+1} on the right of S_i or beneath S_i . The idea is to perpetuate this layout technique until all selected widgets have been placed. The placement strategy is defined as the following:

if the total length does not exceed the limit
then
 place S_{i+1} with horizontal sequencing
 three cases are to be considered
 1. height (S_i) = height (S_{i+1})
 apply proportional uniformity
 2. height (S_i) > height (S_{i+1})
 if S_{i+1} = edit box
 then
 if S_i = list box or edit box
 then apply bottom justification
 else apply upper justification
 3. height (S_i) < height (S_{i+1})
 if available space is sufficient
 then apply bottom justification
 else maximize upper justification
else
 place S_{i+1} with vertical sequencing.

Nsv	Exp	Cont	Npo	Precision	Orientation	CIO
> 0						list box with right aligned items
= 0	yes					combination box with right aligned edit field
	no	no	[2,3]			radio-button with Npo items
			[4,7]			radio-button with Npo items + group box
			[8,Tm]			list box with right aligned items
			[Tm+1,2Tm]			scrolling list box with right aligned items
			> 2Tm			scrolling drop-down list box with right aligned boxes
	yes		[1,10]	low	vertical	scroll bar
					horizontal	Scale
					circular	pie diagram in counterclockwise presentation
					undefined	scale
				high	vertical	vertical thermometer
					horizontal	horizontal thermometer
					circular	dial with Arabian presentation
					undefined	horizontal thermometer
			[11,Tm]	high		spin button
				low		scale
			> Tm	high		spin button
				low	vertical	scroll bar
					horizontal	scale
					circular	dial with Arabian presentation
					undefined	scale

Table 3. Rules for selecting Concrete Interaction Objects (Adapted from [14]).

If we want to adapt this technique, we must revert most of mathematical relationships that are logically defined. This then becomes the Left-Bottom strategy:

if the total length does not exceed the limit
then
 place S_{i+1} with horizontal sequencing from left to right
 three cases are to be considered
 1. height (S_i) = height (S_{i+1})
 apply proportional uniformity
 2. height (S_i) > height (S_{i+1})
if S_{i+1} = edit box

then
if S_i = list box or edit box
then apply upper justification
else apply bottom justification
 3. height (S_i) < height (S_{i+1})
if available space is sufficient
then apply upper justification
else maximize bottom justification
else
 place S_{i+1} with vertical sequencing.

	PDI		IDV		MAS		UAI		LTO	
	rank	score	rank	score	rank	score	rank	score	rank	score
Arab Countries	7	80	26/27	38	23	53	27	68		
Argentina	35/36	49	22/23	46	20/21	56	10/15	86		
Australia	41	36	2	90	16	61	37	51	15	31
Austria	53	11	18	55	2	79	24/25	70		
Bangladesh									11	40
Belgium	20	65	8	75	22	54	5/6	94		

Table 4. Some values of the Five Cultural Dimensions as estimated in [10].

At first glance, at are these little adjustments may appear sufficient. But at second glance, updating a GUI for Arabian languages is more than a simple matter of repositioning widgets. Indeed, culture has been identified [7,8,9,13] has a very important aspects that is often forgotten in the design of globalized user interfaces and localized user interfaces, thus clearly affecting the identity of the web site in the country [12]. Marcus [10,11] classified 53 countries according to Hofstede [8] five cultural dimensions: PDI: Power distance index, IDV: Individualism index, MAS: Masculinity index, UAI: Uncertainty avoidance index, and LTO: Long-term orientation index. For instance, Table 4 reproduces some lines of this work (Source: <http://www.amanda.com/resources/hfweb2000/hfweb00.marcus.html>).

While some of the indexes are mainly indicative and not necessarily true in every circumstance, they suggest aspects to be considered in culturally-aware UIs. For instance, figure 6 depicts the same web site but with complete redrawing while keeping the same contents. While consistency across languages may certainly be desirable, other aspects could be considered as well. For instance,

Marcus suggests that the following countries have very different masculinity index: 95 Japan, 79 Austria, 63 South Africa, 62 USA, 53 Arab countries, 47 Israel, 43 France, 39 South Korea, 05 Sweden. This is not at all reflected in the GUI design.





Figure 6. The same web site in English (LTR) and in Arabic (RTL).

A CLASSIFICATION OF ISSUES FOR ARABIAN UIS

In order to classify a manual operation and, therefore, a subsequent beautification operation, Nielsen's linguistic model of interaction [14] was selected for these reasons: it decomposes a human-computer interaction in terms of seven inter-related, but independent, levels with a communication protocol between them; it has already been successfully used to classify usability guidelines according to their level of importance; and it allows identification criteria to univocally locate each modification to one and only one level. Table decomposes a simple goal (i.e., delete a paragraph in a letter) into subsequent units of interaction for each level:

- **Level 1 (Goal):** expresses a user's mental goal, such as "search for a particular customer having a water meter in a specific region".
- **Level 2 (Pragmatic):** translates this mental goal into a task to be carried out in the system according to the system concepts, such as "search for a subscriber having at least one water meter in zone x".
- **Level 3 (Semantic):** translates the real-world objects into system objects and functions, such as "search for a subscriber with a code region filled in".
- **Level 4 (Syntactic):** structures the semantic into an ordered sequence of operations in time and space, such as "select a zone code from the list and launch a query".
- **Level 5 (Lexical):** decomposes each operation into the smallest possible pieces of information, such as "a zone code".
- **Level 6 (Alphabetic):** specifies the unit of information (e.g., a lexeme, a metric) for each information item, such as "an integer for representing the zone code".
- **Level 7 (Physical):** specifies the physically-coded information in terms of light, sound, color, etc., such as "display the integer in black on white for input".

L e v e l	Title	Units	Definition	Example	W o r l d
1	Goal	Concepts of real world	Mentalization of a goal, a wish in the user's head	Delete a paragraph from my letter	Conceptual
2	Pragmatic	Concepts of system	Translation of a goal into system concepts	Delete 6 lines of the current paragraph in the edited text	
3	Semantic	Detailed functions	Real world objects translated into system objects manipulated by functions	Delete a certain amount of lines	
4	Syntactic	System sentences	Time & space sequencing of information units	DELETE 6	Perceptual
5	Lexical	Information units	Smallest elements transporting significant information: word, figure, screen coordinates, icon	[DELETE] command, [6] number	
6	Alphabetic	Lexems	Primitive symbols: letter, numbers, columns, lines, dots, phonemes, ...	D, E, L, E, T, E, 6	Physical
7	Physical	Physically coded information	Light, sound, physical moving	Pressing [CTRL]+[D] followed by [6]	

Table 5. Definition of the seven levels of Nielsen's linguistic model of interaction [14].

According to Nielsen's linguistic model, we can classify the problems discussed in this paper as follows:

Text Localization is the replacement of strings with the Arabic opponents; it replaces primitive symbols, so it fits at level 6 (alphabet). Writing direction switching involves syntactical handing of the text, so it fits at level 4.

Image Localization is just like Text Localization if we see localized images as the path to the image (the unit of information is the image path). If we look at images as the unit of information (like when generating images dynamically), then Image Localization is considered at level 5 (Lexical).

Control Localization fits at level 5 (lexical) or at level 4 (Syntactic) according to the way we implement it. We can use "RTLLabel" and "LTRLabel" instead of "Label" to implement localized controls, then we are adding new lexemes and localization fits at level 5. If no new lexemes are introduced, we can still support localization by introducing a new syntax: *Label rtl*, "any Arabic string" and the Control localization fits at level 4. We believe that UsiXML should implement control localization at level 4.

The Orientation (RTL layout) is at level 4 since it is related to spatial aspects, but it can be seen at level 3 (semantic). At level 4, the UI will be described as:

```
Add Window RTL
Add Label RTL, "An Arabic string"
Add TextBox RTL
...
Add Label LTR, "E-Mail"
Add TextBox LTR
```

While at level 3, we can describe the same UI above as:

```
UIContext RTL
Add Window
Add Label "An Arabic string"
Add TextBox
...
Add Label LTR, "E-Mail"
Add TextBox LTR
```

In this example, the page direction is defined globally and controls inherit the orientation property. The direction can be still set manually for a specific control if different than the global setting. The algorithms we discussed before affect widget selection and layout, thus they fit at level 4. We note also that a little semantic is used which is related to the orientation problem.

The cultural differences and effect on Arabian UI (the Hofstede's cultural dimensions) are at level 2 (Pragmatic) since taking into account cultural aspects is very much based on high-level considerations.

CONCLUSION

In this paper, we explained the challenges imposed by RTL languages. We provided a solution in UsiXML to handle all the challenges we mentioned. The proposed solution provides better support for designers than other UI models thanks to unique characteristics of UsiXML. The "dir" property of the platform is used to handle the orientation problem and the "dir" property of the ui elements provides support for localized controls. Direction switching can be fixed using the direction tag separators. Image localization for non-horizontally symmetrical ones is also handled. On top of all that, the designer maintains only one copy of the design thanks to the context model in UsiXML.

FUTURE WORK

The Orientation problem can be generalized to address Top To Bottom UI and Bottom To Top UI. Studying the effect of RTL on 3DUI can be another interesting area to evaluate. In the context of cultural differences effect on UI, more work should be spent on Arabian UI to come up with concrete results that may help to enhance Arabian usability guidelines.

ACKNOWLEDGEMENTS

The authors would like to acknowledge the FP7 Serenoa project funded by the European Commission and the UsiXML project (ITEA2 Call3) under reference 2008026.

REFERENCES

1. Bodart, F., Hennebert, A.-M., Leheureux, J.-M., and Vanderdonckt, J. Towards a Dynamic Strategy for Computer-Aided Visual Placement. In *Proc. of 2nd ACM Workshop on Advanced Visual Interfaces AVI'94* (Bari, 1-4 June 1994). T. Catarci, M.F. Costabile, S. Levialdi, G. Santucci (eds.), ACM Press, New York, 1994, pp. 78-87.
2. Bodart, F., Vanderdonckt, J. On the Problem of Selecting Interaction Objects. In *Proc. of BCS Conf. HCI'94 "People and Computers IX"* (Glasgow, 23-26 August 1994), Cambridge University Press, Cambridge, 1994, pp. 163-178.
3. Coyette, A. and Vanderdonckt, J. A Sketching Tool for Designing Anyuser, Anyplatform, Anywhere User Interfaces. In *Proc. of 10th IFIP TC 13 Int. Conf. on Human-Computer Interaction INTERACT'2005* (Rome, 12-16 September 2005). Lecture Notes in Computer Science, Vol. 3585, Springer-Verlag, Berlin, 2005, pp. 550-564.
4. Cyr, D., Head, M., and Larios, H. Colour appeal in website design within and across cultures: A multi-method evaluation. *International Journal of Human-Computer Studies*, Vol. 68, No. 1-2, January 2010, pp.1-21.
5. DelGaldo, E.M. and Nielsen, J. *International User Interfaces*. John Wiley & Sons, New York, 1996.
6. Evers, V. and Day, D. The role of culture in interface acceptance. In *Proc. of the IFIP TC13 Int. Conf. on Human-Computer Interaction INTERACT'97* (Sydney, 14- 18 July 1997), Chapman & Hall, London, 1997, pp. 260-267.
7. Hau, E. and Aparicio, M. Software Internationalization and Localization in Web Based ERP. In *Proc. of the 26th Annual ACM Int. Conf. on Design of communication SIGDOC'2008* (Lisbon, 22-24 September 2008). ACM Press, New York, 2008, pp. 175-180.
8. Hertzum, M., Clemmensen, T., Hornbæk, K., Kumar, J., Shi, Q., and Yammiyavar, P. Usability constructs: a cross-cultural study of how users and developers experience their use of information systems. In *Proc. of the 2nd Int. Conf. on Usability and Internationalization* (Beijing, 22-27 July 2007). Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2007, pp. 317-326.
9. Hofstede, G. *Culture and organisation: Software of the Mind*. McGraw Hill, New York, 1997.
10. Hofstede, G. *Culture's Consequences: International Differences in Work-Related Values*. Sage Publications, Beverly Hills, 1980.

11. Mahemoff, M. and Johnston, L. "Culturally-aware" requirements for internationalized software. In *Proc. of 3rd Australian Conf. on Requirements Engineering ACRE'98* (Geelong, Australia, 26-27 October 1998), D. Fowler, L. Dawson (eds.), Deakin University, Geelong 1998, pp. 83-90. Accessible at <http://mahemoff.com/paper/reqsil8n/>
12. Marcus, A. and West Gould, E. Cultural Dimensions and Global Web User-Interface Design. *Interactions*, Vol. 7, No. 4, July 2000, pp. 32-46.
13. Marcus, A., Armitage, J., Frank, V., and Guttman, E. Globalization of User-Interface Design for the web. In *Proc. of 5th Int. Conf. on Human Factors and the Web HFWeb'99* (Gaithersburg, 3 July 1999), 1999. Accessible at <http://www.amanda.com/resources/HFWEB-99/HFWEB99.Marcus.html>.
14. Nielsen, J. A Virtual Protocol Model for Computer-Human Interaction. *International Journal of Man-Machine Studies*, Vol. 24, No. 3, 1986, pp. 301-312.
15. Nomura, S., Ishida, T., Masaki, S., Yokozawa, M., and Shinohara, T. International Comparative Study of Identity as Presented on the Internet. In *Proc. of 6th Conf. on Human Factors and the Web HFWeb'2000* (Austin, 19 June 2000), Ph. Kortum, E. Kudzinger (eds.), University of Texas, Austin, 2000.
16. Patrick Rau, P.-L., Gao, Q., and Max Liang, S.-F. Good computing systems for everyone-how on earth? Cultural aspects. *Behaviour & Information Technology*, Vol. 27, No.4, July 2008, pp. 287-292.
17. Pérez-Quinones, M.A., Padilla-Falto, O.I., and McDevitt, K. Automatic language translation for user interfaces. In *Proc. of the Int. Conf. on Diversity in computing TAPIA'2005* (Albuquerque, 19-22 October 2005). ACM Press, New York, 2005, pp. 60-63.
18. Portaneri, F. and Amara, F. Arabization of Graphical User Interfaces. In "International User Interfaces", E. Del Galdo, J. Nielsen (eds.), John Wiley & Sons, New York, 1996. <http://www.langbox.com/staff/arastub-.html>.
19. Rehm, M., Bee, N., Endrass, B., Wissner, M., and André, E. Too close for comfort?: adapting to the user's cultural background. In *Proc. of the Int. Workshop on Human-centered multimedia HCP'07* (Augsburg, 28-28 Sept. 2007). ACM Press, New York, 2007, pp. 85-94.
20. Rejmer, P., Cooper, M., and Vanderdonckt, J. Lessons Learned From Internationalizing a Web Site Accessibility Evaluator. In *Proc. of 3rd Int. Workshop on Internationalisation of Products and Systems IWIPS'2001 "Designing for global markets 3"* (Milton Keynes, 11-14 July 2001), D.L Day & L.M. Dunckley (eds.). Digital Printing Services, The Open University, Milton Keynes, 2001, pp. 61-79.
21. Taylor, D. Global software: Developing applications for the international market. Springer-Verlag, Berlin, 1992.
22. Trompenaars, F. and Hampden-Turner, Ch. Riding the waves of culture, McGraw-Hill, New York, 1998.
23. UsiXML User Interface eXtensible Markup Language :<http://www.w3.org/2005/Incubator/model-based-ui/wiki/UsiXML>
24. Vanderdonckt, J. Knowledge-Based Systems for Automated User Interface Generation: the Trident Experience. In *Proc. of CHI'95 Workshop on Knowledge-Based Support for the User Interface Design Process* (Denver, 7-8 May 1995). ACM Press, New York, 1995, pp. 21-33.
25. Vanderdonckt, J., Ouedraogo, M., and Yguetengar, B. A Comparison of Placement Strategies for Effective Visual Design. In *Proc. of BCS Conf. HCI'94 "People and Computers IX"* (Glasgow, 23-26 August 1994). G. Cockton, S.W. Draper, G.R.S. Weir (eds.), Cambridge University Press, Cambridge, 1994, pp. 125-143.
26. Wikipedia, *List of User interface Markup languages*. 2010. Accessible at http://en.wikipedia.org/wiki/List_of_user_interface_markup_languages
27. XUL, accessible <https://developer.mozilla.org/En/XUL>
28. Zahir, S., Dobin, B., and Gordon Hunter, M. Analysis of the cross-cultural dimensions of national web portals, Managing globally with information technology. Idea Group Publishing, Hershey, 2003

A Comparative Study of Task Concepts in UsiXML and CTML and their Applications

Maik Wurdel

Department of Computer Science
Albert-Einstein Str. 21
18055 Rostock, Germany
+49 381 498 7443
maik.wurdel@uni-rostock.de

Peter Forbrig

Department of Computer Science
Albert-Einstein Str. 21
18055 Rostock, Germany
+49 381 498 7620
peter.forbrig@uni-rostock.de

ABSTRACT

In this paper we compare the task concepts of two modeling languages, namely UsiXML and CTML. We begin with a classification of application types which can be created via a model-based UI processes. Then we evaluate not only the pure task concepts but also the embedding of task modeling in the process lifecycle. One of the results of the study is that the task concepts are actually similar but UsiXML and CTML differ in the high level modeling concepts which in turn use task modeling. In order to compare the languages also the application domains for each are examined.

Keywords

Task modeling, UsiXML, CTML, model-based development

INTRODUCTION AND BACKGROUND INFORMATION

The development of interactive applications is becoming more and more complex for several reasons. First user interfaces (UI) need to be more appealing in terms of functionality and usability as users employ systems more consciously nowadays. This applies for all types of applications even though this issue can be addressed for single platform applications, compared to the subsequent ones, easily.

However the diversity of platforms in use for the very same interactive system raises the need for different UIs for the diverse capabilities and limitation of each platform still providing a consistent look & feel. This issue is mainly addressed by Multiple User Interfaces (MUI) [46]. Certainly the development efforts for MUIs increase with the set of platforms. In order to develop an appropriate UI for a certain platform the context of use needs to be investigated.

Delivering the UI for different platforms is by far not sufficient as devices are mobile these days and rapid changes of context may occur. Therefore adaptation at runtime is consequently the next step and comprises new challenges. Context-awareness has been a research area since the 1990s (an overview of the roots can be found in [12]) and investigates adaptation mechanisms for software systems with respect to continuously changing context of use [44].

It has been constantly discussed what constitutes context, how it is formalized best and how an application should

make us of it. The range of context used in system design varies a lot. Exemplary the authors in [58] use power consumption of sensor nodes as context for routing whereas in [35] the physiological state of the user is considered as context of use. Those examples already indicate that there is not yet a common agreed set of features to be considered as context for a particular domain. From our point of view that is one of the reasons why it is still cumbersome to implement a context-aware application as methodical knowledge of engineering as well as implementation frameworks for context-aware applications are still not extensive enough.

Beyond that, there is also the research field of implicit interaction [45] in which interaction is taking place even the user is not aware of. Implicit interaction is rooted in context-aware computing and differs from explicit interaction (e.g. ordinary graphical user interface (GUI)) by the purpose of the action. When interacting explicitly an action is performed to interact with the system whereas in the other an action is interpreted by the system but has not been initiated by the user to interact (e.g. raising a hand).

Intelligent assistance [5], smart environment [11] and ambient intelligence [1] are currently vital research areas in which implicit interaction is under investigation. To implement implicit interaction the context of use and the domain the user is confronted with are enormously important as action may have a totally different semantics under a slightly different context or domain. Usually implicit interaction is accompanied, at least by a certain degree of, explicit interaction to synchronize the concurrently acting user and the executing system.

Smart environments are physical spaces being able to react and adapt to users' needs assisting the users in their objectives [27] and make their experience more comfortable [11]. Application areas are among others meeting and conference rooms, homes and teaching environments.

In [27], a comprehensive overview of the application domains in the field is given. However the definition above is rather fuzzy because it only characterizes smart environments as physical place assisting the users. How and to which degree user assistance is performed is not included in the definition.

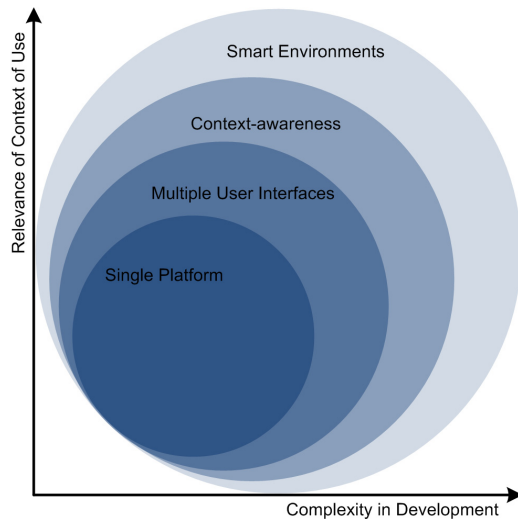


Figure 1. Complexity Chart for Application Types.

The interaction paradigm used for those environments differ vastly. Especially approaches in the HCI focus on explicit interaction in which UIs are generated dynamically. In [3] the authors propose an approach based on mobile devices whose UIs adapt according to the available services in the surroundings. Clerckx et. al follow a similar approach [10]. Other related approaches can be found in [14, 48]. In the research community of smart environments, implicit interaction is the prevalent paradigm to reduce explicit interaction to a minimum. Heider & Kirste [23] propose a layered model to achieve so called goal based interaction which is a special case of implicit interaction in which goals are expressed explicitly by formal statements. The layered model is based upon those goals to reduce complexity. On the highest level of abstraction the users' intention is analyzed which is directly mapped to a goal. Certainly a goal may be achieved by different actions sequences. The derivation and selection of action sequences is performed on the next level of abstraction: strategy planning. The resulting sequences of actions can be triggered to support the users' intention. With regard to task modeling intention analysis is particular interesting as task models are able to specify the potential tasks to achieve a goal. One can consider the task model as a pattern of actions indicating the currently executing task which helps to infer the current intention. Preliminary results on this topic have already been published by Giersich et. al [20]. Extensions to this approach have been proposed in [6, 53].

Please note that smart environment imply not only challenges by means of interaction but also research in middleware, networking, sensing and service composition just to name a few [11] yet this is not the scope of the paper. The interest reader may consult [11, 27].

Dimensions of Complexity

A classification of the prior named types of applications can be found in Figure . The chart shows the relation of the importance of context and the complexity of development for each type of application. The color denotes the amount

of explicit interaction. The brighter the color the less interaction is necessary to use the application in an ideal case. This implies that even single platform applications and MUI already support implicit interaction to a certain degree. This is actually the case as background processes may be started and assumptions about the users' habits are encoded in the system [26]. The increasing complexity in development from single platform application to MUI is due to the diverse platforms in use in MUI applications. The relevance of context increases because an application in MUI may be used in different contexts; one for each platform in the simplest case. The shift from MUI to context-awareness is explained by the use of any kind of sensor to adapt the system accordingly. This applies for both axes as the sensors contribute to the context as well as to the complexity in development. Moving to smart environment context becomes even more influencing as explicit interaction is minimized resulting in implicit interaction based upon context. Certainly incorporating more context information and using them to provide proactive assistance adds a vast amount of complexity in development. For both, context-awareness and smart environments, adaptability plays also a major role for the complexity in development as systems of those kinds should adapt to users' needs accordingly.

In Figure 1 only three dimensions (context of use, complexity in development and explicit interaction (by the color)) are depicted. Yet this chart can be easily extended to other dimensions:

- **Attentional Demand.** When interacting explicitly the users' attention is focused on the system. If systems disappear into the background and so do their interaction the attentional demand of the user is no longer bound to the system but to the actual goal [26] which is one of the major objectives of context-aware systems and smart environment. Attentional demand can therefore be seen as an indicator for the quality of a context-aware system, smart environment respectively.
- **Modality.** The usage of multiple modalities (gestures, voice, pointing) may result in an increases relevance of context and definitively adds new complexity in terms of development. Semantic unification of the multimodal application is of enormous interest to ensure a seamless integration of the different modalities [36]. Please note that some modalities are only available in context-aware computing and smart environments as sensor may be needed.
- **Initiative.** Who takes the initiative is highly related to the interaction paradigm of a system [26]. If implicit interaction is prevalent the system may be more proactive then in explicit interaction. Proactiveness is a corner stone to implement smart environment but should be used with care as users are frustrated easily by erroneous proactive behavior of systems.
- **Automation.** Initiative and attentional demand can be combined to scales of automation. In [37] the authors investigate the ten scales of automation originally

published by Sheridan. The scales start with no automation offered by the system and gradually allocates tasks to the system until the whole process is automated.

Again, those dimension named above are only related to interaction. In fact, from system development viewpoint dimensions like system integration, network topology and others may be added as well.

Model-based Development

To cope with those dimensions methodologies need to be developed incorporating the specifics of each application type. It should not be the aim to comprise all types in a methodology as the requirements are too diverse. One approach which has been employed for each type of application individually is model-based development.

In particular model-based UI development is a major research area in HCI focusing on engineering the interaction for different types of applications. In recent years numerous methodologies accompanied by tools have been proposed to design the UI [16, 31, 32]. Naturally, they differ in level of abstraction, used notation, focus and procedure. However over the years consent about the general rationale of model-based UI process has emerged.

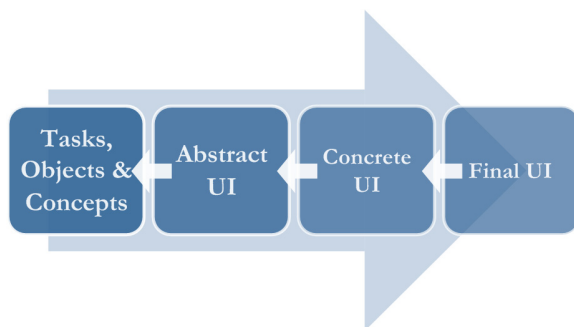


Figure 2. Basic Model-based UI Process.

Figure 2 shows the rationale of the basic model-based UI process. It mainly consists of four steps. Starting with task and domain analysis a conceptual model of the task world of the user is retrieved. After that the tasks are specified with regard to the envisioned software system under construction an abstract UI description is created. There are different approaches to do so. Dialog modeling is one technique to derive the first abstract canonical prototype. Tasks are grouped into dialogs and transitions of dialogs are defined by means of task execution [42]. A related approach has been published by Traetteberg [49] in which data flow of dialogs is emphasized.

In the next phase abstract UI components may be replaced by concrete ones. For MUI an abstract UI is used to derive a concrete UI for each platform. So, abstract UIs are platform independent whereas concrete UIs are not. In the last step the final UI is generated into a specific technology. Design adaptations might be needed to fine-tune the final UI (beautification). For the last two phases XML-compliant languages such as UIML or XUL are used. Languages like UsiXML or XIIML try to support the whole process.

Model-based UI design is not a straightforward process which is indicated by the arrows from right to left in Figure . It is an iterative design process driven by evaluation and feedback. This can be achieved by prototyping on each level of abstraction within the development lifecycle.

There are also research activities in the field of intention recognition for implicit interaction employing model-based development. These are by far not mature as model-based UI development but may benefit from it. The richness of model-based UI development originates from a, more or less, common agreed on development methodology. There is a set of models, accompanied by tools, facilitating the use of models and methodologies giving stepwise advice how to create an appropriate UI. Such a process would be worthwhile for intention recognition as well. Beyond, the notion of task and task modeling is also of interest for implicit interaction as knowledge of tasks can help to infer the goal of interest of the user. An adoption of concepts like the process chain of modeling concepts seems to be fruitful especially on the level of task modeling.

Task Modeling

In HCI, it is commonly agreed upon that task modeling is a good starting point for model-based UI design [15]. Most currently existing methodologies for model-based UI design consider the task model as ground to start from. Software enables the user to achieve a goal by the execution of tasks. Those tasks need to be presented in a UI which shows the relation of a UI and a corresponding task model. Task-based approaches argue that the closer the UI corresponds to naturally way a goal is achieved the better the user will perceive the UI. To specify an adequate task model task analysis is performed which is an effective requirements elicitation device as it helps to gain understanding of how people currently work. According to Johnson, *“the role for the task analysis is to provide an idealized, normative model”* of the tasks users carry out to achieve goals [25]. However, for model-based design an idealized image of the work situation is often not sufficient. Because of that and other reasons several task modeling languages have been developed. They can be classified according to their origins [30, 40]:

Cognitive psychology. Task modeling is understood as vehicle to transmit knowledge about how users interact with machines or software. Task analysis identifies cognitive processes during interaction and task structures. In addition assessment of work load, performance, task allocation and usability can be performed. Representatives of task modeling languages of this type are HTA [2], TKS [25], GOMS [9], TAG [39], MAD* [43].

Software Engineering. Task models are used as specification mechanism in various stages of software engineering. Task analysis can be used in early stages to elicit requirements which in turn may be specified by means of task models [22, 42] as it the case in model-based UI development. As already hinted in the last sections, task models may also be used at runtime which requires a machine

readable format and precisely defined semantics especially for remote systems. Therefore some task modeling language (CTT [38], CTML [57], TaoSpec [13]) can be traced back to process algebras such as CSP [24]. In addition, domain specific extension may be introduced to incorporate the special constraints and concepts of the domain of interest [4, 20, 47, 54].

Ethnography. Ethnography studies, among others, the interacting of humans through empirical studies. Different methods are employed to gather the data such as interviews or observation. Task modeling can be another source of data as task models can be discussed with the humans under observation. GTA [51] is one representative of this type of task model.

Even though different notations for task models exist, they assume the following common tenet: tasks are performed to achieve a certain goal. Moreover, complex tasks are decomposed into more basic tasks until an atomic level, an action, has been reached. Temporal operators are used to restrict the task execution order (sequence, choice, etc.). Moreover, tasks are classified according to the responsibility of execution and abstraction level (e.g. user tasks, interaction tasks, etc). This rather simply meta-model has been widely adopted.

UsiXML

Addressing the challenges of model-based UI development for multiple context of use is the objective of UsiXML and its development methodology so called multi-path UI development [31]. It is based on the Cameleon Reference Framework [8] and allows for creating UIs by a set of declarative models and transformation steps. One of the assets of multi-path UI development, and thus also UsiXML, is that development can be started on any level of abstraction. Such an approach offers means for abstraction, reification (which are complementary) of models and translation of models to other context of use. UsiXML is fully MDA compliant by supporting three criteria: models, development methodology and tools. The tool support for UsiXML to support the multi-path development process is versatile [52].

Besides the classical models involved in model-based UI development UsiXML also incorporates models for the organizational structure, workflow modeling, modeling resources and process modeling. Moreover, not only entities are expressed by means of models but also transformations and mapping.

The application domain of UsiXML is precisely defined. With respect to Figure UsiXML serves for the first two application types. Context-awareness at runtime is not a feature inherently supported by UsiXML as different contexts are considered at runtime, but resulting in models for each context. Switching context at design time however implies the substitution of the whole model chain while the user is interacting with the system. Yet UsiXML allows for creating UIs for diverse modalities which is another type of dimension [19].

The task model itself is an extended version of CTT. It offers besides the standard operator set additional ones (distinction between non deterministic and deterministic choice, operators with information passing). Deterministic choice can be paraphrased as user choice whereas non-deterministic choice is a choice made by the system from the users' point of view [47]. The problem of operators with information passing is that it is not formalized. To our knowledge only Klug and Kangasharju assigned semantics to those operators based on ports [28]. Moreover in UsiXML additional attributes for tasks (e.g. frequency, userAction, importance) are introduced. Even though CTT supports the informal usage of preconditions no statements of this concept are made for UsiXML.

Task modeling in UsiXML is performed as specified in the Cameleon Reference Framework [8] which is the de facto standard for model-based UI development for multiple context of use. Basically this is an extension to the process depicted in Figure in which for each context of use such a process is run through. Additionally operations are offered to translate a model from one context of use to another (so called *translation* in [8]).

In IdealXML ([33]), the editor for task, domain and abstract UI models, abstract prototypes can be created which are based on user defined mappings of tasks, domain objects and abstract UI elements.

However, recently the task concepts and their tool support has been enhanced by FlowiXML [19]. In this work task modeling is studied in context of workflow modeling. More precisely extensions to task modeling have been introduced to combine high level workflow modeling and task modeling for workflow items as necessary. Workflows are modeled by means of adapted Petri-nets. Task modeling comes into play by further defining transitions using extended CTT models as described before. The rationale of this approach is based on the assumption that task models highlight the user' point of view on the system much better than a process model [18]. Unfortunately there are no statements how such a combined modeling approach of workflows and task models suits with the Cameleon Reference Framework and UsiXML in general.

Still this approach overcomes the limits of CTT task modeling to incorporate modeling of distributed actions of different users. In FlowiXML workflows are used to specify the casual dependencies of tasks of different users, in FlowiXML called *workers*, from different *organizational units*. This is definitely an enhancement of CTT and CCTT (Cooperative CTT) in terms of expressiveness. Moreover such an approach bridges the gap of two worlds: business process modeling and task modeling which have been mostly consider separately.

Beside the modeling facilities FlowiXML also offers modeling elicitation methods and tools. Task identification criteria are introduced in [17] and an appropriate tool is shipped with FlowiXML. A precise classification of tasks becomes indispensable especially if tasks are considered in

the context of workflow modeling as there are also processes and workflows involved. A distinction of tasks, processes and workflows is necessary to keep the level of abstraction consistent on each layer while modeling. Another elicitation tool offered by FlowiXML uses text-based scenarios to elicit modeling entities in the text.

As FlowiXML employs Petri-nets for workflow modeling and CTT as task modeling language semantics can be defined. However, to our knowledge such an interpreter (e.g. a workflow management system or task model interpreter) is not included in FlowiXML. This is actually unfortunate as animation has been proven as an excellent validation tool. Moreover, an interpreter may also be used at runtime to govern the information and control flow. Without an interpreter transformations are necessary to derive executable models. Without interpreter or transformations artifacts are bound to be requirements or documentations.

CTML

According to levels of complexity introduced in Figure CTML belongs to the category of modeling languages for smart environments. Therefore the complexity of the scenario needs to be supported. In terms of interaction design, explicit and implicit interaction need to be incorporated. Supporting task modeling for both does not introduce complexity in terms of expressiveness, but needs to allow for modeling on distinct levels of abstraction because *move from A to B* might be a relevant task for implicit interaction but results in an overspecified model for explicit interaction. Due to the option of gradual refinement in task modeling this issue can be solved even though two different specifications for implicit and explicit interaction may be needed in some cases.

CTML not only supports task modeling for smart environments, but has been designed to do so. Thus, location-awareness, object dependencies and device modeling are already considered in early stages of modeling.

The design of CTML is based on four fundamental assumptions:

1. In limited and well-defined domains the behavior of an actor can be approximated through her role.
2. The behavior of each role can be adequately expressed by an associated collaborative task expression.

3. The execution of tasks may depend on the current state of the environment (defined as the accumulation of the states of all available objects) and in turn may lead to a state modification.
4. The execution of task of individual users may contribute to a higher level team task.

Based on these assumptions we define a collaborative task model in a two-folded manner by the cooperation model and a set of configurations. Whereas the first holds information about the general structural and behavioral properties, the latter specifies runtime information (such as initial assignment) and animation/simulation configurations.

Cooperation Model

Figure shows a schematic sketch of a cooperation model. Elements in the inner bold oval show modeling entities of the cooperation model (post fixed with “-1”) whereas diagrams outside of it denote how specifications realizing the corresponding entities (post fixed with “-2”).

On a higher level of abstraction the cooperation model specifies the entities relevant to task modeling in smart environments. Therefore roles (e.g. A-1), devices (e.g. B-1), a location model (C-1), a domain model (D-1) and a team model (E-1) can be specified. Roles categorize users of the same kind in terms of capability, responsibility, experience and limitations according to the domain. In CTML the role concept is employed to define the pool of actions of a user by means of task expressions. In task analysis and modeling this approach is quite common but is usually restricted to a one-to-many relation of role and user ([34]). However, this is a rather rigorous constraint for smart environments as role switching occurs frequently (being a many to many relation of role and user).

The role concept implemented in CTML incorporates this case. Moreover a user cannot only perform roles concurrently but also other relations such as choice, orderindependence, enabling, disabling and suspend/resume are considered (adapted from temporal operators) which allows for modeling more realistic scenarios. In the example in Figure 3, a meeting scenario, the roles are *Presenter*, *Listener* and *Chairman*. They represent the different types of stereotypical behavior in the meeting scenario.

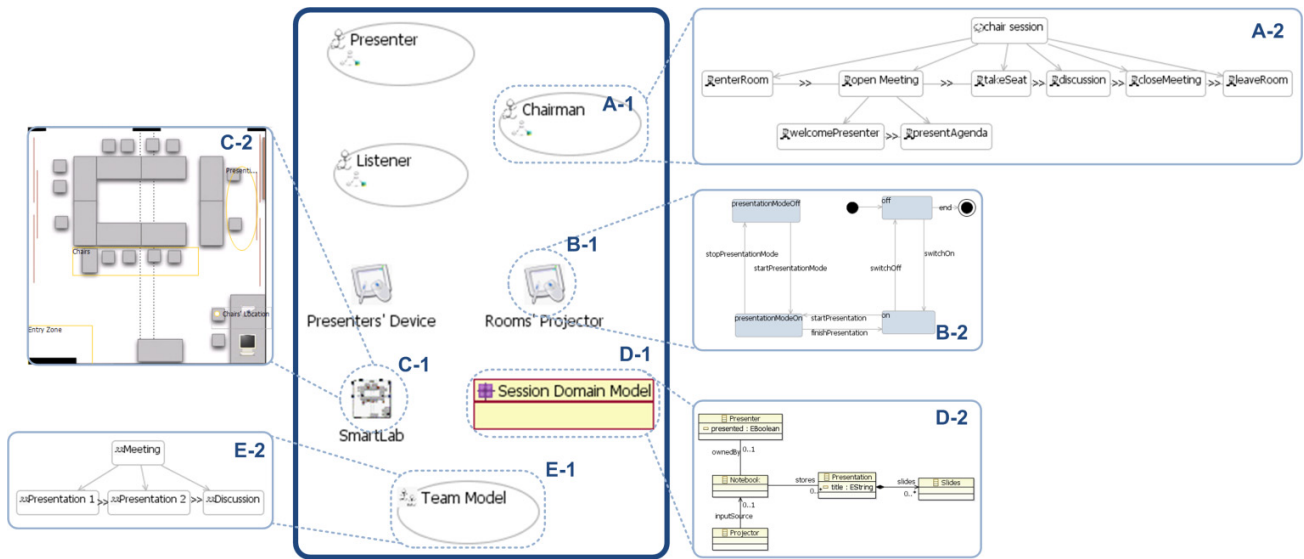


Figure 3. Schematic Cooperation Model for Meeting Scenario.

In CTML the potential action a user is able to perform is determined by her role(s). More precisely a role is associated with a collaborative task expression (A-2 in Figure 3), which has the form of a task tree in a CTT-like [38] notation (those concepts have been adopted). In addition, each task is attributed with a (unique) identifier, a set of precondition and a set of effects. Preconditions add additional execution constraints to a task as a task may only be performed if its precondition is satisfied. An effect denotes a state change of the system or environment as a result of a task execution. Both, preconditions and effects are needed to model collaboration and synchronization across collaborative task expressions. They also denote the binding to devices, location and domain.

CTML comprises the integration of different types of model: location, device and domain model. For reason of brevity only the location model with its binding to task modeling is investigated in detail. For the other models the procedures is analogue.

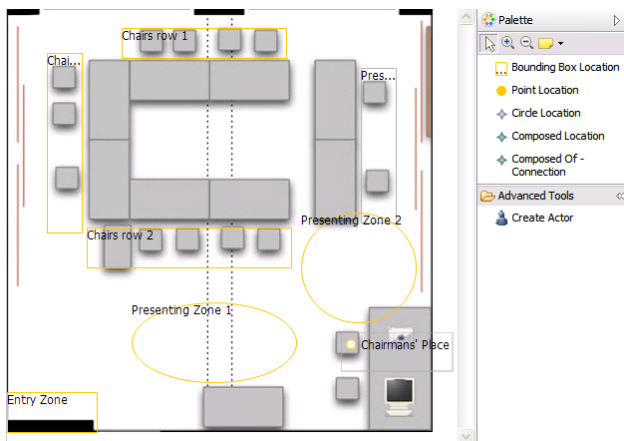


Figure 4. Location Modeling in CTML.

In Figure 4, the location model editor is shown. The software designer can create several types of locations. Be-

sides atomic locations (rectangles, ellipsis and x-y) there are also composed locations. To integrate the defined locations into CTML preconditions and effects are used. The designer may specify that a certain task is only enabled if the executing actor is located at the defined place. In the same vein effects are defined. One can specify that an actor changes its location through the execution of a task.

Configuration

Besides the cooperation model a CTML specification also contains one or more configurations providing essential runtime information for the cooperation model. A configuration represents necessary information for a concrete situation. This allows for testing different settings for the same cooperation model without much effort by defining different configurations. As the cooperation model relies on a role based specification actors operating in the environment need to be defined in accordance with a corresponding actor-role mapping (indicated by the directed arrows in Figure 5). Please note that an actor may fulfill more than one role as already explained earlier. Additionally an initial object model is needed to start an animation run of a CTML specification (as depicted in Figure 6).

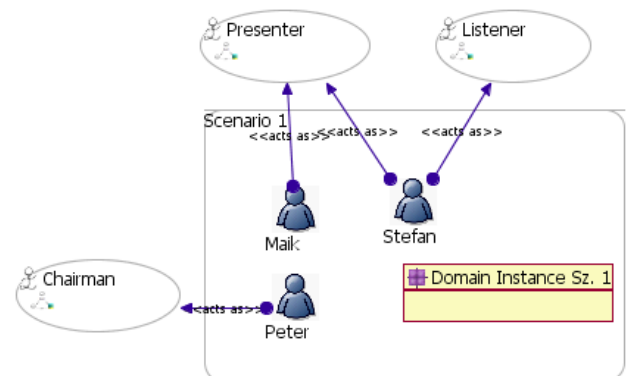


Figure 5. Configuration of Meeting Scenario.

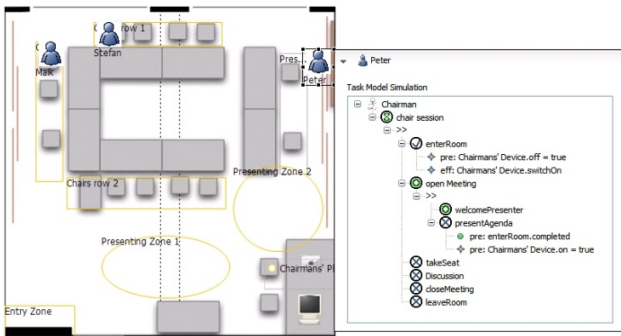


Figure 6. CTML Animation in 2-D View of the Smart Lab.

Development Methodology

The development of CTML specifications can be quite overwhelming due to the various types of modeling entities involved. Moreover a stepwise modeling is adjuvant not only to reduce complexity but to accompany each step of modeling with a thorough analysis. Therefore we propose a development methodology for implicit and explicit interaction on the level of task modeling as well as in prior modeling stages.

We consider an iterative, incremental development methodology as most suitable for interaction design which is substantiated by literature [21]. In order to obtain feedback in early stages iteration cycles are kept short. Prototypes can be created during each iteration on various levels of abstractions.

Incremental development denotes that software functionality is developed stepwise by increments. Each increment adds some tested, executable piece of software to the total system. This helps to keep track of the important requirements as well as to reduce project risks [29].

In [55] we proposed a development process for interaction design employing text-based scenarios, use cases and CTML specifications. It is tailored to smart environments but can also be generalized to meet the characteristics of model-based UI development. It inherently supports iterative development through validation by animation and refinement because model adaptation is a key principle and needs to be supported to enable iterative development. We understand validation as interactive exploration of CTML specifications being possible at any level of abstraction. Refinement is a verification technique which answers the question whether a certain specification T_1 truthfully implements a base specification T_0 . Therefore an algorithm has been developed which makes use of operators assigned to tasks, so called meta-operators, defining whether a certain task needs to be retained in subsequent development steps [56].

Comparison

In order to compare UsiXML and CTML certain properties of both languages need to be kept in mind: the focus or envisioned usage scenario and the application area. Due to that we start by first comparing those two proper-

ties of each language. Then we examine the task concepts more in detail. Finally the tool support is compared.

Focus

UsiXML and CTML have both been designed for model-based development of interactive systems. However, UsiXML is specialized to explicit interaction which is the interaction style in the first three categories in Figure (single platform, MUI and context-awareness). In the last category (smart environments) however not only explicit interaction is necessary but also implicit interaction.

UsiXML offers rich concepts and tools for explicit interaction supporting multiple modalities. Therefore it can be considered as a reference implementation of the Cameleon Framework. By embedding task models into workflow models task modeling is considered in a higher level context. It specifies the interaction to achieve high level workflow tasks. Still the application area stays the same: model-based UI development. The tool chain of UsiXML for model-based UI development can be applied to the task model specification within the workflow model.

In contrast, to our knowledge, CTML is the only approach trying to offer concepts for explicit and implicit interaction especially for smart environments. In order to do so, task modeling is not considered isolated but needs to be performed in combination with domain, location and device modeling. Based upon task descriptions a two-folded approach is followed. First, model-based UI development can be started which may incorporate location-awareness, object and device dependencies and cooperational aspects. Then, the next step in development is dialog modeling according to the modeling notation of Reichart [41, 42]. This approach covers explicit interaction in smart environments. Implicit interaction is supported by the construction of probabilistic models based on task models which are used in intention analysis. Basically intention analysis tries to derive users' intention by sensor data. However, knowledge-free approaches fail because of the diversity of potential actions available in smart environments. Thus intention analysis becomes only feasible by employing a priori knowledge about the relevant actions in a certain configuration of a smart environment. One option to deliver this knowledge is task modeling. However, an idealized model as proposed by Johnson is not appropriate in this case. Therefore diverse model dependencies need to be introduced as it has been done for CTML.

Task Concepts

The task modeling concepts implemented in UsiXML and CTML stick pretty much to CTT. Hierarchical decomposition and temporal operators are used. Additionally different task types exist. In UsiXML and CTML additional task types have been introduced. An extension to CTT represents semantically relevant preconditions and effects of tasks which however has been implemented also in other task modeling languages (see [7] for an overview).

The actual difference of task modeling is the context of modeling. In UsiXML tasks are specified either isolated as pure task tree to define mapping to other models or tasks are modeled in the context of workflow modeling. Task models embedded in workflows are considered as single user jobs. Collaboration needs to be modeled on higher level of abstraction, such as in the workflow or process model. Therefore the task model is only used to specify the interaction necessary to accomplish a workflow item by a single user which allows for separation of concerns.

In CTML task modeling is performed to specify the tasks in smart environment for the previously explained two-folded approach. As those specifications can become quite complex a formal foundation has been defined to enable the use of formal methods during model creation or adaptation. The formal framework uses similar but not identical concepts as CTT to define semantics.

Tool Support

The tool support for UsiXML is immense. The website currently lists more than 30 public available tools to facilitate the use of UsiXML [50]. In addition to editors, generators and interpreters also tools for model management, requirement elicitation, reverse engineering and usability engineering are available. With regard to task modeling one editor is used throughout all tools using task models in UsiXML. It is a lightweight version of CTTE which is the development environment of CTT. Several tools exist which makes use of this editor. In IdealXML an extended version is used which allows for interactive animation and calculation of the enabled task sets. The modeling concepts in IdealXML are identical with regard to the other tools in UsiXML. After designing the task model a domain model and abstract UI description can be defined. Next, mapping rules can be interactively created. A prototype of the envisioned UI can automatically be derived. Exemplary FlowiXML is an editor for workflow specifications employing the task modeling tool above. The same applies for other UsiXML tools.

The tool support for UsiXML can be quite overwhelming. Almost for each methodology and approach an own standalone tool exist. From our point of view consolidation is needed by integrating the tools into one environment.

CTML does not offer as many tools as UsiXML. This is due to manpower involved in the project and the focus of modeling. CTML focuses more on the pure task modeling and interfaces to lower level models whereas UsiXML tackles the whole model-based UI development life cycle. The tool support is integrated into the Eclipse environment and offers graphical editors for CTML specifications. Moreover, different animation modes have been developed to interactively explore the specification. In addition, verification algorithms have been implemented. As the dialog modeling module of Reichart is also integrated in Eclipse as well model-based UI development

can be continued seamlessly. Next, there is also a graphical editor for hidden Markov models (HMM) which is the preferred model for intention analysis in our smart lab. CTML to HMM transformations have also been developed to make effectively use of CTML for intention analysis. The results of the derived HMM are promising and we are planning to extend the approach.

DISCUSSION

Diverse task modeling notations exist. In this paper we investigated two representatives: UsiXML and CTML. We classified them according to different dimensions of complexity and examined their application domains. Moreover, we examined not only the task modeling concepts but also the methodology in which task modeling is employed to support software engineering.

One result of this comparison shows that the actual task concepts do not differ so much. Both are based on CTT but introduce new task types whereas CTML additionally incorporates semantically relevant preconditions and effects. The difference is more the higher level context task modeling is embedded in. IdealXML defines mappings of tasks to domain elements and abstract UI components to derive UIs. FlowiXML embeds task modeling into workflow modeling to provide a specification how a workflow item is achieved with respect to interaction. The usual modeling elements from workflow modeling are adopted in which also cooperative task performance is considered. CTML also tries to overcome the limitations of CCTT by introducing a cooperation model which provides the basis for modeling cooperative task performance. In CTML role-based behavior is explicitly specified by task descriptions which are combined by preconditions and effects. Moreover, other model types are considered as explained before.

The application areas of both languages are overlapping. UsiXML is dedicated to model-based UI development for the first three categories according to Figure whereas CTML is designed to meet the requirements of smart environment to specify models for both: implicit and explicit interaction. One way to implement explicit interaction is model-based UI development which is pursued by CTML.

For future research we feel that a rethinking of task modeling is necessary. An idealized model of task performance as Johnson stated is very helpful in early phases of development but with the advent of model-based UI development more expressive task specification approaches are necessary. FlowiXML and CTML are representatives of this new kind of techniques dedicated to specify real world tasks for design.

ACKNOWLEDGMENTS

Supported by a grant of the German National Research Foundation (DFG), Graduate School 1424, Multimodal Smart Appliance Ensembles for Mobile Applications (MuSAMA).

REFERENCES

1. Aarts, E., R. Harwig, and M. Schuurmans, *Ambient intelligence*, in *The invisible future: the seamless integration of technology into everyday life*. 2002, McGraw-Hill, Inc. p. 235-250.
2. Annett, J. and K.D. Duncan, *Task Analysis and Training Design*. Journal of Occupational Psychology, 1967. **41**: p. 211-221.
3. Blumendorf, M., S. Feuerstack, and S. Albayrak, *Multimodal user interfaces for smart environments: the multi-access service platform*, in *Proceedings of the working conference on Advanced visual interfaces*. 2008, ACM: Napoli, Italy.
4. Bomsdorf, B., *The WebTaskModel Approach to Web Process Modelling*. TaMoDia 2007, 2007. **4849**: p. 240-253.
5. Boy, G.A. and T.R. Gruber, *Intelligent Assistant Systems: Support for Integrated Human-Machine Systems*. 1990, Knowledge Systems, AI Laboratory, Stanford University.
6. Burghardt, C., S. Propp, T. Kirste, and P. Forbrig, *Rapid Prototyping and Evaluation of Intention Analysis for Smart Environments*. in *IMC*. 2009.
7. Caffiau, S., P. Girard, D. Scapin, L. Guittet, and L. Sanou, *Assessment of Object Use for Task Modeling*, in *Engineering Interactive Systems*. 2008. p. 14-28.
8. Calvary, G., J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon, and J. Vanderdonckt, *A Unifying Reference Framework for multi-target user interfaces*. Interacting with Computers, 2003. **15**(3): p. 289-308.
9. Card, S., T.P. Moran, and A. Newell, *The Psychology of Human Computer Interaction*. 1983, Hillsdale, N.J.: Erlbaum.
10. Clerckx, T., C. Vandervelpen, K. Luyten, and K. Coninx, *A task-driven user interface architecture for ambient intelligent environments*, in *Proceedings of the 11th IUI*. 2006, ACM: Sydney, Australia.
11. Cook, D. and S. Das, *Smart Environments: Technology, Protocols and Applications (Wiley Series on Parallel and Distributed Computing)*. 2004: Wiley-Interscience.
12. Dey, A.K. and G.D. Abowd. *Towards a Better Understanding of Context and Context-Awareness*. in *CHI 2000 Workshop on The What, Who, Where, When, Why and How of Context-Awareness*. 2000.
13. Dittmar, A. and P. Forbrig, *Higher-order Task Models*, in *DSV-IS*. 2003: Funchat, Portugal.
14. Duarte, C. and L. Carri, *A conceptual framework for developing adaptive multimodal applications*, in *Proceedings of the 11th IUI*. 2006, ACM: Sydney, Australia.
15. Forbrig, P., A. Dittmar, D. Reichart, and D. Sinnig, *User-Centered Design and Abstract Prototypes*, in *Proceedings of BIR 2003*, SHAKER, Editor. 2003: Berlin. p. 132 - 145.
16. Gaëlle, C., C. Joëlle, T. David, L. Quentin, S. Nathalie, B. Laurent, F. Murielle, and J. Vanderdonckt, *Plasticity of user interfaces: A revised reference framework*, in *TaMoDia*. 2002, Publishing House.
17. Garcia, J.G., Vanderdonckt, J., Lemaigre, Ch., Identification Criteria in Task Modeling, Proc. of 1st IFIP TC 13 Human-Computer Interaction Symposium HCIS'2008 (Milan, 8-9 September 2008), P. Forbrig, F. Paterno, A.M. Pejtersen (eds.), International Federation of Information Processing, Vol. 272, Springer, Boston, 2008, pp. 7-20.
18. Garcia, J.G., C. Lemaigre, J.M.G. Calleros, and J. Vanderdonckt, *Model-Driven Approach to Design User Interfaces for Workflow Information Systems*. Journal of Universal Computer Science, 2008. **14**(19): p. 3160--3173.
19. Garcia, J.G., J. Vanderdonckt, and J.M.G. Calleros, *FlowXML; a step towards designing workflow management systems*. Journal of Web Engineering, 2008. **4**(2): p. 163-182.
20. Giersich, M., P. Forbrig, G. Fuchs, T. Kirste, D. Reichart, and H. Schumann, *Towards an Integrated Approach for Task Modeling and Human Behavior Recognition*. Human-Computer Interaction, 2007. **4550**: p. 1109-1118.
21. Gulliksen, J., B. Göransson, I. Boivie, J. Persson, S. Blomkvist, and Å. Cajander, *Key Principles for User-Centred Systems Design*, in *Human-Centered Software Engineering — Integrating Usability in the Software Development Lifecycle*. 2005, Springer-Verlag Netherlands. p. 17-36.
22. Hackos, J. and J. Redish, *User and Task Analysis for Interface Design*. 1998: Wiley.
23. Heider, T. and T. Kirste. *Multimodal appliance co-operation based on explicit goals: concepts & potentials*. in *Proc. of the SOC-EUSAI 2005*. 2005. Grenoble, France: ACM.
24. Hoare, C.A.R., *Communicating sequential processes*. Commun. ACM, 1978. **21**(8): p. 666-677.
25. Johnson, P., *Human-computer interaction : psychology, task analysis and software engineering*. 1992, London u.a.: McGraw-Hill.
26. Ju, W. and L. Leifer, *The Design of Implicit Interactions: Making Interactive Systems Less Obnoxious*. Design Issues, 2008. **24**(3): p. 72-84.
27. Kirste, T., *Smart Environments*, in *True Visions*, E. Aarts and J.L. Encarnacao, Editors. 2006, Springer: Heidelberg. p. 323-339.
28. Klug, T. and J. Kangasharju, *Executable Task Models*, in *TaMoDia 2005*. 2005: Gdansk, Poland.
29. Larman, C., *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition)*. 2004: Prentice Hall PTR.
30. Limbourg, Q. and J. Vanderdonckt, *Comparing Task Models for User Interface Design*, in *The Handbook of Task Analysis for Human-Computer Interaction*, D. Diaper and N. Stanton, Editors. 2003, Lawrence Erlbaum Associates. p. 135-154.

31. 11. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Lopez, V. UsiXML: a Language Supporting Multi-Path Development of User Interfaces. In *Proc. of 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems EHCI-DSVIS'2004* (Hamburg, July 11-13, 2004). Lecture Notes in Computer Science, Vol. 3425, Springer-Verlag, Berlin, 2005, pp. 200-220.
32. Luyten, K., *Dynamic User Interfaces Generation for Mobile and Embedded Systems with Model-Based user Interface Development*. 2004, Universiteit Maastricht: Maastricht.
33. Montero, F. and V. López-Jaquero, *IdealXml: An Interaction Design Tool*, in *Computer-Aided Design Of User Interfaces V*. 2008. p. 245-252.
34. Mori, G., F. Paternò, and C. Santoro, *CTTE: Support for Developing and Analyzing Task Models for Interactive System Design*. IEEE Trans. Softw. Eng., 2002. **28**(8): p. 797-813.
35. Oliver, N. and F. Flores-Mangas, *MPTrain: a mobile, music and physiology-based personal trainer*, in *Proceedings of the 8th conference on Human-computer interaction with mobile devices and services*. 2006, ACM: Helsinki, Finland.
36. Oviatt, S., *Ten myths of multimodal interaction*. Commun. ACM, 1999. **42**(11): p. 74-81.
37. Parasuraman, R., T. Sheridan, and C. Wickens, *A model for types and levels of human interaction with automation*. Systems, Man and Cybernetics, Part A, IEEE Transactions on, 2000. **30**(3): p. 286-297.
38. Paternò, F., *Model-Based Design and Evaluation of Interactive Applications*. 1999, London, UK: Springer-Verlag.
39. Payne, S.J. and T.R.G. Green, *Task-Action Grammars: A Model of the Mental Representation of Task Languages*. Human-Computer Interaction, 1986. **2**(2): p. 93 - 133.
40. Pontico, F., C. Farenc, and M. Winckler. *Model-Based Support for Specifying eService eGovernment Applications*. in *TaMoDia*. 2007.
41. Reichart, D., A. Dittmar, P. Forbrig, and M. Wurdel. *Tool Support for Representing Task Models, Dialog Models and User-Interface Specifications*. in *DSV-IS*. 2008.
42. Reichart, D., P. Forbrig, and A. Dittmar, *Task models as basis for requirements engineering and software execution*. TAMODIA, 2004: p. 51-58.
43. Scapin, D. and C. Pierret-Goldbreich. *Towards a method for task description : MAD*. in *Proceedings of Work with Display Units (WWU '89)*. 1989: North-Holland: Elsevier Science.
44. Schilit, B., N. Adams, and R. Want. *Context-Aware Computing Applications*. in *IEEE Workshop on Mobile Computing Systems and Applications*. 1994. Santa Cruz, CA, US.
45. Schmidt, A., *Implicit Human Computer Interaction Through Context*. Personal and Ubiquitous Computing, 2000. **4**(2/3).
46. Seffah, A. and H. Javahery, *Multiple User Interfaces : Cross-Platform Applications and Context-Aware Interfaces*. 2004, Hoboken, NJ: J. Wiley.
47. Sinnig, D., M. Wurdel, P. Forbrig, P. Chalin, and F. Khendek, *Practical Extensions for Task Models*, in *TaMoDia 2007*. 2007, Springer. p. 42-55.
48. Sottet, J.-S., G. Calvary, and J.-M. Favre, *Models at Run-time for Sustaining User Interface Plasticity*, in *Models@Runtime workshop at Models 2008*. 2008. p. 1-10.
49. Trætteberg, H., *Integrating Dialog Modeling and Domain Modeling - the Case of Diamodl and the Eclipse Modeling Framework*. JUCS, 2008. **14**(Human-Computer Interaction).
50. UsiXML (2010). *USer Interface eXtensible Markup Language*. <Accessed: March 10, 2010> <Available from: <http://www.usiXML.org/>>.
51. van der Veer, G., B. Lenting, and B. Bergevoet, *GTA: Groupware Task Analysis - Modeling Complexity*. Acta Psychologica, 1996. **91**: p. 297-332.
52. Vanderdonckt, J. *Model-Driven Engineering of User Interfaces: Promises, Successes, and Failures*. In *Proc. of 5th Annual Romanian Conf. on Human-Computer Interaction ROCHI'2008* (Iasi, September 18-19, 2008), S. Buraga, I. Juvina (eds.). Matrix ROM, Bucurest, 2008, pp. 1-10.
53. Wurdel, M., C. Burghardt, and P. Forbrig, *Supporting Ambient Environments by Extended Task Models*. Proc. AMI'07 Workshop on MDSE for Aml Applications, 2007.
54. Wurdel, M., C. Burghardt, and P. Forbrig. *Making task modeling suitable for smart environments*. in *International Conference on Ultra Modern Telecommunications & Workshops, 2009. ICUMT '09*. 2009.
55. Wurdel, M. and P. Forbrig. *Use Cases and Task Models as Driving Forces to Identify Requirements in Smart Environments*. in *IMC*. 2009.
56. Wurdel, M., D. Sinnig, and P. Forbrig, *Task Model Refinement with Meta Operators*, in *DSV-IS 2007*. 2008: Kingston, Canada.
57. Wurdel, M., D. Sinnig, and P. Forbrig, *Towards a Formal Task-based Specification Framework for Collaborative Environments*, in *CADUI*. 2008: Albacete, Spain.
58. You, J., D. Lieckfeldt, F. Reichenbach, and D. Timmermann. *Context-aware geographic routing for sensor networks with routing holes*. in *Proceedings of the WCNC*. 2009. Budapest, Hungary: IEEE Press.

Generating User Interface for Information Applications from Task, Domain and User models with DB-USE

Vi Tran, Jean Vanderdonckt, Manuel Kolp, and Yves Wautelet

Louvain School of Management-PRISME

Université catholique de Louvain

Louvain-la-Neuve, Belgium

{Vi.Tran, Jean.Vanderdonckt, Manuel.Kolp, Yves.Wautelet}@uclouvain.be

ABSTRACT

Database Systems UI (User Interface) generation from declarative models has been the focus of numerous and various approaches in the human computer interaction community. Typically, the different approaches use the different models based on and exploiting their singular aspects. This paper proposes a new process that combines the task, domain, and user models taken together to drive the information system user interface design and code behind generation. To this end, we propose a framework, i.e., a methodological process, a meta-model and a software prototype called DB-USE. The main difference in our work from other ones is to contribute the mapping rules for creating the UI objects and to combine the three task, domain and user models to generate the code for performing both the UI and the generic functions of a database.

Keywords

Task Model, Domain Model, User Model, Automatic User Interface Generation

INTRODUCTION

Human-computer interactions are today managed through ergonomic and powerful database user interfaces (UI). These UI require to access and present data from different types of structures that reside typically in enterprise functional modules such as human resources, sales, marketing, accounting, finances, manufacturing or productions ERP packages [22]. Naturally database interfacing has been a technical and human interaction issue since the seventies [2, 3]. But, with the today's increasing use of very large databases and extended business applications such as ERP II, Customer Relationship Management, E-procurement, Reporting/Drill-Down or Data Mining systems, these user interfaces require dynamic automation and runtime generation to properly deal with on a large-scale.

UI researchers have richly discussed the capability and importance of automatic user interface generation and propose them as the core of visual-based development environments [4]. There are currently numerous and various approaches using different input materials: designs, patterns, architectures, declarative models, ... In this set of techniques, an emerging method is the automatic UI

generation from declarative models [5, 2, 6], inspired from Fourth Generation Languages code generation [5]. In practice, these models are high-level abstractions such as goal or task [7], presentation, dialogue [8] or interaction, domain [9] models on which we will focus in the research.

Typically, a strong motivation for database application UI generation is the unexploited potential of relationships between objects. This makes possible to create task-domain mapping rules that are in turn mapped onto containers in the UI model [21].

Advantages of Declarative Models in User Interface Generation

User interface is that subset of a software system that interacts with the user of the system. Since it is used to communicate between the end-users and the system's computer, making it a crucial element, the successfulness of a software system depends on, besides the design of its architecture, the user interface design [11]. End-users expect UIs to be easy to use, understand and give the most adequate result.

The development of successful interactive system requires a careful design of the user interface. The user interface affects the acceptance of the users to use this application or not. A good user interface design helps the users to work effectively and quickly. The first step is to take a step-wise approach by using a methodological UI derivation process based on the available resources such as task, domain, data, user, presentation, dialog models. This process is conducted by designing the user interface manually or semi-automatically or automatically. In this paper, we will approach the semi-automatic user interface generation.

Numerous research works in the area of automatic user interface generation [24] have focused mainly on UI generated from the declarative models. They present the following advantages [12] in regard to both the automatic user interface and code application generations:

- They can provide a more abstract description of the UI than the ones provided by the other UI development tools.

- They facilitate designing and implementing the UI in a systematic way since they offer capabilities such as modeling user interfaces using different levels of abstraction; incrementally refining the models; and re-using UI specifications.
- They provide the infrastructure required to automate tasks related to the UI design and implementation processes.

An important point in using declarative models for creating the user interface is that these models typically described at a conceptual level. So that, the user interface created based on these model can be implemented in the different ways on the different delivery platforms.

User Interface Generation from Combining Task, Domain and User models

The information used to build the user interface usually comes from a large and complex context in which the users work to complete their tasks. This context includes the users' characteristics (user model), their current domain of application (domain model often linked in the database world to the data or conceptual model), the tasks they commonly perform (task model), the platform they work on (platform model), the device they are currently using (device model) and so on. This context is not fixed for all the user interface generating processes since it depends on the different approaches. For example, in TRIDENT [7] the user interface is determined from the task and application models while in Mecano [9] the user interface is determined from the sole domain model.

The task, domain and user models are three important models based on which the user interface is easily built for the following reasons:

- *The task model describes the abstract user interface.* The task model expresses how a end user may want to interact with a system in order to reach a given goal. This expression is intended to be independent of any particular implementation or technology. This explains why a same set of models could initiate several different user interfaces [13].
- *The domain model provides the special features needed for creating a user interface and specifying the methods for performing the generic application functions.* These features are the attributes of the objects in domain model, the relationships between these objects and prototype of the generic application functions. The domain model is not used separate from other models to generate the user interface, it is combined to the task, presentation models [21].
- *The user model supports the creation of user interfaces which consider to the preferences of the users.* Like the domain model, the user model is not used separately from other models to generate the user interface since there are various aspects of the user interfaces that are adapted according to user models.

Hence, combining models is an important concept in user interface generation since the different models describe different aspects of the UI. For example, a user interface generated from a task model is expected to be a means by which the user can communicate with the system to accomplish his task. The user interface generated from a user model is expected to support the users based on their characteristics. The user interface generated from several different models carry many needed aspects of a user interface. Various research works have focused on such models to generate UIs. For instance, TOOD [14] uses the task and user models to generate the UI. MECANO [9] generates UI based on the domain model. TRIDENT [8] or FUSE [15] combines the task and domain models while SUPPLE combines the user and device models.

Unfortunately, the researchers who have focused on user interface generation have not investigated and studied the implementation and code generation of application tasks, especially, those generic one related to editing, inserting, deleting and searching data. This research focuses on the code application generation for the tasks of a data-oriented application coming from the following main points:

- The data manipulation demand is very high. It is repeated regularly in most database applications especially for common tasks such as editing, inserting, deleting and searching data.
- These functions are easily performed through receiving data from the user, displaying data to the user, executing the different SQL select, insert, update and delete queries with respect to the different user requirements.
- Typically, we aim at defining and using a methodological framework for developing a user interface to database application from the task model, domain model and user model combined together. This framework consists of:
 - A meta-model that governs the semantics of the task, domain and user models;
 - A methodological process that supports the UI designer/developer to create the UIs semi-automatically from task, domain and user models;
 - A software prototype supporting the methodological process in order to generate both the user interface code and the application code that performs the basic functions of the database application.

The paper is structured as follows. Section 2 overviews the related work in model-based UI generation. Section 3 described the models used in our approach. Section 4 is concerned with the description of our process and Section 5 describes DB-USE in terms of CASE-Tool technology. Finally, we conclude the paper.

MODEL-BASED USER INTERFACE DEVELOPMENT ENVIRONMENT: A SURVEY

To highlight the importance and need of declarative models in UI generation, we present some major processes of user interface generation based on these models. These processes differ from each other in terms of input information, mostly represented by declarative models and from which the generation is effectively done, the generation target, the generation process itself and the tools.

Most MB-UIDEs (Model-Based User Interface Development Environments) use the task and the domain model to specify the user interfaces such as Trident, Teallch, Goliath, Fuse such as shown in Table 1. If a model is used, it is marked with “+”; if not with “—”; “+/-” indicates that there is no concrete validation even if the tool is said to support the model.

MB-UIDE	Models				
	Task	Domain	Presentation	User	Dialogue
Trident	+	+	+	—	—
Teallach	+	+	+	+/-	—
Tadeus	+	+	—	+/-	+
Fuse	+	+	—	+/-	+
Goliath	+	+	+	—	—
Janus	—	+	—	—	—
Mecano	—	+	—	—	—
Genius	—	+	—	—	+

Table 1. the different approaches in MB-UIDE and the supported declarative models.

Trident [7] uses task, domain and presentation models to specify the interface. The task model is represented with an Activity Chaining Graph; the domain model by an entity-relationship diagram and the presentation model by the abstract user interface objects.

Teallach [2] uses the task, domain and presentation models to generate the user interfaces. Each model in Teallach defines a view of the information required to generate the interface to a particular application. The domain model describes the underlying application in terms its data and operation; the task model describes what the user can do with the user interface in terms of its dynamic and information processing requirement; and the presentation model indicates how the resulting interface will appear.

In Tadeus [17], like most of other MB-UIDEs, the UI developer has to create the task model which is represented by a hierarchical structure of the tasks; the domain model which presents the important entities of the application; the user model which presents the characteristics of the

users. Specially, the dialogue model is semi-automatically created based on three created models. The dialogue model is used to describe the interface in term of views and dialogues.

In FUSE [15], the use of declarative models is similar. The task model is used to describe the task hierarchy of the application model. The domain model is used to describe the functions and the data structure of the relevant user interfaces. The user model is used to describe the static and dynamic properties of the user groups and individual users which influences both the UI generation process and the kind and depth of the help offered by the user guidance component. The dialogue model of Fuse is generated based on three models including task, domain and user models. This model is used to describe the transformation of the task, domain and user models.

Goliath [6] uses the declarative models including the application, presentation and the dialogue models to generate the user interfaces. Goliath’s application model is used to define the data types and function signatures. The presentation model is used to describe the basic presentation elements that play a part of the interface. The dialogue model is defined in terms of abstract containers.

Janus [18] emphasizes the use of object-oriented domain model to generate the interface. In this model, a domain object is represented by a class and a class is described by its attributes and methods. During the automatic generation process, a window is generated for each class; its attributes are transformed into controls; and its methods into buttons or menu items.

Like Janus, Mecano [9] uses the domain model to generate the static layout and the dynamic behavior of an interface. The domain model of Mecano is a representation of the objects in a domain and their relationships. The interface model contains all the facets of an interface design.

Genius [8] generates the interfaces for database oriented applications through an existing domain model represented as an extended entity relationship model. In order to structure the information with respect to tasks of the user, the views are defined in the domain model. A view consists of a subset of entities, relationships and attributes of the overall data model. The dialogue model is generated from these views.

Design tools, such as those proposed in Table 2, are essential for modeling the tasks, domain’s objects, UI’s objects ... and for generating the interfaces.

MB-UIDE	Tool	Language used for describing generated user interface
Trident	SEGUIA SIERRA	AION/DS User interface language
Teallach	Teallach	Java code
Tadeus	Tadeus	Textual description for

		UIMS
Fuse	FIRE / FLUID BOSS PLUG-IN	C++ code
Goliath	Goliath's design tool	Caml
Janus	OOA-Tool	C++ code
Mecano	Browser tool Intelligent designer tool	
Genius	ER diagram editor ER : Entity Relationship	Textual description

Table 2. The different approaches in MB-UIDE and their tools.

SEGUIA and SIERRA tools developed for the Trident [7] approach. SEGUIA tool semi-automatic generates the user interface by making questions and suggesting responses to the designer. SIERRA tool generates the guidelines for designing user interface.

Teallach [19] provides three separate editors for manipulating the models. Most of the modeling operations are performed ergonomically by drag-and-drop or cut-and-paste. The user interface is generated in java.

In Tadeus [17], the user interface prototype is automatically generated from the task, domain and dialog models. This prototype is described in terms of textual descriptions which can be implemented through the tool User Interface Management System.

FUSE [15] consists of the BOSS (BedienOberflächen-SpezifikationsSystem), FLUID (FormaL User Interface Development), PLUG-IN (PLan-based User Guidance for Intelligent Navigation) and FIRE (Formal Interface Requirements Engineering) components that may also be used independently. FIRE provides graphical editors for setting up the task, domain and user models. FLUID is used to generate the specification of static and dynamic properties of a logical user interface. Based on this logical user interface, BOSS is used to generate the implementation of the user interface in terms of C++ code. Finally, PLUG-IN is used to generate the user guidance components.

Goliath [6] provides a graphical editor for modeling the task, domain and presentation models. The interface generated by Goliath is a complete one implemented in Caml.

OOA-Tool (*Object-Oriented Analysis*) in Janus [18] generates user interfaces described in C++ code and designed to be as ergonomic as possible.

Mecano [9] provides two separate tools: a browser and an intelligent designer. The browser tool is used to define, review and inspect the Mecano's domain model. Based on this model, dialog specifications are generated. These specifications are classified into high-level and low-level

dialogs. The Intelligent tool creates the abstract interface object based on the created dialogs.

In Genius [8], models are created by an ER diagram editor to generate the executable user interfaces for database-oriented applications. The generated user interface is described by a specific User Interface Management System, but there is no user interface editor.

Most approaches discussed above try to generate the complete and executable user interface based on some of the task, domain, user, presentation, and dialog models. Specially, these approaches have used the domain model although this model is differently defined in the different approaches. Beside of user interface generation, some of the approaches have also generated the application code for performing the generic application functions such as Teallach, Goliath. Most of these functions have a connection with the edition of data in the database. This also is a goal of DB-USE, but unlike with the current approaches, these functions will be automatically created by the DB-USE system instead of being created by the designer. Teallach specifically allows create the interface to non database application by creating domain components which don't derive from the underlying database but derive from the library of Java [19]. This is an important point that DB-USE doesn't use yet.

DB-USE MODELS

We describe in this section the main declarative models used in DB-USE.

Task Model

The task model records the tasks potential end-users of the system may need to perform to do their jobs, independently of dealing with a particular computer platform [20]. Many designs of an interactive system are generated to support these tasks. The task model provides support for modeling both the structure of the tasks and the flow of information between the tasks when carrying out the tasks. The structure of the tasks in the task model is described by the relationships between these tasks. These relationships can be of various types such as temporal and semantic relationships.

DB-USE uses the task model to express how an end user may want to interact with a system in order to reach a given goal. This expression is intended to be independent of any particular implementation or technology. This explains why a same set of models could initiate several different user interfaces [13].

DB-USE's task model is transformed into UIs starting with the ConcurTaskTrees Environment-CTTE [20]. User interface generation will then be ensured by DB-USE.

The task model described by CTTE is created at the analyst level. In order to reuse it in our process, it has to be transformed into a task model at the design level. At this level, the task model has to describe the function the user can do with an interface, how the user can interact with

the system. DB-USE's task model is used to derive the interface. Its task types differ from CTTE and are defined as follows:

- An Action task describes the end-user's command to the system such as close a dialog, delete a data record, search information, open a dialog and so on.
- An Operation task describes the display of information to the end-user or the reception of the information from the end-user.

Task mapping	
 Abstraction task	 Action task
 Interaction task (Control type)	 Action task
 Interaction task (Edit/Monitoring/Selection)	 Operation task
 Cooperation task	 Action task
 Application task	 Operation task
 User task	None

Table 3. Task Mapping.

Table 3 illustrates the mapping of abstract, interaction, application and user tasks from the CTTE's task model at the analyst level to action and operation tasks of DB-USE's task model at the design level. Figure 1 depicts an example of the mapping between tasks in ConcurTask-Trees and in our process.

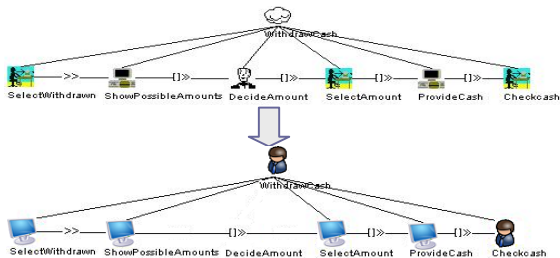


Figure 1. Mapping of WithdrawCash task from the task described in CTT to the task described in DB-USE.

In the new task model, the structure, the relationship between the supertask and its subtasks, the temporal operators are all reused for the syntactic and semantic aspects.

Domain Model

A domain model is a representation of both the objects in a domain and their relationships. The information in the domain model is basically the data model where the data objects are defined including the relationships between the data objects, and other information that is pertinent to the relationships such as business logic. A domain model may thus include a data model of the domain. In our process, the domain model will be built from a certain database (See Figure 5).

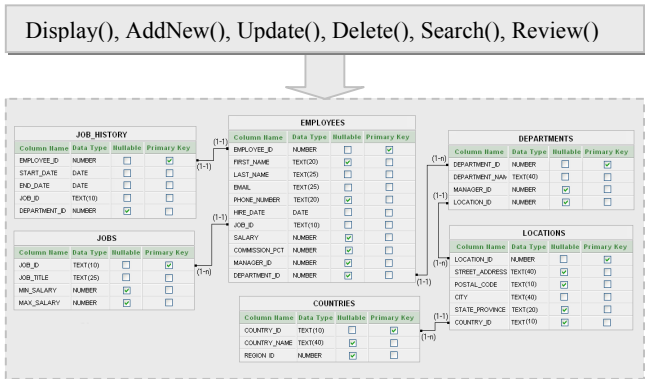


Figure 2. DB-USE's domain model – Employee management.

It is used to identify the attributes of a concrete user interface and to build the important functions of a database application. Besides, in order to obtain the desired behavior of a database application task, the generic functions are also defined in the model; these functions are, for instance, Display(), AddNew(), Update(), Delete(), Search(), Review(), Cancel() and Exit(). Unlike Teallach and Goliath, the generic application functions are simply defined in the domain model and they will be analyzed and built in more detail by the system once the function is linked to a concrete task in a concrete context.

User Model

The user model describes the characteristics of the desired users or groups of users such as experience, skill, knowledge, character. The main purpose of a user model is to support the user interface designer to create the user interfaces which tend to the preference of the user. The user model captures capabilities and limitations of the user population, for instance the kind of interaction techniques that are available for visually disabled people differs from the techniques for other human beings. The user model plays an important role in the user interface design; based on it, the designer will specify a complex or a simple user interface. For instance, creating the user interface for the experimented users who have experience with the software applications is different from creating the user interface for newcomers who have never used the computer.

User characteristics can be classified as application-independent or -dependent. Application independent characteristics include preferences, capabilities, psychomotor skills, etc. Application dependent characteristics include goals, knowledge of system and application, etc. In the DB-USE's user model, the users of the system are grouped into three groups (Complex, Mean and Simple) based on the user characteristics mentioned above. Figure 3 depicts DB-USE's user model which describes the user's experience, cultural and psychological characteristics. Then users are grouped based on these characteristics.

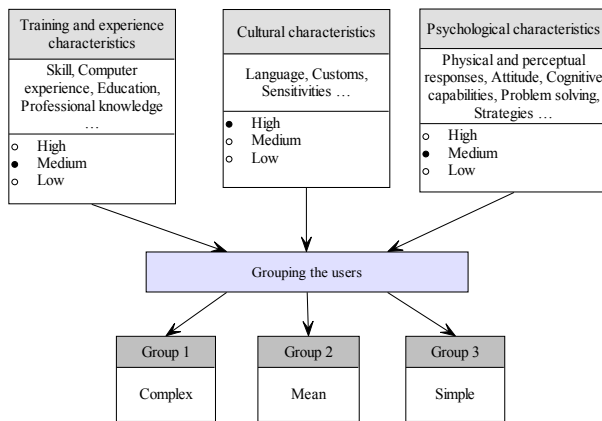


Figure 3. Grouping users based on characteristics.

ENGINEERING USER INTERFACE FROM COMBINING TASK, DOMAIN AND USER MODELS

Figure 4 depicts the main components and the architecture of DB-USE. The Model editor agent uses the task-, database- knowledge bases to load the task model from a XML file, the domain model from a concrete database and the user model from a diagram file. The loaded tasks have already been manually linked to the attributes of the domain objects. From these linked objects, the UI builder agent automatically creates the user interface (UI) objects based on the mapping rules [1]. The Function editor agent uses the Function description base to build the functions of the application which are defined in domain model. Once the UI objects have been created and the functions structured, the code generator agent produces the code. Specifically, our process does not only generate the user interface code, but also the application code behind needed to perform these pre-determined functions. In summary, the components of DB-USE are:

- The Task-knowledge base that describes the rules of the task model.
- The Mapping rules base that describes the rules for specifying the concrete user interface from domain's objects and the relationships between these objects and for transforming the concrete user interface to the final user interface [1].
- The Database-knowledge base that describes generic aspects of the database tasks, the advantages of the syntax and the structure of a query.
- The Layout-knowledge base that contains the syntactic design guidelines for controls, windows and other widgets layouts. It also describes the semantic rules from which the control types are defined.
- The Messages base that contains the generic messages such as errors, warnings, information to users messages.
- The Function description base that describes the generic functions of a task of database application.

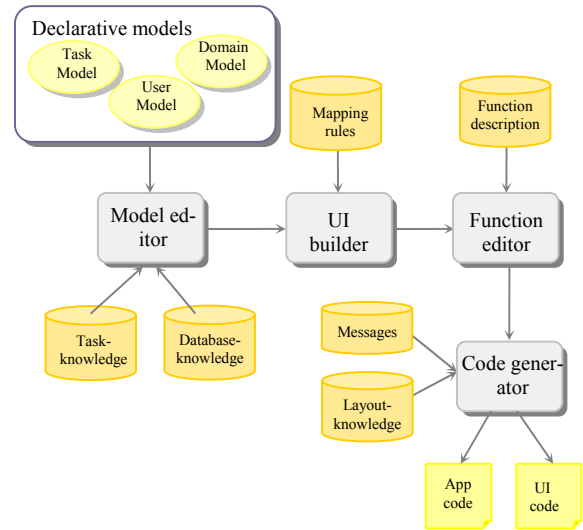


Figure 4. Main components and architecture of DB-USE.

DB-USE is specifically interesting to the benefit of the UI designer. It is built to support them as much as possible in every phases of the UI generation process. In DB-USE, this process consists of five phases: model analysis, relation making, UI design, application function design, and code generation (See Figure 5).

The **Model analyst** is responsible for the model analysis phase. The task, domain and user models are automatically built by the DB-USE system from the existing resources. The **Model Editor** is used to load the task model from a XML file, the user model from a diagram file, the domain model from a concrete database. The generic functions of a database application which are defined by DB-USE are included into its domain model. Once these models have been loaded, they are verified and modified by the **Model analyst**. He can change the attributes of the tasks such as name, type or delete an existing task or add a new one. Specially, he can create a new DB-USE task model with the **Model Editor** if it is not created yet.

The associations between the components of the task and domain models are made by the **UI Designer** in the relation making phase. Firstly, **UI Designer** has to specify a composite action task that he wants to generate the UI. Then DB-USE displays its sub-tasks based on rules RST (**R**ules for selecting the **S**ub-**T**asks). The operation tasks are linked to the domain's attributes and the action tasks are linked to the functions defined in domain model (See Figure 6). One operation task can be linked to more than one domain's attribute; one domain's attribute can be linked to more than one operation task. But one action task can be linked to only one function.

- **RST1:** All sub-tasks at level 1 are selected.
- **RST2:** All sub-tasks of a sub-task of task type operation is selected.

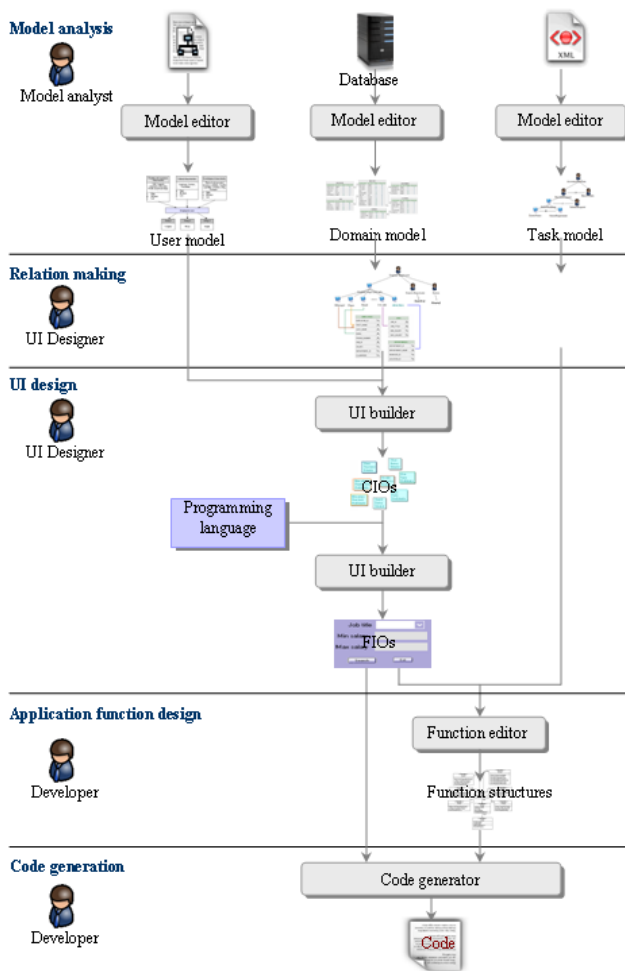


Figure 5. The metrological process of DB-USE.

In the UI design phase of the DB-USE process, the CIOs [16] (Concrete Interaction Objects) are created by the **UI Builder** based the associations made in the relation making phase and the user model. The user model is used to generate individual user interfaces and to select the control type among the possible ones that matches the preference of the end-users.

Once CIOs have been created, in order to reuse them in the different running environments (including platform, device and language); DB-USE stores them in terms of UsiXML (User Interface eXtensible Markup Language) [23]. Since the programming language has been determined, the FIOs (Final Interaction Object) are created based on the CIOs (See Figure 7).

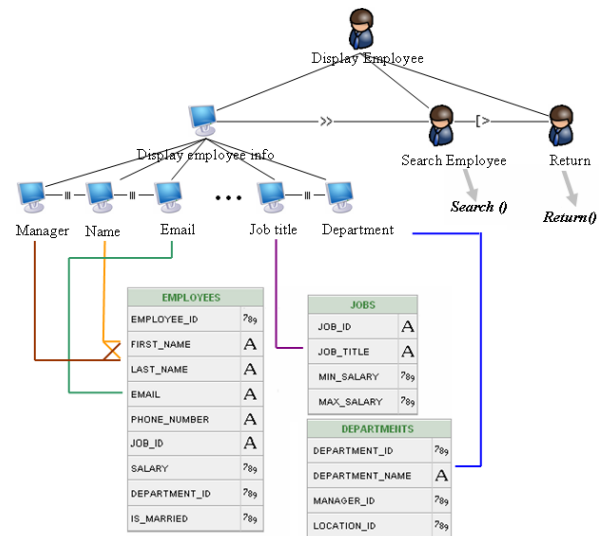


Figure 6. Associations between tasks and domain's components.

In order to specify the attributes of the UI objects, DB-USE uses the **Mapping rules** [1] base which defines the rules for making the associations between the tasks and domain's components, specifying the control type, location, dimension. In this phase, the **UI Designer** plays a crucial role since he has to make sure that the created UI objects satisfy the goal of the tasks.

The application function design phase starts once the FIOs have been completely created. The **Function** base will determine the linked functions of the created FIOs to construct the structure for these functions (Display(), AddNew(), Update(), Delete(), Search(), Review(), ...). One strong advantage of DB-USE is that application functions are automatically constructed by the system not by the **Developer**.

Finally, the UI code and the application code are generated in the code generation phase by the **Code generator**. The role of the **Developer** in this phase is to verify the generated code then modify it if needed in order to ensure it is successfully compiled. Typically, the code generated by DB-USE is clear and complete enough to be compiled and executed immediately.

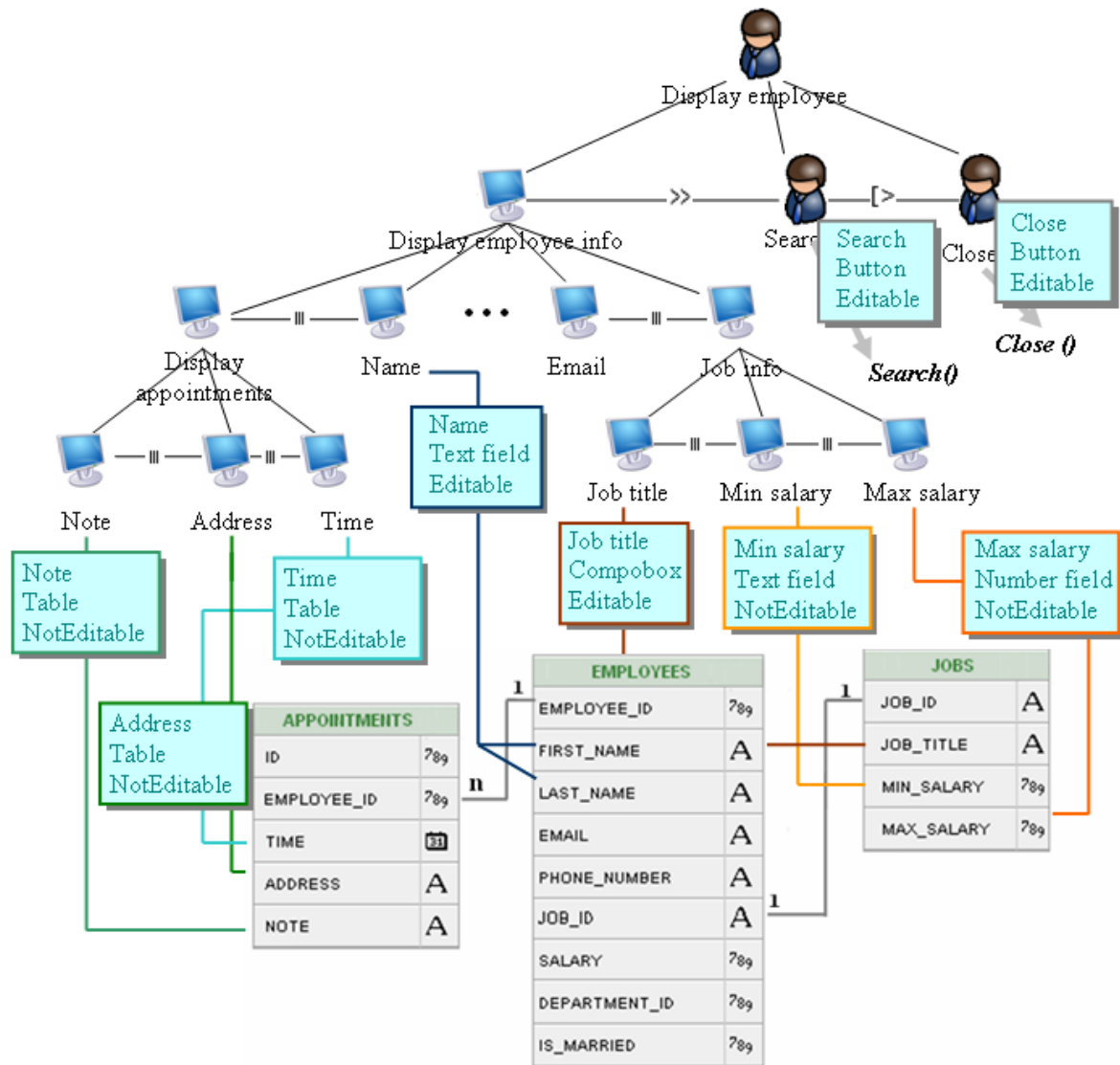


Figure 7. Creating FIOs based on the CIOs.

DB-USE TOOL

DB-USE has been implemented in Java, widely used in the large database-oriented software packages such as Oracle or SAP. Besides, Java is platform independent.

A presentation

The main purpose of DB-USE is to help designers to create the user interface and generate the application code for performing the generic functions of a database application easily and quickly. We summarize below the important goals DB-USE aims to achieve:

A friendly and simple interface: The interface of DB-USE has been designed in order that any designer can work with it giving her/him the important features mentioned above. Similarly to other applications, the main interest of our tool is the visual and/or graphical generated result.

Performing most of the functions discussed in the domain model: DB-USE has been developed to perform most of the functions defined in the domain model. These important functions makes the user interface directly specified based on the associations between the task and domain models; and the capability of the application code generation.

Managing efficiently existing resources: As already discussed, the task, user and domain models used in our process are the existing resources; they are used by the other processes such the analyst process.

Visualizing the process: The interface visually supports our process as much as possible. Currently, the tasks are displayed in terms of graphical objects. The designer can review the result in terms of graphics whenever he wants during working.

Generating and running the code: In the current version, the user interface and application codes generated by DB-USE are implemented in Java. The code generated has been made clear, easy to understand and documented. The generated code can be immediately compiled by a java compiler without modifications.

Figure 8 depicts DB-USE. The tool is divided into three separate areas. The first area is used to display the task model; the second the domain model and the third one the information of the UI objects

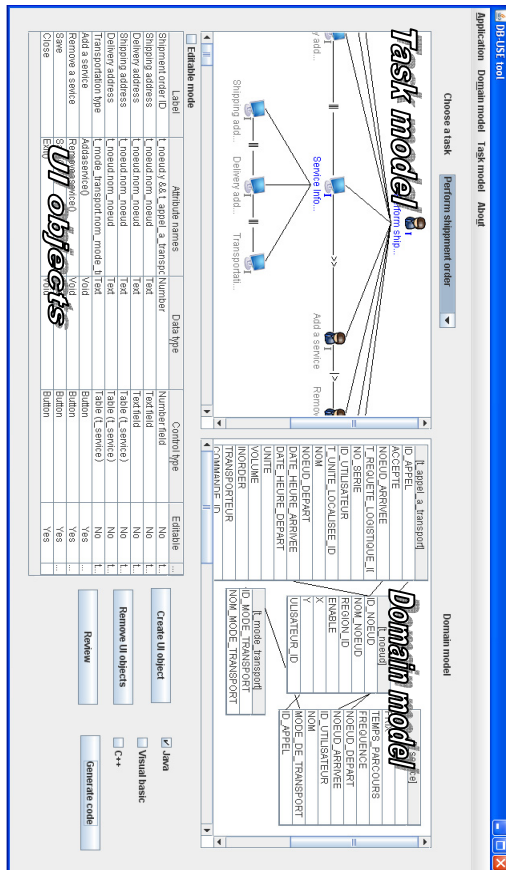


Figure 8. DB-USE Tool.

A concrete example - Edit Employee Information

In order to illustrate how the user interface code and application code are generated by DB-USE, a simple example is showed. The goal of this example is to allow the end-user review the employee information, create a new employee, delete an exiting one, and modify the employee information.

DB-USE performs this example through six steps.

Step 1: Loading Edit Employee Information Task model

The task model may be loaded from an existing CTTE task model or created by using DB-USE. (See Figure 9)

(1) in Figure 9 allows the designer to create a new task model, load an existing one, or save it in term of DB-USE's task model.

(2) in Figure 9 allows the designer to edit the tasks in the task model.

(3) in Figure 9 is the task dialogue which is used to modify the task's information.

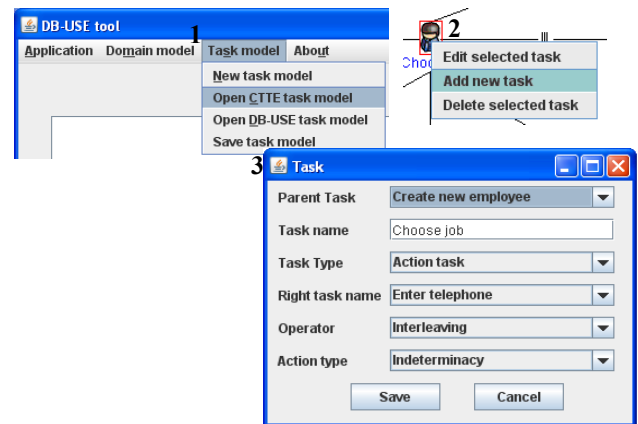


Figure 9. Loading or creating task model.

Edit Employee Information task model is showed in Figure 10.

Step 2: Loading domain model

The domain model is loaded from Oracle database.

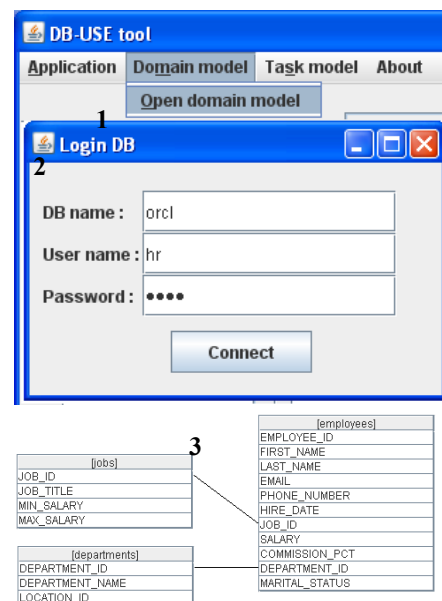


Figure 11. Loading domain model.

(1), (2) and (3) are the steps performed to load the domain model. Edit Employee Information task involves to only three objects including Jobs, Departments, and Employees. So the other objects are not showed in this case. Beside of creation of objects, the DB-USE also creates the application functions in this model. These functions are showed in the function popup menu of Figure 11.

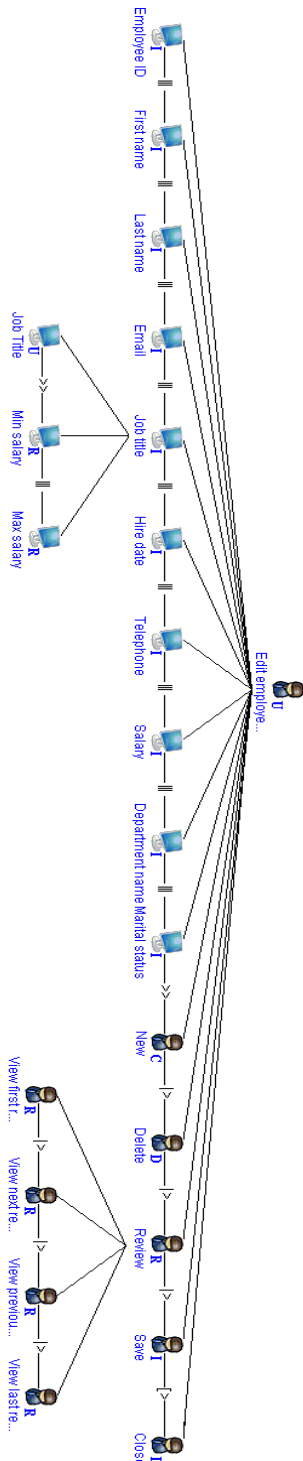


Figure 10. Edit Employee task model.

Step 3: Making the associations between the tasks and the domain's components:

The association between the operation tasks and the attributes of the domain's objects is created by selecting a concrete task and the attributes then just simply clicking on the "Create UI object" button. The association be-

tween the action tasks and defined application functions is created by selecting a concrete task, with the mouse, then selecting the application function in the menu (See Figure 11).

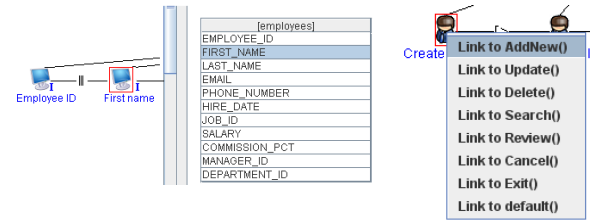


Figure 11. Making the associations between the tasks and the domain's attributes, the generic functions.

For *Edit Employee Information* task: The *Employee id*, *First name*, *Last name*, *Email*, *Hire date*, *Telephone*, *Salary*, *Marital status* tasks are linked to *Employee_ID*, *First_Name*, *Last_Name*, *Email*, *Hire_Date*, *Telephone_Number*, *Salary*, *Marital_Status* attributes of the *Employees* object. The *Create new employee* task is linked to *cancel()* function to reset the control to empty; the *Save* task is linked to *AddNew()* function to insert to data to the database; the *Delete* task is linked to *Delete()* function to delete the current record; the *Review* task is linked to *Review()* function to review the first, previous, next and last records; the *Close* task is linked to *Exit()* function to close the form.

Step 4: Creating the UI objects.

Label	Attribute names	Data type	Control type	Editable
Employee ID	employees.employee_id	Number	Number field	Yes
First name	employees.first_name	Text	Text field	Yes
Last name	employees.last_name	Text	Text field	Yes
Email	employees.email	Text	Text field	Yes
Job Title	jobs.job_title	Text	Combo box	Yes
Min salary	jobs.min_salary	Number	Number field	No
Max salary	jobs.max_salary	Number	Number field	No
Hire date	employees.hire_date	Date	Date picker	Yes
Telephone	employees.phone_number	Text	Text field	Yes
Salary	employees.salary	Number	Number field	Yes
Department name	departments.department_name	Text	Combo box	Yes
Marital status	employees.marital_status	Boolean	Radio control	Yes
Create new employee	Cancel()	Void	Button	Yes
Save	AddNew()	Void	Button	Yes
Delete	Delete()	Void	Button	Yes
Review	Review()	Void	Button	Yes

Figure 12. User interface object are displayed in terms of text.

Once an association has been created, a UI object is automatically created by DB-USE. The information of this UI object is displayed in a table as show in Figure 12 This information includes the label of the control, the attribute linked, the data type of these attributes, the control type and the editable attribute of control.

Step 5: Storing the UI objects by using UsiXML

The created UI objects will be described in UsiXML such as represented in Figure 13 by selecting **Application \ Save UI as UsiXML** menu. The objective of using the UsiXML is that the UI can be implemented in the differ-

ent environments. Figure 13 depicts how to save the UI objects in terms of UsiXML.

```
<outputText id="id1" name="Employee ID label" content="xpath" defaultContent="Employee ID" isVisible="true"
isBold="false" textColor="#000000"/>
<inputText id="id0" name="Employee ID control" maxLength="50" numberOfColumns="1" isEditable="true"/>
<outputText id="id1" name="First name label" content="xpath" defaultContent="First name" isVisible="true"
isBold="false" textColor="#000000"/>
<inputText id="id2" name="First name control" maxLength="50" numberOfColumns="1" isEditable="true"/>
<outputText id="id3" name="Last name label" content="xpath" defaultContent="Last name" isVisible="true"
isBold="false" textColor="#000000"/>
<inputText id="id4" name="Last name control" maxLength="50" numberOfColumns="1" isEditable="true"/>
<outputText id="id5" name="Job title label" content="xpath" defaultContent="Job title" isVisible="true"
isBold="false" textColor="#000000"/>
<comboBox id="id6" name="Job title control" isEnabled="true" currentValue="" />
<outputText id="id7" name="Department name label" content="xpath" defaultContent="Department name"
isVisible="true" isBold="false" textColor="#000000"/>
<comboBox id="id8" name="Department name control" isEnabled="true" currentValue="" />
<button id="id9" name="Add a new employee" content="xpath" defaultContent="New"/>
```

Figure 13. Some UI objects in Figure 12 are described by using UsiXML.

Step 6: Creating the final user interface

Based on the programming language determined by the designer and the created concrete interaction objects, the final interaction objects are automatically specified.

In order to help the designer to materialize the user interface that will be generated during the last step of the process, DB-USE has to allow to review the user interface in terms of graphics.

DB-USE generates the user interface code based on the user interface objects that have been created and the application code based on the selected functions. The code generator produces java code by using graphical libraries such as Swing and AWT. Unlike the current applications, DB-USE generates both the user interface and application codes. The generated code is complete and can be compiled and ran immediately when usually code generated by other applications are just prototypes or code skeletons.

Figure 14. *Edit Employee Information* form.

Figure 14 depicts *Edit Employee Information* form that is generated from the previous steps.

CONCLUSION

To be efficient, data-intensive systems that are an important component of today's software applications need effective human-computer interaction. User interfaces for such data systems has been a recurrent research issue and nowadays these UI have to support automatic generation to adequately be dealt with.

This framework has aimed at offering a low cost, short time-to-implementation and efficient UI development environment from the business user side. Indeed, the objective of this tool has not only been to generate the user interfaces but also to generate the application code for performing the generic functions of the database applications. In future, the navigation between the dialogues will be automatically specified based on the operators of the task; the position and dimension attributes of UI object will be optimized; and finally, allow define the data type which do not come from the database.

This research has also contributed to define rules for mapping task and domains models to one another in both ways and translate these models into code in order to automate the user interface design process.

REFERENCES

1. Tran, V., Vanderdonckt, J., Faulkner, S., Generating User Interface from Task, User and Domain Models. In *Proc. of 2nd Int. Conf. on Advances in Human-oriented and Personalized Mechanisms, Technologies, and Services CENTRIC'2009* (Porto, September 20-25, 2009), IEEE Computer Society Press, Los Alamitos, 2009, pp. 19-26.
2. Griffiths, T., Barclay, P.J., Paton, N.W., McKirdy, J., Kennedy, J., Gray, P.D., Cooper, R., Goble, C.A., Pinheiro da Silva, P., Teallach: a model-based user interface development environment for object databases. *Interacting with Computers*, Vol. 4, No. 1, December 2001, pp. 31-68.
3. Eisenstein, J., Puerta, A., Adaptation in automated user-interface design. In *Proceedings of the 5th international conference on Intelligent user interfaces*, p.74-81, January 09-12, 2000, New Orleans, Louisiana, United States.
4. Nichols, J., Faulring, A., "Automatic Interface Generation and Future User Interface Tools". In: *CHI. Proceedings of the Workshop on the Future of User Interface Design Tools*, 2005.
5. Pinheiro da Silva, P., Griffiths, T., N. Paton, 2000. Generating user interface code in a model based user interface development environment. In: *Proceedings of Advanced Visual Interfaces*. ACM Press, New York, pp. 155-160.

6. Julien, D., Ziane, M., and Guessoum, Z.. GOLIATH: An extensible model-based environment to develop user interfaces. In *Proceedings of the Fourth International Conference on Computer Aided Design for User Interfaces*, P. 95–106. Kluwer Academics Publishers, 2004.
7. Bodart, F., Hennebert, A.-M., Leheureux, J.-M., Provot, I., Sacre, B., Vanderdonckt, J. Towards a Systematic Building of Software Architectures: the TRIDENT methodological guide. In *Proc. of 2nd Eurographics Workshop on Design, Specification, Verification of Interactive Systems DSV-IS'95* (Toulouse, 7-9 June 1995), Ph. Palanque, R. Bastide (eds.), Springer-Verlag, Vienna, 1995, pp. 262-278.
8. C. Janssen, A. Weisbecker, J. Ziegler: Generating User Interfaces from Data Models and Dialogue Net Specifications. In *Ashlund S., Mullet K., Henderson A., Hollnagel E., White T. (eds.)*. New York: ACM Press 1993 (pp. 418-423).
9. Puerta, A., “The MECANO Project: Comprehensive and Integrated Support for Model-Based Interface Development,” *Proc. CADUI 96*, J. Vanderdonckt, ed. <http://www.info.fundp.ac.be/~jvd/dsvis/cadui96.html>.
10. Moroney, L. and MacDonald, M., ASP.NET Applications in Pro ASP.NET 1.1 in VB .NET From Professional to Expert, Apress, 2006, pp. 183- 230.
11. Myers, B., Hudson, S. E., Pausch, R.. Past, present, and future of user interface software tools. *ACM Transactions on Computer-Human Interaction (TOCHI)*, v.7 n.1, p.3-28, March 2000.
12. Pinheiro da Silva, P., User Interface Declarative Models and Development Environments: A Survey. In *Proceedings of DSV-IS2000*, volume 1946 of LNCS, P. 207-226, Limerick, Ireland, 2000. Springer-Verlag.
13. Wilson, S., Johnson, P.: Bridging the generation gap: From work tasks to user interface designs. In *Computer-Aided Design of User Interfaces*. Namur University Press, 1996, P. 77-94.
14. Mahfoudhi, A., Abed, M., Abid, M.. “Towards a User Interface Generation Approach Based on Object Oriented Design and Task Model”. *TAMODIA'2005 : 4th International Workshop on Task Models and Diagrams for user interface design For Work and Beyond Gdansk*, Poland September 2005. (P. 26-27)
15. Lonczewski, F., Schreiber, S.: The FUSE-System: An Integrated User Interface Design Environment. In: *J. Vanderdonckt (ed.): Computer-Aided Design of User Interfaces*. Namur University Press, 1996, 37-56.
16. Vanderdonckt, J. and Bodart, F. Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection. In *Proc. of the ACM Conf. on Human Factors in Computing Systems INTERCHI'93* (Amsterdam, 24-29 April 1993), ACM Press, New York, 1993, pp. 424-429.
17. Elwert, T. and Schlunbaum, E., Modelling and Generation of Graphical User Interfaces in the TADEUS Approach. In *Designing, Specification and Verification of Interactive Systems*, pages 193{208, Vienna, 1995. Springer.
18. Balzert, H., From OOA to GUI - The JANUS-System, in *Proceedings of the 5th IFIP TC13 Conference on Human-Computer Interaction*, Lillehammer, June 1995, pp. 319-324.
19. Barclay, P.J., Griffiths, T., McKirdy, J., Paton, N.W., Cooper, R., and Kennedy, J.. The Teallach Tool: Using Models for Flexible User Interface Design. In *pro. 3rd International Conference on Computer-Aided Design of User Interfaces*, Louvain-la-Neuve (Belgium), 21-23 October 1999.
20. Paternò, F., Mancini, C., and Meniconi, S., Concurtasktrees: A diagrammatic notation for specifying task models. In *Human-Computer Interaction*, pages 362–369. Chapman and Hall, 1997
21. Pribeanu, C., Tool Support for Handling Mapping Rules from Domain to Task Models. *Coninx, K., Luyten, K., Schneider, K. (Eds.): Proc. of TAMODIA 2006, Hasselt, Belgium, 23 – 24 October*. Lecture Notes in Computer Science - LNCS 4385, Springer 2007, pp. 16-23.
22. Laudon, Kenneth C., Laudon, Jane P., Management Information Systems. Pearson Education (US), 2009.
23. Limbourg, Q., Vanderdonckt, J., Multi-Path Transformational Development of User Interfaces with Graph Transformations, in Seffah, A., Vanderdonckt, J., Desmarais, M. (eds.), “Human-Centered Software Engineering”, Chapter 6, HCI Series, Springer, London, 2009, pp. 109-140.
24. Vanderdonckt, J. Model-Driven Engineering of User Interfaces: Promises, Successes, and Failures. In *Proc. of 5th Annual Romanian Conf. on Human-Computer Interaction ROCHI'2008* (Iasi, September 18-19, 2008), S. Buraga, I. Juvina (eds.). Matrix ROM, Bucurest, 2008, pp. 1–10.

A Proposal for Enhancing the UsiXML Transformation Meta-Model

Nathalie Aquino, José Ignacio Panach, Oscar Pastor

Centro de Investigación en Métodos de Producción de Software

Universidad Politécnica de Valencia

Camino de Vera s/n, 46022 Valencia, Spain

+34 96 387 70 07 Ext. 83534

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

ABSTRACT

UsiXML proposes a set of conceptual models that represent diverse aspects of user interfaces. Some of these models represent a user interface at a high abstraction level. From these high level models, other models at a lower abstraction level can be obtained by means of model-to-model transformations. Model-to-model transformations can occur between several abstraction levels. Finally, the user interface code is reached through a model-to-code transformation. Therefore, transformations are a fundamental piece in the UsiXML development process. Current UsiXML model-to-model transformation rules are specified using a graph-based notation. This strong dependency on graphs can result in a lack of efficiency in transformations when the amount of involved graphs is medium or large. In order to face this problem, we propose a transformation meta-model independent of graphs and any other transformation technology. The analyst can use this transformation meta-model to specify how transformations should be carried out throughout different abstraction levels. Once this specification has been finished, the analyst will be able to translate the transformation rules into a specific transformation language that performs the transformation. In the paper, we introduce an example to illustrate that a same transformation rule specified in our proposal can be translated into two different translation languages such as graph notation and ATL, depending on analyst's preferences.

Keywords

Model-driven engineering, UsiXML, meta-model, transformation, interaction modelling, conceptual model.

INTRODUCTION

Model-driven engineering (MDE) is a software development methodology which focuses on creating models, or abstractions, closer to concepts of a particular domain (i.e., at the problem space level) than to computing concepts (i.e., at the solution space level). According to [12], MDE is simply the notion that we can construct a model of a system that we can later transform into the real thing. MDE states that the analyst's effort must be gathered in a conceptual model, and the system is implemented later by means of transformation rules that can be automated or semi-automated. In other words, the MDE paradigm distin-

guishes between conceptual models, defined by analysts, and the code that implements the system, which can be generated with as much automation as possible from the corresponding conceptual model. In this context, model-to-model and model-to-code transformations are an essential piece to develop systems according to the MDE paradigm.

UsiXML [11] proposes an MDE method for user interface development. It comprises various types of high level models that allow to represent interactive tasks in a way that is independent of platforms and interaction modalities (e.g., character, graphical, vocal), as well as other lower level models that add the relevant details about platforms, devices, modalities, users, etc. Furthermore, UsiXML uses a transformational approach: higher level models are transformed to lower level models by means of model-to-model transformations, and the final user interface code is reached from lower level models by means of model-to-code transformations. Model-to-model and model-to-code transformations are specified by means of transformation rules.

The current proposal to perform model-to-model transformations in UsiXML is based on graph transformations [11]. The current version of the meta-model that characterizes this proposal is defined in the UsiXML V1.8 Reference Manual [21]. This meta-model incorporates classes which are too much related to the structure of graphs. This can be seen as a disadvantage, since it can result in a very high correlation between UsiXML and graphs transformations with their related tools. Furthermore, graph transformations have some disadvantages [22] that are inherited by UsiXML. For example, the number of nodes is critical. Graphs with many nodes require much time to perform transformations. Another critical factor is the maximum degree of nodes. The more degrees, the more time is needed to carry out transformations. These disadvantages suggest that graph transformations would not be suitable for models of large size.

In the literature, there are other proposals related to transformations which are not focused on graphs, like the Query/View/Transformation (QVT) standard [18] and the ATLAS Transformation Language (ATL) [2]. QVT is the standard of the Object Management Group for specifying model transformations. It aims to transform MOF-based

models. MOF (Meta-Object Facility) [14] is also a standard of the Object Management Group and is used for specifying meta-models. Its current version is MOF 2.0. The UML 2 meta-model (UML 2.0 and above) has been defined according to MOF 2.0, so all models compliant with UML 2 are MOF-based models and hence, can be transformed using QVT. QVT has been implemented in various languages. Among those languages, ATL is currently the most widely used [17]. ATL allows describing transformations using a syntax that is similar to the one of declarative and imperative programming languages. Since all UsiXML meta-models are defined using UML 2, we consider that features of languages that implement QVT, as ATL, are good options to take into consideration to perform UsiXML transformations.

In this paper, we propose some changes to the UsiXML transformation meta-model in order to enable the creation of transformation models in which rules are expressed in a way that is independent not only from the graph notation, but also from the specific syntax and notation of any transformation language. We incorporate concepts that are general enough to express the current UsiXML transformation rules that are expressed with the graph notation, as well as general transformation rules that can be expressed with languages like ATL. In fact, the proposed enhancements were inspired by features of ATL, taking care of not adopting its specific syntax and notation. Moreover, our proposed meta-model is also abstract enough to be used in other MDE methods that build user interfaces or fully functional software applications, like the OO-Method [15].

The main advantage of our proposal is that the analyst can specify transformations in a way that is abstract and independent of any specific transformation language. In a next step, it will be possible to automatically translate the transformation rules specified according to our proposed meta-model, to a selected transformation language. This translation must be performed according to the syntax of the target transformation language, such a way, transformations rules will be executed in the corresponding transformation technology. This transformation technology and language could be graph notation, ATL or other.

The remainder of the paper is structured as follows: Section 2 presents related works. Section 3 provides a brief description of UsiXML and its transformational approach. Section 4 presents our proposal for enhancing the current UsiXML transformation meta-model. Section 5 illustrates our proposal with an example. And, finally, Section 6 presents conclusions and future works.

RELATED WORK

In the literature there are many proposals to perform transformations. Graph grammars and graph transformations have been recognized as powerful techniques for specifying complex transformations that can be used in various situations in a software development process [1].

Besides graph transformations used in UsiXML, there are many other proposals based on graphs. For instance, Karsai [8] has defined a graph-transformation-based technique for specifying model transformations. Karsai proposes starting from a domain specific modelling language specified with UML class diagrams. Domain specific models are networks of objects, where each object (link) belongs to a corresponding class (association) in the class diagram. From a mathematical point of view, domain specific models are graphs where the labels denote the corresponding entities in the meta-model. According to the author, the design transformation process specified in this way is formal, and it assigns a semantic to the input models in terms of the target domain.

Another author who proposes performing transformations by means of graphs is Gogolla [6]. This author has studied transformation rules on the UML meta-model layer. The aim of those rules is to identify the minimal set of UML concepts, a UML core, which is necessary in order to express all UML language features. Due to the diagrammatical nature of UML, graph transformation is a natural language for the formulation of such rules. Specifically, Gogolla has defined transformations for class diagrams and statechart diagrams.

Other authors propose performing transformations by means of algebraic compositions. In this group, we can comment the work of Ho [7]. Ho has specified a freely available UML transformation framework for manipulating UML models, called UMLAUT. These manipulations are expressed as algebraic compositions of reified elementary transformations. Thus, they are open to extension through inheritance and aggregation. By means of UMLAUT, a UML model of a distributed application can be automatically transformed into a labelled transition system validated using a pre-existing protocol validation tool.

Other authors have defined more formal transformations than algebraic compositions. In this group, we would like to mention the work of Caplat [4]. This author has presented a model for Model Mappings based on formalisms. As a model is expressed in a given formalism, the transformation of a model into another model leads to compare the formalisms they are based upon, i.e., to compare their primitives and their semantics. As a consequence, model mapping can be seen as a meta-modelling activity.

A widely used transformation language is the Extensible Stylesheet Language Transformation (XSLT) [23]. XSLT is the language used in XSL style sheets to transform XML documents into other XML documents. An XSL processor reads the XML document and follows the instructions in the XSL style sheet, and then it outputs a new XML document or XML-document fragment. Some authors, like Peltier [16], propose carrying out transformations using this language. He has defined a process called MTRANS to perform transformations. MTRANS uses XSLT to transform models, but in a more abstract way than XSLT.

MTRANS is based on a meta-modelling approach, where a meta-model is used to define the semantics of each model. The MTRANS framework supplies a language and an environment to write models transformations. The language is composed by a set of instructions and a part depending on the particular used meta-model.

With regard to QVT (Query/View/Transformation) [18], as previously mentioned, it is a standard defined by the Object Management Group. Transformations in the context of QVT are classified into relations and mappings. On the one hand, relations allow verifying the consistence among related models. On the other hand, mappings implement transformations, in other words, they transform elements from one model to another. The ATLAS Transformation Language (ATL) [2] implements the QVT standard and it is composed of a mixture of imperative and declarative expressions. In the context of ATL, a source model is transformed into a target model by means of a set of transformations. Source and target models are specified by means of meta-models. ATL transformations are unidirectional, source model can only be read and target model can be modified.

Another proposal was developed by the Object Management Group: the Common Warehouse Metamodel Specification [5]. It specifies a model for describing transformations that introduces the concepts of black-box and white-box transformations. Both of these transformation styles only provide a relationship among model elements which are the sources and targets of a transformation, but do not express exactly what the resulting target will consist of. White-box transformations may have a procedure expression associated with the transformation, allowing for a program fragment in some implementation language to describe the implementation of the transformation.

Finally, there are authors who have defined a specific taxonomy, such as Mens [13]. By taxonomy, the author means “a system for naming and organizing things into groups which share qualities”. That taxonomy can help software developers in choosing a particular model transformation approach that is best suited for their needs; it can help tool builders to assess the strengths and weaknesses of their tools compared to other tools; and it can help scientists to identify limitations across tools or technology that need to be overcome by improving the underlying techniques and formalisms.

As conclusion, it is important to highlight the amount of proposals to express model-to-model transformations. Each proposal has advantages and disadvantages and there is not only one proposal more useful than others. The choice of one notation depends mainly on the analyst that will work with it. Comparing our proposal with existing ones, our work is not focused on a specific transformation language. This is the main contribution of our work with regard to existing proposals. This paper aims to propose a meta-model that is abstract enough to support the main concepts related

to transformation rules in such a way that these concepts can be expressed independently of the particular syntax or notation of a specific transformation language. Once the transformation has been specified by means of the transformation model, the analyst can transform that model into a specific transformation language, according to analyst's preferences.

USIXML AND ITS TRANSFORMATIONAL APPROACH

This section briefly presents UsiXML and its transformational approach, including a description of the current UsiXML transformation meta-model which is based on graph notation.

UsiXML [11] proposes a user interface development process which is compliant with MDE. The UsiXML approach proposes a user interface description language aimed at describing user interfaces with various levels of details and abstractions, depending on the context of use. UsiXML supports a family of user interfaces such as, but not limited to: device-independent, modality-independent, and context-independent.

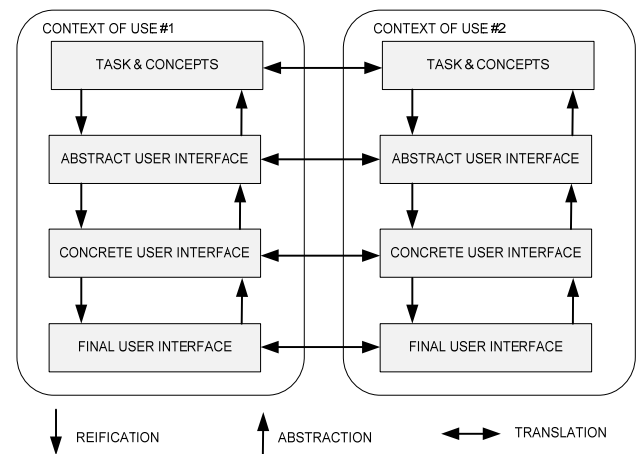


Figure 1. The four basic levels of abstraction of UsiXML and the three basic transformation types

UsiXML is structured according to four basic levels of abstraction defined by the Cameleon Reference Framework [3] (see Figure 1).

- **Task and Domain:** this level describes the various interactive tasks to be carried out by the end user and the domain objects that are manipulated by these tasks.
- **Abstract user interface (AUI):** this level provides a user interface definition that is independent of any modality of interaction (e.g. graphical interaction, vocal interaction, etc.).
- **Concrete user interface (CUI):** this level represents a user interface independently of any computing platform or programming toolkits peculiarities.

- **Final user interface (FUI):** in this level operational user interfaces are located, i.e., any user interface running on a particular computing platform either by interpretation or by execution.

Since the Task and Domain level is the more abstract level, and the FUI is the less abstract level, we can consider that these 4 abstraction levels are ordered and that there are 3 pairs of adjacent levels: 1) Task and Domain is adjacent to AUI; 2) AUI is adjacent to CUI; and 3) CUI is adjacent to FUI.

The Cameleon Reference Framework also exhibits three types of basic transformations (Figure 1): (1, 2) *Abstraction* (respectively, *Reification*) is a process of elicitation of artefacts that are more abstract (respectively, concrete) than the artefacts that serve as input to this process. Abstraction is the opposite of reification. (3) *Translation* is a process that elicits artefacts intended for a particular context of use from artefacts of a corresponding level of abstraction but aimed at a different context of use.

Previous works proposed a transformational approach to handle transformations between the different described abstraction levels and considering different interaction modalities and contexts of use [9, 10, 19, 20]. This transformational approach is based on graph transformation rules. Graph transformations are performed as follows. Let G be the initial UsiXML specification, when 1) a *Left Hand Side (LHS)* matches into G and 2) a *Negative Application Condition (NAC)* does not match into G , 3) the LHS is replaced by *Right Hand Side (RHS)*. G is consequently transformed into G' (the resultant UsiXML specification). All elements of G that are not covered by the match are left unchanged. This transformation approach is sustained by TransformiXML, a tool that allows the definition and application of transformation rules.

The current version of the meta-model that characterizes this transformational approach is defined in the UsiXML V1.8 Reference Manual [21]. Figure 2 illustrates this meta-model. As we can see in Figure 2, the current transformation meta-model incorporate concepts that are strongly related to graph transformation rules. Next, according to the definitions provided in [21], we briefly explain each class of this meta-model.

- **uiModel:** is the topmost superclass containing common features shared by all component models of a user interface.
- **transformationModel:** contains a set of rules enabling the transformation of one model, at a certain level of abstraction, into another, or to adapt a model for a new context of use.
- **developmentPath:** refers to the abstraction level where the transformation starts (Task and Domain, AUI, CUI or FUI), and the abstraction level where the transformation ends. The source and target levels can be non-adjacent.

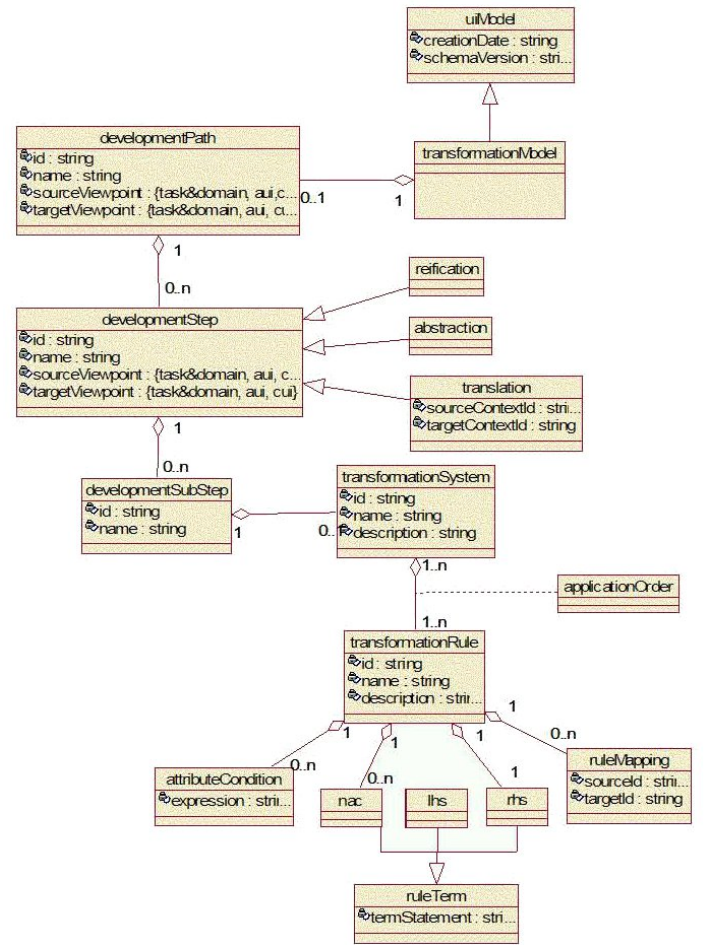


Figure 2. UsiXML transformation meta-model [21].

- **developmentStep:** is a transformation between adjacent abstraction levels, or a direct transformation between 2 contexts of use at the same abstraction level. There is a specialization of this class for each basic transformation type: reification, abstraction and translation.
- **developmentSubStep:** a development step can be composed of several sub-steps that need to be carried out to accomplish a transformational goal. For instance, when transforming from Task and Domain to AUI (development step), the following sub-steps, or goals, were identified in [10]: 1) identification of AUI structure; 2) selection of abstract individual components; 3) spatio-temporal arrangement of abstract interaction objects; 4) definition of abstract dialog control; and 5) derivation of AUI to domain mappings.
- **transformationSystem:** is composed of a set of sequentially applied transformation rules that realize a basic development activity.
- **transformationRule:** performs a unitary transformation operation on a model. It is composed of a LHS, a RHS, and a NAC.

- **applicationOrder:** defines the order of execution of a transformation rule with respect to a particular transformation system.
- **attributeCondition:** textual expression indicating a condition scoping on element attributes of the LHS of a transformation rule.
- **ruleTerm:** is the content of a rule. It is composed of any fragment of uiModel.
- **nac:** NAC attached to a transformation rule.
- **lhs:** LHS of a transformation rule.
- **rhs:** RHS of a transformation rule.
- **ruleMapping:** defines the source and target models of a transformation rule.

This meta-model has been used as the basis for our proposal, as we explain in the next section.

A PROPOSAL FOR ENHANCING THE USIXML TRANSFORMATION META-MODEL

This section presents a transformation meta-model abstract enough to represent transformations independently of the underlying structure of a specific transformation language or technology. The proposed meta-model aims to enhance the current UsiXML transformation meta-model avoiding the use of explicit concepts related to graph-based transformation rules. Hence, our proposal has been based on the UsiXML transformation meta-model presented in [21]. On this basis, we have tried to eliminate explicit references to graphs concepts, and we have tried to add more expressiveness taking as inspirations features from ATL. We have also tried to provide clear explanations for all concepts that appear in our proposed meta-model.

Figure 3 presents the Transformation package of our proposed meta-model. The main concepts related to transformations are included in this package. Next, these concepts are explained.

Different *Models* can be created to specify an interactive system. For instance, we could use models of tasks, domain, AUI, CUI, context, etc., in order to specify an interactive system. All these models have some basic properties that were defined in [21].

A *Transformation Model* is a specialization of Model that has the aim to specify the rules that enable the transformation of a model at a certain level of abstraction, into another model at a different level. A Transformation Model also allows the adaptation of a model for a new context of use. When defining a Transformation Model, it is necessary to specify the initial abstraction level and the final abstraction level. These abstraction levels could be adjacent levels or not (see Figure 1). For instance, we could define a Transformation Model for transforming from the Task and

Domain level to the adjacent AUI level, or for transforming from the AUI level to the non-adjacent FUI level. Furthermore, when defining a Transformation Model it is also necessary to specify the initial and final contexts of use. These contexts could be the same or different ones. For instance, we could define a Transformation Model for transforming from FUI models at context C1 to CUI models at context C1 or to CUI models at context C2. It is important to note that when considering different contexts of use, it is also meaningful to specify a Transformation Model with the same initial and final abstraction level, but for different contexts. For instance, we could define a Transformation Model for Transforming from CUI models at the C1 context to CUI models at the C2 context.

A Transformation Model aggregates an ordered set of *Transformation Steps*. A Transformation Step is a transformation between adjacent abstraction levels, or a direct transformation between two contexts of use at the same abstraction level. A Transformation Step is specified with a source abstraction level and a target abstraction level. Source and target must be adjacent abstraction levels, or the same level.

When the source abstraction level is higher than the target adjacent abstraction level, the Transformation Step is a *Reification*. For instance, we could define a Reification in which the source abstraction level is AUI and the target abstraction level is CUI. When the source abstraction level is lower than the target adjacent abstraction level, the Transformation Step is an *Abstraction*. For instance, we could define an Abstraction in which the source abstraction level is CUI and the target abstraction level is AUI. Another specialization of a Transformation Step is the *Translation*. A Translation allows the specification of a direct transformation, at a same abstraction level, from a source context of use to a target context of use. For instance, we could define a Translation in which the source abstraction level and the target abstraction level are CUI, the source context is C1, and the target context is C2. Thus, the ordered set of Transformation Steps that are chosen for a Transformation Model must be composed of Abstractions, Reifications and/or Translations that allow reaching the target abstraction level and the target context, from the source abstraction level and context.

Each Transformation Step is an ordered aggregation of *Transformation Systems*. A Transformation System groups a set of rules that accomplish a transformational goal. As an example, we have previously mentioned the goals that need to be reached when performing the transformation step from Task and Domain to AUI. Identification of the AUI structure and selection of abstract individual components are two of these goals. Each one of them corresponds to a transformation system.

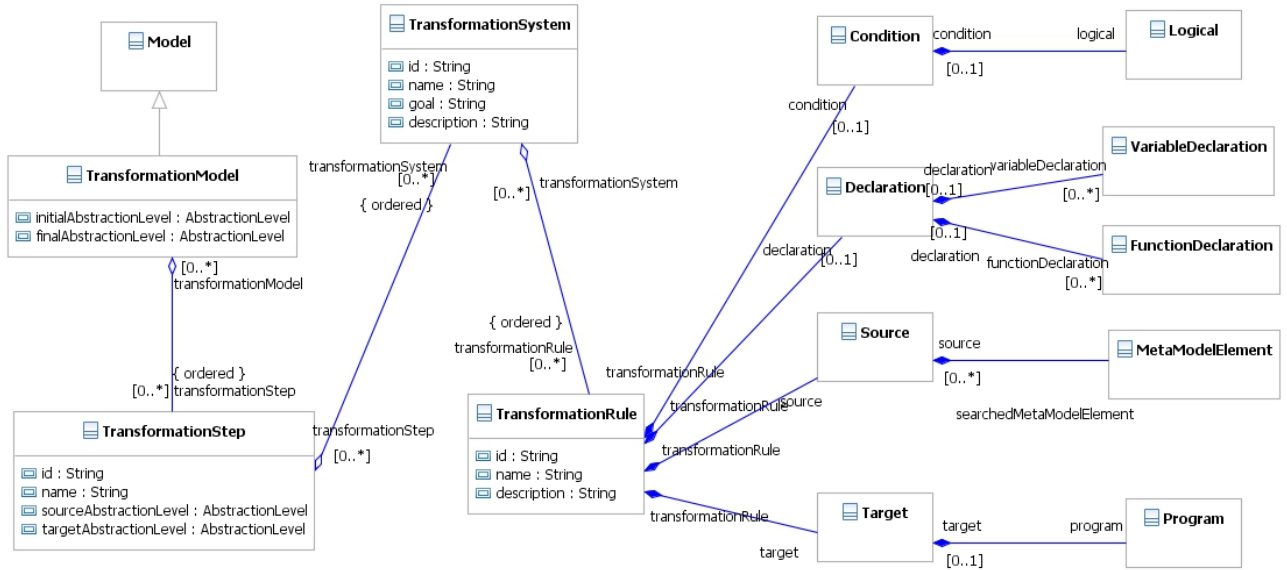


Figure 3. Transformation package of the new proposed transformation meta-model.

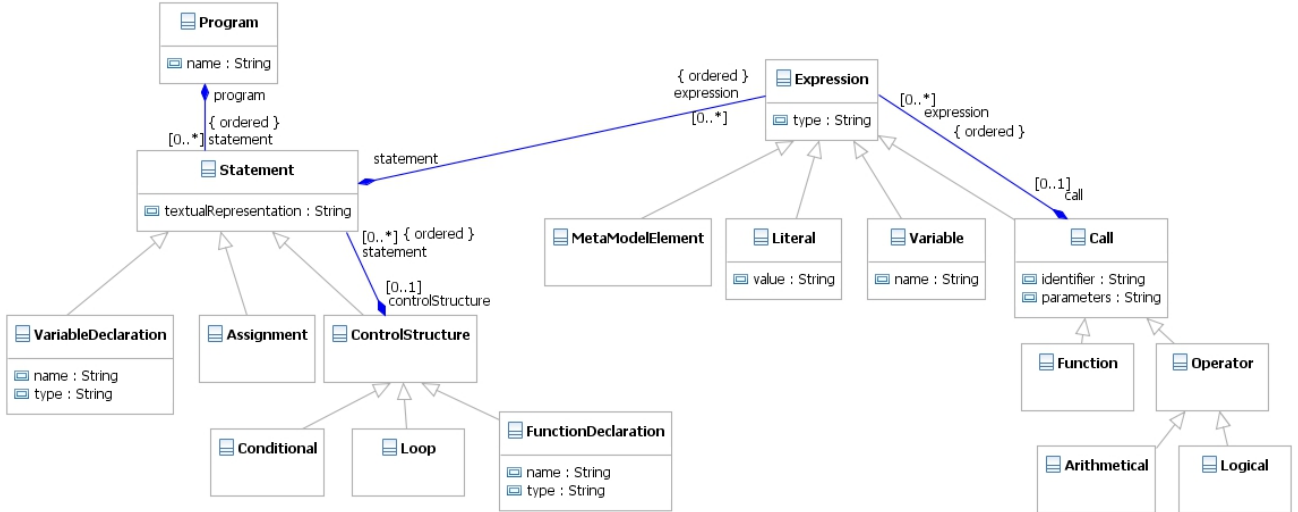


Figure 4. Program package of the new proposed transformation meta-model

Each Transformation System is an ordered aggregation of *Transformation Rules*. A Transformation Rule is a unitary transformation operation. For instance, the transformation system that corresponds to the goal of identifying the AUI structure is composed of several transformation rules, one of which generates an abstract container for each sub-task of the root task [19].

Regarding its structure, a transformation rule has an optional *Condition*. If the condition exists, it must be satisfied for the transformation rule to be applied. The condition is composed of a *Logical* expression that can be evaluated to true or false. A transformation rule can also have an optional *Declaration* section. In this section, auxiliary resources (e.g., variables, functions) can be specified in order to be used later in the specification of the rule. Hence, the declaration section can be composed of *Variable Declara-*

tions and *Function Declarations*. Each transformation rule has a *Source* that represents a *Meta-Model Element* whose instances belong to the source model. Finally, each transformation rule has a *Target* that represents what will be written in the target model. The target of a transformation rule is associated to a *Program*, which has enough expressivity to specify what needs to be written in the target model.

The previously mentioned concepts Logical, Variable Declaration, Function Declaration, and Program, are defined in the *Program package*, which is illustrated in Figure 4. This package represents a generic imperative programming language with which *Programs* can be written. A program is composed of an ordered set of *Statements*. *Variable Declaration* and *Assignment* are specializations of a statement which can not be composed of other statements. A variable

declaration allows defining the name and type of a variable. An assignment allows specifying a value for a variable. *Control Structure* is another specialization of statement which can be composed of other ordered statements. Conditional, Loop, and Function Declaration are specializations of control structure. *Conditional* is a type of statement in which it is possible to specify a logical condition, a set of statements to trigger if the condition evaluates to true, and a set of statements to trigger if the condition evaluates to false. *Loop* is a type of statement in which it is possible to specify a logical condition and a set of statements that will be repeatedly executed while the condition evaluates to true.

A *Function Declaration* allows specifying functions with a name and a return type. Furthermore, function declarations are composed of variable declarations that play the role of input parameters, variable declarations that play the role of local variables, and an ordered set of statements which compose the body of the function. The specific restrictions on the structure of a statement which is conditional, or loop, or function declaration, are specified by means of OCL constraints.

All type of statements can be composed of *Expressions* which return a value. Literal, Variable and Meta-Model Element are expressions which can not be composed of other expressions. A *Literal* expression returns its associated value. A *Variable* expression returns the value of a specific variable. A *Meta-Model Element* expression returns an object which is an instance of a meta-model element. *Call* is another specialization of Expression which can be composed of other ordered expressions. Function and Operator are specializations of Call.

A *Function* call is an expression that returns the value which results from the execution of a specified function. In a similar way, an *Operator* call, (*Arithmetical* or *Logical*) is an expression that returns the value which results from the execution of a specified operator. Again in this case, the specific restrictions on the structure of the different types of expressions are specified by means of OCL constraints.

Even though there are relationships between elements of the Transformation package and elements of the Program package, the Program package is independent enough to be replaced, if necessary, by another Program package which represents other type of programs, for instance, declarative or functional programs.

Finally, the previously mentioned concept Meta-Model Element is defined in the MetaModel package, which is illustrated in Figure 5. In the context of UsiXML, a *Meta-Model Element* represents an element of any UsiXML meta-model published in [21]. The meta-model element is composed by a set of *Attributes*. Furthermore, *Relationships* can be established among meta-model elements.

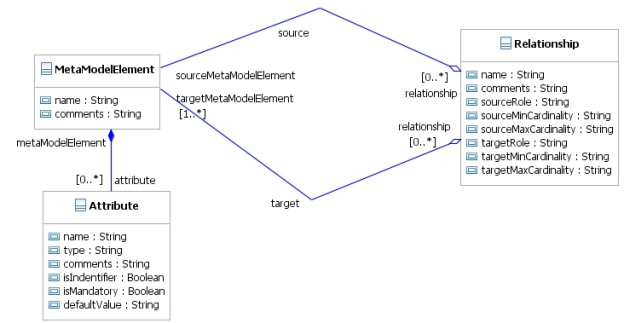


Figure 5. MetaModel package of the new proposed transformation meta-model.

AN ILLUSTRATIVE EXAMPLE

As an example, we are going to explain how the proposed meta-model can store transformation rules independently of the language used to carry out the transformations. Once the transformation is expressed according to the meta-model, all this information can be translated into a specific transformation language, such as the graph notation or ATL.

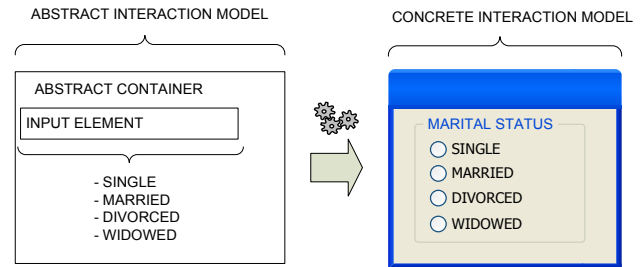


Figure 6. Example to instantiate the transformation meta-model.

The goal of the example is to transform from the Task and Domain level to the CUI level. This aim is divided into two transformation steps: a transformation from the Task and Domain level to the AUI level, and a transformation from AUI level to the CUI level. For space reasons and to simplify the example, we focus on the last step. More specifically, we focus on a transformation rule to transform an input element with restricted values of an AUI model to a list of radio buttons in a CUI model, as Figure 6 shows. This transformation is useful, for instance, in a form where the user has to provide his/her marital status. Possible values are only: single, married, divorced and widowed.

Firstly, we are going to show how different instances of classes of our proposed meta-model store the required information to perform the transformation of the example.

- **TransformationModel:** it includes the set of transformation rules necessary to perform a transformation from the Task and Domain level to the CUI level. The attributes have the following values: *initialAbstractionLevel*: Task and Domain; *finalAbstractionLevel*: CUI.

- **ContextModel**: initial and final contexts are the same, there is no change of context.
- **TransformationStep**: in the example, this class has two instances of type *Reification*. One instance to represent the transformation from the Task and Domain models to the AUI model and another from the AUI model to the CUI model. With regard to the last instance, the attributes are: *id*: AUItoCUI; *name*: From AUI to CUI; *sourceAbstractionLevel*: AUI; *targetAbstractionLevel*: CUI.
- **TransformationSystem**: according to [10], the goals that need to be achieved when transforming from AUI to graphical CUIs are: 1) reification of abstract containers into concrete containers; 2) selection of concrete individual components; 3) arrangement of concrete individual components; 4) navigation definition; 5) concrete dialog control definition; and 6) derivation of CUI to domain mappings. Each one of these goals will have a corresponding transformation system instance. Our specific rule to transform an input element to a list of radio buttons is included in the second goal, selection of concrete individual components. The attributes of this transformation system instance have the following values: *id*: 4333e; *name*: Selection of concrete individual components; *goal*: Select the concrete individual components to be used; *description*: transforms abstract individual components to concrete individual components.
- **TransformationRule**: this instance stores the rule to transform an input element to a list of radio buttons. The values of its attributes are: *id*: Rdbtt1; *name*: Input element to radio button; *description*: transforms an input element with a limited number of possible values into a list of radio buttons.
- **Condition**: only input elements with a set of limited possible values can be transformed into radio buttons. In the example, this instance is related to a logical expression that verifies if the input element has a set of limited possible values.
- **Source**: this instance is used to specify which element of the AUI model is going to be transformed. In this example, it is an input element from an abstract container.
- **Target**: this instance stores a program that specifies how to build the target element of the transformation. In this example, it is a program that builds a list of radio buttons of the CUI model.
- **MetaModelElement**: instances of this class represent all the classes of the AUI meta-model and the CUI meta-model involved in the transformation. Classes affected of the AUI meta-model are: *Abstract Container*; *Abstract Individual Component*; *Facet*; *Input*; *Selection Value*. With regard to the CUI meta-model,

affected classes are: *Window*; *Tabbed Dialog Box*; *Radio Button*.

- **Attribute**: each instance of this class represents an attribute of the classes belonging to the AUI meta-model or to the CUI meta-model included in the transformation. For example, the class *Selection Value* of the AUI meta-model has an attribute called *name*.
- **Relationship**: instances of this class store the relationships among classes of the AUI meta-model or the CUI meta-model included in the transformation. For example, the class *Input* is related to the class *Selection Value* in the AUI meta-model.
- **Program**: this instance represents the set of statements of the transformation process that are used to reach the target.
- **Statement**: this class is specialized in other ones depending on the type of statement. In our example, there are assignments to perform the transformation from an input element to a *RadioButton*.
- **Expression**: this class is specialized in other classes to represent sentences that return a value. For example, there is an instance of the class *MetaModelElement* for each element of the CUI model created in this transformation.

All the information stored in a transformation model compliant with our proposed transformation meta-model can be translated into an existing transformation language to perform the transformation, such as, the graph notation or ATL.

Regarding the graph notation, from the information available in the previously explained transformation model example, we could automatically generate the corresponding graph-based transformation rules defined in [19]. Figure 7 illustrates the NAC of the transformation rule. Figure 8 illustrates the LHS, and Figure 9 illustrates the RHS. (In Figure 7, 8, and 9, the red part corresponds to the transformation to a graphical CUI model).

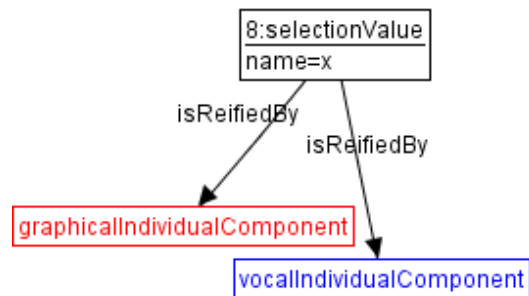


Figure 7. NAC statement to transform an input element to a radio button with possible options [19].

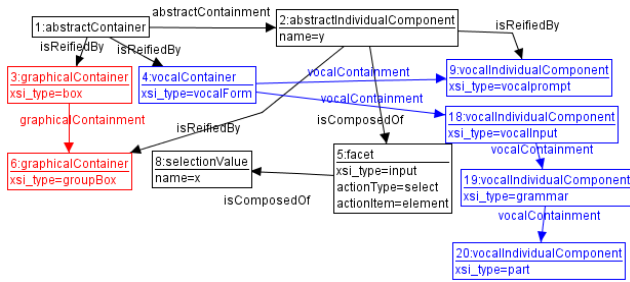


Figure 8. LHS statement to transform an input element to a radio button with possible options [19].

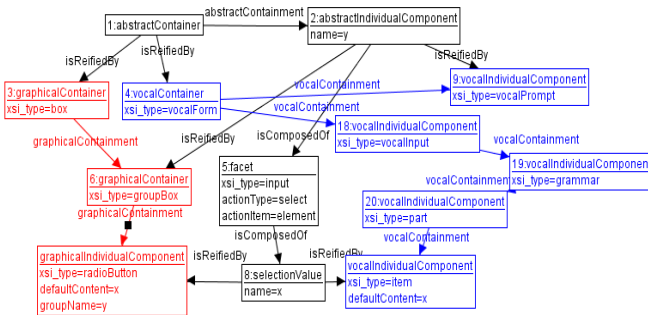


Figure 9. RHS statement to transform an input element to a radio button with possible options [19].

From the information available in the previously explained transformation model example, we could also automatically generate the following corresponding ATL code:

```

module TransformationRadioButton;
create OUT: CRadioButton from IN: AInoutElement;
uses strings;
rule container2window {
    from
    c:AInoutElement!abstractContainer
    to
    out: CRadioButton!window (
        title <- c.title)
}
rule input2TabbedDialogBoxes {
    from
    ie:AInoutElement!Input(ie.type.ocIsKindOf(AInoutElement!SelectionValue))
    to
    out: CRadioButton!TabbedDialogBoxes(
        name <- ie.name)
}
rule selectionValue2RadioButton {
    from
    sv : AInoutElement!SelectionValue (
        sv.type.ocIsKindOf(AInoutElement!Input))
    to
    out: CRadioButton!RadioButton(
        value <- sv.name)
}

```

This example suggests that our proposed meta-model is abstract enough to represent concepts of transformation languages as dissimilar as ATL and graphs. Analysts can put all their efforts on specifying the transformation rules independently of transformation language details, and in a next step, these rules can be translated into a specific transformation language that performs the transformation automatically.

CONCLUSION

This paper proposes a new transformation meta-model independent of graph transformations for a user interface description language called UsiXML. The aim is to provide a transformational approach independent of the specific syntax and notations of transformation languages or technologies. This paper is a first step to reach the aim. Once the transformation meta-model has been defined, a precise syntax must be specified to write transformation rules. The transformation meta-model expresses only how to store transformation rules, but the analyst must have an unambiguous language to represent those rules. This syntax is going to be defined as a future work.

Furthermore, once the transformation meta-model supports all type of transformations and a precise syntax has been defined to specify them, the last step will be to specify different translators. These translators must translate all the information stored in the transformation model into a transformation language selected by the analyst. Each transformation technology will have a specific translator that performs the translation automatically taking as input a transformation model.

The paper has exemplified how a transformation model stores the required information to translate transformation rules into a graph-based notation or ATL. As future work, we are going to test with other transformation technologies such as XSLT or CWM. Moreover, we have not tested the transformation meta-model to assess whether or not it supports model-to-code transformations. This kind of transformations is very common in UsiXML because final user interfaces are automatically generated from CUI models. Hence, it would be very interesting that analysts could also specify this type of transformations by means of the transformation model.

ACKNOWLEDGMENTS

This work has been developed with the support of the ITEA2 Call3 UsiXML project under reference 2008026. It also has the support of MICINN under the project SESAMO (TIN2007-62894) and has been co-financed by ERDF.

REFERENCES

1. Assmann, U.: How to uniformly specify program analysis and transformation with graph rewrite systems. 6th International Conference on Compiler Construction, Vol. LNCS 1060. Springer Berlin / Heidelberg (1996).
2. ATL: <http://www.eclipse.org/m2m/atl/>. Last visit: March 2010

3. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J. A Unifying Reference Framework for multi-target user interfaces. *Interacting with Computers*, 15(3):289–308, 2003.
4. Caplat, G., Sourrouille, J.L.: Model Mapping in MDA. Workshop in Software Model Eng. (WISME) (2002).
5. CWM Partners. Common Warehouse Metamodel (CWM) Specification. OMG Documents: ad/01-02-{01,02,03}, Feb. 2001.
6. Gogolla, M.: Graph Transformations on the UML Metamodel. ICALP Workshop Graph Transformations and Visual Modelling Techniques, Waterloo, Canada (2000).
7. Ho, W.M., Jézéquel, J.-M., Guennec, A.L., Pennaneach, F.: UMLAUT: an Extendible UML Transformation Framework. In: Tyugu, R.J.H.a.E. (ed.): 14th IEEE International Conference on Automated Software Engineering, ASE'99. IEEE (1999).
8. Karsai, G., Agrawal, A., Shi, F., Sprinkle, J.: On the use of graph transformation in the formal specification of model interpreters. *Journal of Universal Computer Science*, Vol. 9 (2003) 1296–1321.
9. Limbourg, Q., Vanderdonckt, J., Multi-Path Transformational Development of User Interfaces with Graph Transformations. In Seffah, A., Vanderdonckt, J., Desmarais, M. (eds.), “Human-Centered Software Engineering”, Chapter 6, HCI Series, Springer, London, 2009, pp. 109-140.
10. Limbourg, Q. *Multi-Path Development of User Interfaces*. PhD thesis, Université catholique de Louvain, November 2004.
11. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Lopez, V. UsiXML: a Language Supporting Multi-Path Development of User Interfaces. In *Proc. of 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems EHCI-DSVIS'2004* (Hamburg, July 11-13, 2004). Lecture Notes in Computer Science, Vol. 3425, Springer-Verlag, Berlin, 2005, pp. 200-220..
12. Mellor, S.J., Clark, A.N., Futagami, T.: Guest Editors' Introduction: Model-Driven Development. *IEEE Software*, Vol. 20 (2003) 14-18.
13. Mens, T., Gorp, P.V.: A Taxonomy of Model Transformations. *Language Engineering for Model-Driven Software Development - Dagstuhl Seminar Proceedings*, Dagstuhl, Germany (2005).
14. MOF: <http://www.omg.org/spec/MOF/2.0/>. Last visit: March 2010.
15. Pastor, O., Molina, J.: *Model-Driven Architecture in Practice*. Springer, Valencia (2007)
16. Peltier, M., Bézivin, J., Guillaume, G.: MTRANS: A general framework based on XSLT for model transformations. Workshop on Transformations in UML, WTUML'01, Genova, Italy (2001).
17. Pérez-Medina, J. L., Dupuy-Chessa, S., and Front, A. A Survey of Model Driven Engineering Tools for User Interface Design. In M. Winckler, H. Johnson, and P. A. Palanque, editors, *TAMODIA*, volume 4849 of *Lecture Notes in Computer Science*, pages 84–97. Springer, 2007.
18. QVT: <http://www.omg.org/spec/QVT/1.0/>. Last visit: March 2010.
19. Stanciu, A. *A Methodology for Developing Multimodal User Interfaces of Information Systems*. PhD thesis, Université catholique de Louvain, June 2008.
20. Stanciu, A., Limbourg, Q., Vanderdonckt, J., Michotte, B., Montero, F., A Transformational Approach for Multimodal Web User Interfaces based on UsiXML. In *Proc. of 7th Int. Conf. on Multimodal Interfaces ICMI'2005* (Trento, 4-6 October 2005), ACM Press, New York, 2005, pp. 259-266.
21. Université catholique de Louvain. UsiXML V1.8 Reference Manual. 2008. Available at: <http://www.usixml.org/index.php?mod=pages&id=5>. Last visit: March 2010.
22. Varró, G., Schürr, A., Varró, D.: Benchmarking for graph transformation. *IEEE Symposium on Visual Languages (VL/HCC 2005)*, University of Texas at Dallas (2005).
23. XSLT Specification: <http://www.w3.org/TR/xslt>. Last visit: March 2010.