UsiComp: an Extensible Model-Driven Composer

Alfonso García Frey[1], Eric Céret[2], Sophie Dupuy-Chessa[3], Gaëlle Calvary[4], Yoann Gabillon[5]
UJF[1,2], UPMF[3], Grenoble INP[4], CNRS[1,2,3,4], LIG[1,2,3,4], UVHC[5]
41 rue des mathmatiques, 38400 Saint Martin d'Hres, France[1,2,3,4]
Univ. Lille Nord de France, F-59000 Lille, France, LAMIH, F-59313 Valenciennes, France[5]
{Alfonso.Garcia-Frey, Eric.Ceret, Sophie.Dupuy, Gaelle.Calvary}@imag.fr,
yoann.gabillon@univ-valenciennes.fr

ABSTRACT
Modern User Interfaces need to dynamically adapt to their context of use, i.e. mainly to the changes that occur in the environment or in the platform. Model-Driven Engineering offers powerful solutions to handle the design and the implementation of such UIs. However this approach requires the creation of an important amount of models and transformations, each of them in turn requiring specific knowledge and competencies. This leads to the need of an adapted tool sustaining the designers' work.

This paper introduces UsiComp, an integrated and open framework that allows designers to create models and modify them at design time as well as at runtime. UsiComp relies on a service-based architecture. It offers two modules, for design and execution. The implementation has been made using OSGi services offering dynamic possibilities for using and extending the tool. This paper describes the architecture and shows the extension capacities of the framework through two running examples.

Author Keywords
Model-Driven Engineering; User Interfaces; Design Tools.

ACM Classification Keywords
H.5.2 Information Interfaces and Presentation: User Interfaces – theory and methods.: D.2.2.Software Engineering: Design Tools and Techniques – user interfaces.

General Terms
Design, Human Factors.

PROBLEM AND MOTIVATION
With the increasing amount of platforms and devices as well as of the new expectations of users, designers need to create User Interfaces (UIs) that are able to adapt to their context of use, i.e. to the changes that occur in the environment, the platform and/or the user profile.

However, the huge amount of possible combinations of these context elements makes it no longer possible to anticipate and predefine all the eventual situations at design time. Systems have to be designed to be able to adapt themselves to their context of use while preserving usability [2]. Model Driven Engineering (MDE), which is based on the generation of applications from models, provides powerful solutions for the creation of such UIs. In this paradigm the models represent the different facets of the system to be created. These models are successively transformed and combined to finally generate the code. This opens possibilities like easier evolutions and reuse [6], dynamic adaptation to the context of use, greater quality, early detection of defects and inclusion of knowledge in executable models [11].

However, creating all the models and all the transformations for an application is a long and complicated work: the designer has to understand the underlying meta-models, write the models that conform to these meta-models and elaborate some transformations. Then, the designer needs to create a system that runs the transformations and generates the final code. This is why several studies have been driven to create frameworks to manage these stages and handle models at runtime.

This paper introduces UsiComp, a tool for creating a complete set of models (Tasks, Abstract UI, Concrete UI, Domain, Context, Mapping, Quality) and simplifying the creation of transformations. UsiComp's design and execution modules, relying on an extensible service-based architecture, include an easy graphical interface that offers an efficient way of creating models by drawing them or by combining predefined components, permitting fast prototyping possibilities. This makes it a powerful and innovative tool for designing a system with an MDE approach. This paper first relates the state of the art in the field. Then it presents UsiComp's architecture and exemplifies its extensibility on two running exemples.

RELATED WORKS
Several MDE frameworks have been proposed in academic or commercial projects. For instance, UsiXML [9] offers a rich set of tools like SketchiXML [3], IdealXML [12] or

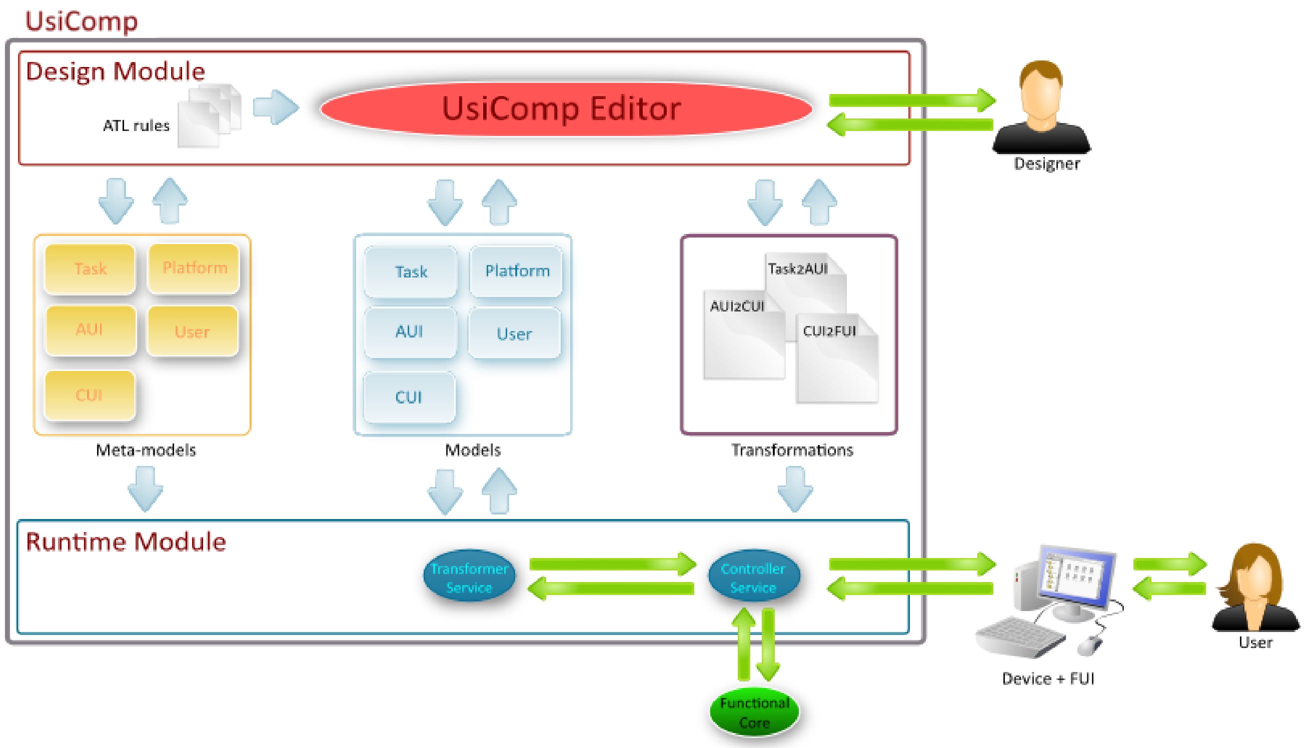


Figure 1. UsiComp software architecture: meta-models, models and transformations at the heart of both design time (IDE for designers) and runtime (FUIs for end-users).

GraphiXML [10]. This series of tools aim at covering the different phases of the development process while complying with the separation of concerns principle. Altogether, they support flexibility in the sense that they make it possible to forward and reverse engineer sketches as well as final UIs. However, in practice, effort still needs to be put on their methodological integration so that to guide the designer in selecting the right set of tools according to his needs and situation.

MARA [16] is both an architecture and a tool for managing UIs through models at runtime. Like UsiXML, MARA implements a transformation sequence starting from the Task model to successively generate the Abstract UI (AUI), Concrete UI (CUI) and Final UI (FUI). MARA supports the dynamic selection of models, meta-models and transformation. However, MARA neither includes a models editor nor does it support multiple entry points: the designer has to start with the Task model and then to implement all the predefined set of models.

Leonardi is a commercial software with two versions: a free version available on the Internet[1], and a retail version[2]. The free version requires that the designer starts from a mockup of the UI, while the full version adds the possibility of starting from a workflow. Leonardi offers a direct transformation of the UI prototype into code that is interpreted at runtime. The two versions rely on a fixed set of models with a proprietary format.

Blumendorf et al. [1] have also proposed a framework that makes it possible to manage and to extend the calls to the services of the functional core. However, the editor is the standard Eclipse Modeling Framework which offers poor specific help for designing models and transformations. In addition, the transformation sequence starts at the Task model only.

MARIA [15] introduces a rich event manager and supports design and runtime for creating applications that use the functionalities offered by Web services. These services are annotated with hints helping the generation of UIs. However, this language (and the associated tool) does not allow the designer to extend the set of models.

SOFTWARE ARCHITECTURE

The software architecture of UsiComp relies on services. The term service refers to “a set of related software functionalities that can be reused for different purposes, together with the policies that should control its usage” [18]. These services are implemented according to the OSGi specification [13]. The main service is the Controller Service (figure 1). The Controller Service is in charge of orchestrating the whole process in which a UI is generated by successive transformations. Transformations may be reifications or abstractions [2]. Reification (respectively Abstraction) lowers (respectively increases) the level of abstraction of a model. Currently, only

[1]`http://www.leonardi-free.org`
[2]`http://www.w4.eu`

reifications have been implemented and integrated into UsiComp. However, the architecture is fully generic, and so capable of integrating abstractions as well.

UsiComp (figure 1) is made of two modules: one for design, another one for runtime. They share common resources: meta-models, models and transformations.

Design Module

The design module includes a visual editor (figure 2) for designing and prototyping UIs. The UsiComp editor offers the following functionalities:

- It allows designers to define all the models and transformations needed to produce a UI. The UI of the UsiComp editor is divided into three different areas (figure 2): 1) a toolbar with the most common actions, 2) the workspace presenting graphical representations of the models, and 3) the right panel which provides access to the different elements of each meta-model. Designers can create models by picking up the needed components and combining them. For instance, figure 2 shows the UsiComp editor and three models with their respective transformations. The model at the top of the figure is a task model, represented with the CTT notation [14]. This task model is transformed into an AUI model represented with blue boxes. These blue boxes show different Abstract Interaction Units and their arrangement. The AUI model is in turn transformed into a graphical CUI model that UsiComp represents with a mock-up.

- Transformations between models are composed of rules. A rule specifies how one specific set of elements of a source model is transformed into a set of target model elements. Designers can select what rules they want to apply to a given model, and the system will automatically compose the resulting transformation. These rules are represented by arrows from the source element to the target. Most common rules are already available in the system, but designers are free to add other rules if needed. Transformations and rules are written in the Atlas Transformation Language (ATL) [8].

- The UsiComp editor verifies that the designed models comply with their corresponding meta-models. For instance, a binary operator in the task model must link two different tasks. The UsiComp editor also composes and compiles the transformations and rules thanks to an integrated ATL compiler.

- The resulting Final UI, which is the code of the UI, can be directly executed from the IDE (green play button on the toolbar) giving designers the opportunity to preview the generated UI.

Runtime Module

The UsiComp runtime infrastructure is built on OSGi services [13]. It works as follows:

- Once a new device becomes available to the framework (a specific client is installed into the device for this purpose), UsiComp identifies its specific platform model containing the platform details. The current version of UsiComp contains platform models specified by hand.

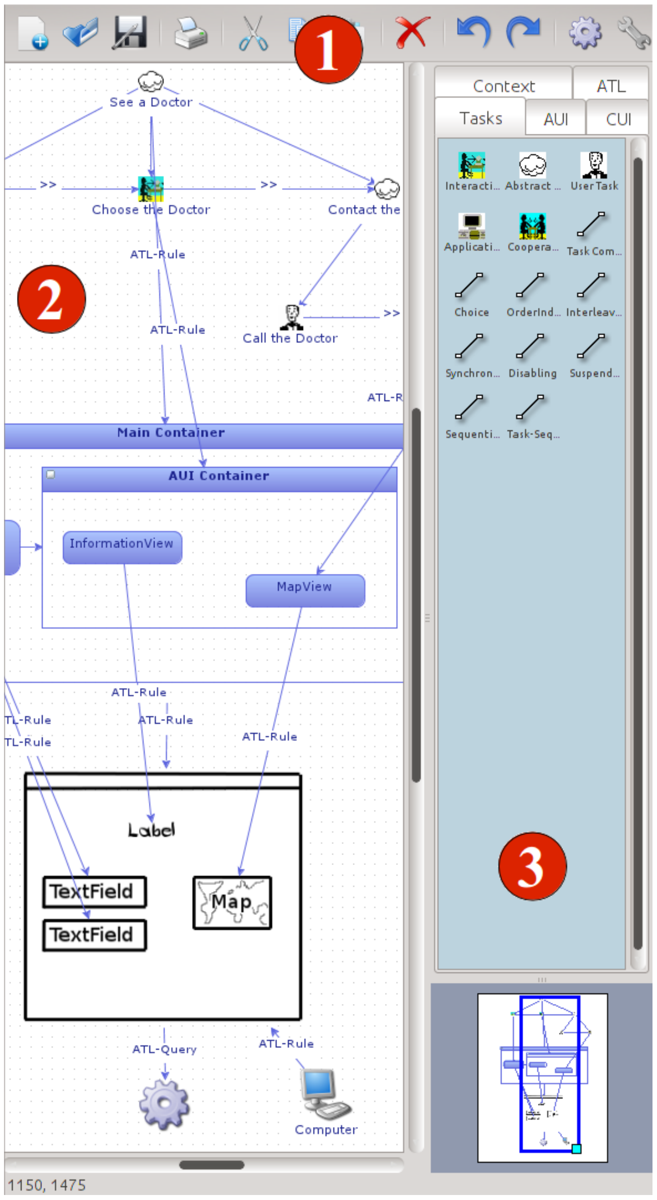


Figure 2. UsiComp Development Environment. From Top to Bottom: Task model, AUI model, CUI model. Transformations are represented by arrows.

- The Transformer Service (Figure 1) is a generic transformation service that can apply any transformation to any model or models, producing models or text as output.

- To produce the UI, the Controller Service manages the transformations, their order of execution and their related models and meta-models, calling to the Transformer Service as many times as needed. The platform model is considered in the transformation process to produce an adapted UI.

- In the transformation process, the Controller weaves the functional core of the application into the UI, embedding the calls from and to the UI.

The models, meta-models and transformations involved in the generation are directly accessed by the Controller Service, which is also responsible of linking the application logic from the functional core to the UI and viceversa.

UsiComp has been entirely implemented in Java, EMF [17] and ATL [8]. The development environment can be launched as a normal Desktop application or as a Web application embedded in an applet. Thanks to the OSGi services, it is possible to dynamically update the editor without stopping the application. For instance, updating a service or replacing the transformation language for another one can be dynamically achieved.

Code Generation

UsiComp currently supports the generation of Java code. The Java code is directly generated from CUI models with an ATL transformation. ATL supports not only model to model transformations, but also model to "primitive value" transformations. This last type of transformations is called queries. They can be used to generate text from models. In this particular case, the primitive value is a String data type containing all the generated code of the UI.

The code generation is directly done by transformation instead of using external tools for several reasons. First, most of the technologies that already exist focus on one language only (as for instance JaMoPP [7] for Java), or only one programming paradigm, mainly imperative in most of the cases. As the generated UI must be platform independent, the code generation cannot rely on only one specific language or paradigm. For instance, we would like to generate GTK UIs in the future for a functional language such as Haskell. Not all the languages and paradigms are supported by external generators, so integrate an external tool each time is not always possible.

Technically, the code generation is done by parsing the CUI model with a Depth First Search algorithm, i.e., translating the first element of the CUI model (at the top of the model, for instance, the main window) and exploring/transforming as far as possible along each branch before backtracking. This is possible because the CUI metamodel forces a free loops tree-like CUI models.

EXTENSION ABILITIES

UsiComp has extension abilities that are illustrated in the two following examples. Both extensions are summarized in figure 3. Figure 4 shows the classical transformation sequence and its related models and transformations, and how the examples extend UsiComp at different levels. These extensions are Compose for the task model generation and Balsamic Mockup for the CUI model generation. They are integrated at the Tasks and CUI levels respectively.

Example 1: Find a Doctor

This first example shows how to integrate an external and independent tool into UsiComp. The goal is to produce a Final

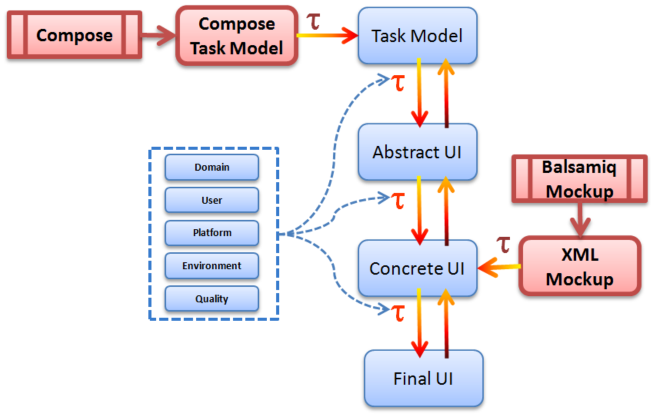


Figure 3. Two examples of the UsiComp extensibility.

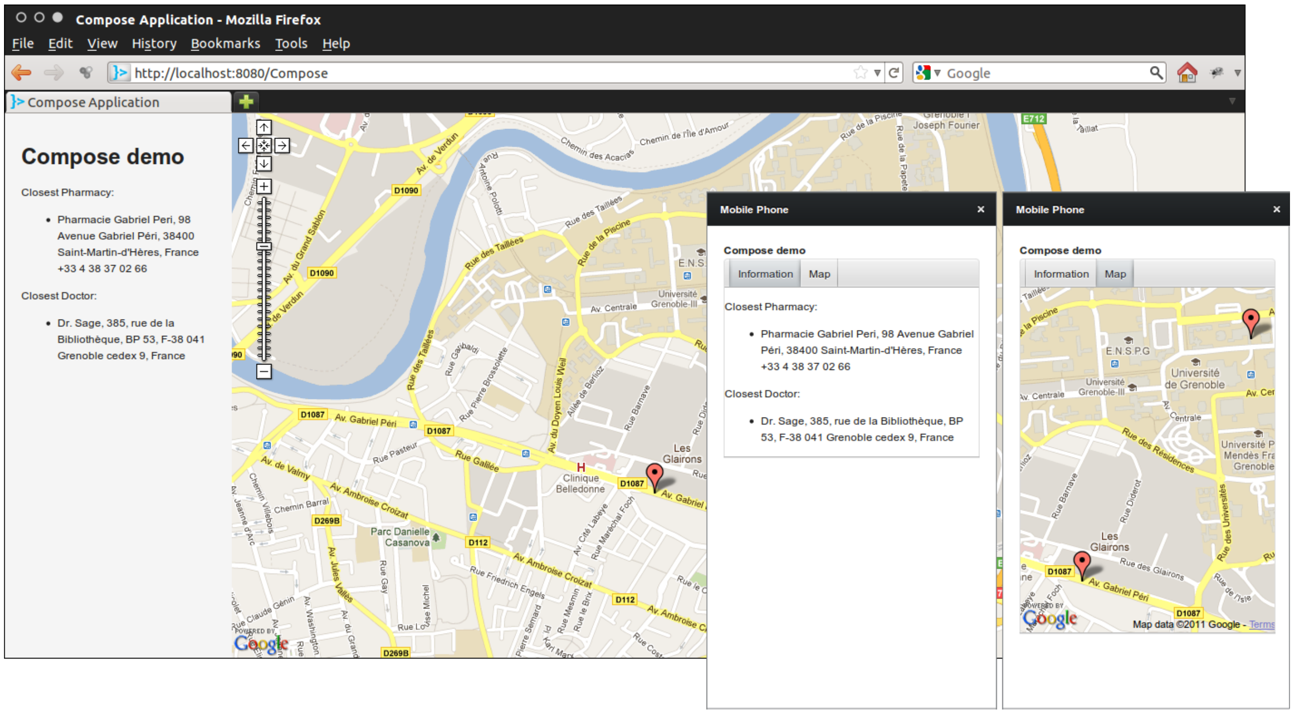
UI from classical UI models in a top-down transformation process, starting from the task model and ending with the generation of the code. To show the versatility of UsiComp we extend the tool with task model generation capabilities, i.e. the task model is not created by a designer at design time as done in traditional Model-Based Design approaches, but produced with another tool called Compose. Compose [4] is a framework that generates a task model according to a specific goal. This goal can be expressed by the end-user or, as in our case, by the designer. A video showing the global process and the generation of different user interfaces from the same goal is provided with this paper and available online[3].

The generation of the task model is made by automated planning algorithms [5]. Compose relies on a set of possible abstract or concrete actions described by predicates. The combination of these predicates make the goal achievable (or not). For instance, the action "find a possible route" with the help of a map needs an Internet access (described by the predicate "internet", which is a logical condition that can be true or false). When this action is executed, the map is displayed if a map widget is available (described by the predicate "map").

The output of Compose is a task model that fulfils the goal. This task model uses a specific Compose notation. To generate a UI for this task model, we first need to integrate the model into the UsiComp transformation chain. To this end, we have written an ATL transformation rule that converts the task model produced by Compose from its specific format to the UsiXML format, which is understandable by UsiComp.

The UsiComp editor produces UIs that are adapted to different platforms. These platforms are described through a platform model, which includes platform features such as the number of screens, their resolutions, the operating system of the platform, the available technologies and their versions, etc. For instance, if we want to generate Java code for the UI we would like to know if there is a Java support on the target platform, and what the current version of the available Java

[3]http://youtu.be/Q_Ub3XHQxck


Compose Application - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/Compose
Google
Compose Application
Mobile Phone
Mobile Phone
Information
Map
Information
Map

Compose demo

Closest Pharmacy:

- Pharmacie Gabriel Peri, 98 Avenue Gabriel Péri, 38400 Saint-Martin-d'Hères, France +33 4 38 37 02 66

Closest Doctor:

- Dr. Sage, 385, rue de la Bibliothèque, BP 53, F-38 041 Grenoble cedex 9, France

Figure 4. Two UIs generated from the same task model. The UI in the background has been generated for a PC screen with higher resolution that the UI for the mobile phone in front of the figure.

Virtual Machine is. Two different platform models are currently available in the tool, a PC platform and a mobile phone platform. The video shows how to use the editor for changing from one platform to another, and (re)generate the UI in a few clicks. Figure 4 shows the two different UIs that are produced for these two platforms. In the background, we can see the UI adapted to the screen of the PC platform. In the front, the two tabs of the generated UI for the mobile platform are shown. Some of the adaptations being performed in the process are also visible. Among others, the original screen from the PC platform has been split into two tabs due to the small resolution of the mobile phone screen. The zoom controller of the map widget has been removed as well.

Example 2: Balsamiq Mockups

The UsiComp classical transformation sequence has also been extended to integrate a UI mockup at the CUI level. This possibility refers to the situation in which a designer has a very precise idea of the UI he wants to implement. Indeed, the successive transformations do not guaranty that the resulting UI will match the precise picture of the designer. With this extension, the designer can draw a mockup of the desired UI using an external tool, like Balsamiq Mockup. Balsamiq Mockup[4] is an online tool offering a sketching service for UI prototyping. The UI produced with Balsamiq Mockup can be exported to a XML file. This file is integrated into UsiComp transformation sequence and converted into a CUI via a supplementary transformation (see Code 1 for an excerpt). In order to indicate where this CUI has to be considered in the transformation process, a task node is decorated to indicate that it will be defined at a more concrete level.

```
-- Concrete Containers

rule Control2Window {
  from
    s : BalsamiqModel!Control (
        s.isAWindow()
    )
  to
    t : CUI!Window (
        Title <- s.controlProperties.text.regexReplaceAll('%20', ' '),
        WindowComposedOfMenuBar <- s.WindowComposedOfMenuBar(),
        WindowComposedOfPanels <- s.WindowComposedOfPanels()
  )
}

rule Control2MenuBar {
  from
    s : BalsamiqModel!Control (
        s.isAMenuBar()
    )
    to t : CUI!MenuBar (
        MenuBarComposedOfButtons <- s.MenuBarComposedOfButtons()
        ->collect(e | thisModule.String2Button(e))
    )
}
```

Code 1. Excerpt of the ATL transformation from Balsamiq Mockup to the CUI.

CONCLUSIONS AND PERSPECTIVES

This paper presents UsiComp, a UI development and execution environment based on UsiXML. The underlying architecture is composed of a design module which includes an integrated editor for designing purposes, and a runtime

[4]`http://www.balsamiq.com`

module, responsible for generating the UI, weaving it with the functional core, and keeping the target platform in the loop. The open architecture of UsiComp allows us to integrate new (meta-)models and transformations, such as the Compose task model. This model is integrated into the UsiComp generation process via a supplementary transformation.

In future work we plan to add more (meta-)models to UsiComp, including one Transformation meta-model so we can model transformations as well. We plan to add more transformation rules in two specific areas of UsiComp. First, enriching the current repository of transformations from the task model to AUI and from AUI to CUI will help the management of sophisticated UIs. Second, improving the code generation for supporting new languages and programming paradigms.

We are planning to evaluate UsiComp with an industrial project implementation in order to measure the usefulness, the ease of use and the completeness of the tool. The usefulness will be measured according to the possibilities for designers to produce the expected UIs. The ease of use and completeness will be evaluated with questionnaires about users' satisfaction.

ACKNOWLEDGMENTS

This work is funded by the european ITEA UsiXML project.

REFERENCES

1. Blumendorf, M., Lehmann, G., Feuerstack, S., and Albayrak, S. Executable models for human-computer interaction. In *Interactive Systems. Design, Specification, and Verification*, T. Graham and P. Palanque, Eds., vol. 5136 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2008, 238–251.
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), 289–308.
3. Coyette, A., and Vanderdonckt, J. A sketching tool for designing anyuser, anyplatform, anywhere user interfaces. In *Human-Computer Interaction - INTERACT 2005*, M. Costabile and F. Patern, Eds., vol. 3585 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2005, 550–564. 10.1007/11555261_45.
4. Gabillon, Y., Petit, M., Calvary, G., and Fiorino, H. Automated planning for user interface composition. In *Proceedings of the 2nd International Workshop on Semantic Models for Adaptive Interactive Systems: SEMAIS'11 at IUI 2011 conference*, Springer HCI (2011).
5. Ghallab, M., Nau, D. S., and Traverso, P. *Automated planning - theory and practice*. Elsevier, 2004.
6. Hamid, B., Radermacher, A., Lanusse, A., Jouvray, C., Gérard, S., and Terrier, F. Designing Fault-Tolerant component based applications with a model driven approach. In *SEUS* (2008), 9–20.
7. Heidenreich, F., Johannes, J., Seifert, M., and Wende, C. Closing the gap between modelling and java. In *Software Language Engineering*, M. van den Brand, D. Gaševic, and J. Gray, Eds., vol. 5969 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2010, 374–383.
8. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I., and Valduriez, P. Atl: a qvt-like transformation language. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, OOPSLA '06, ACM (New York, NY, USA, 2006), 719–720.
9. Limbourg, Q., and Vanderdonckt, J. USIXML: a user interface description language supporting multiple levels of independence. In *ICWE Workshops* (2004), 325–338.
10. Michotte, B., and Vanderdonckt, J. GrafiXML, a multi-target user interface builder based on UsiXML. In *ICAS* (2008), 15–22.
11. Mohagheghi, P., Fernández, M. A., Martell, J. A., Fritzsche, M., and Gilani, W. *MDE Adoption in Industry: Challenges and Success Criteria*. 2008.
12. Montero, F., and López-Jaquero, V. Idealxml: An interaction design tool. In *Computer-Aided Design of User Interfaces V*, G. Calvary, C. Pribeanu, G. Santucci, and J. Vanderdonckt, Eds. Springer Netherlands, 2007, 245–252.
13. OSGi Alliance. OSGi Service Platform Release 4. [Online]. Available: http://www.osgi.org/Main/HomePage. [Accessed: Mar. 20, 2012], 2007.
14. Paterno, F., Mancini, C., and Meniconi, S. ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. In *INTERACT '97: Proceedings of the IFIP TC13 Interantional Conference on Human-Computer Interaction*, Chapman & Hall, Ltd. (London, UK, UK, 1997), 362–369.
15. Paternò, F., Santoro, C., and Spano, L. D. Maria: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. *ACM Trans. Comput.-Hum. Interact. 16*, 4 (Nov. 2009), 19:1–19:30.
16. Sottet, J.-S., Calvary, G., Coutaz, J., and Favre, J.-M. A model-driven engineering approach for the usability of plastic user interfaces. In *Proc. of EIS '08*, Springer-Verlag (2008), 140–157.
17. Steinberg, D., Budinsky, F., Paternostro, M., and Merks, E. *EMF: Eclipse Modeling Framework (2nd Edition)*, 2 ed. Addison-Wesley Professional, Dec. 2008.
18. Wikipedia. Service (Systems Architecture) - Wikipedia. [Online]. Available: http://en.wikipedia.org/wiki/Service_(systems_architecture). [Accessed: Mar. 20, 2012], 2012.