

Model-Based Software Development and Usability Testing

Peter Forbrig, Gregor Buchholz, Anke Dittmar, Andreas Wolff, Daniel Reichart

University of Rostock, Department of Computer Science, Albert-Einstein-Str. 21
18051 Rostock, Germany
pforbrig@informatik.uni-rostock.de
<http://wwwswt.informatik.uni-rostock.de/>

Abstract. In this paper we discuss a user-centered approach of developing software, which is based on models and allows bridging the gap between software engineering and usability engineering. The development process consists of a sequence of interactive model transformations. It is demonstrated how first prototypes of interactive systems, which are animated models, can help to capture requirements and how the models evolve to the final interactive system. This model-based approach provides excellent opportunities for testing the usability of software based on the tasks users have to perform. A tool is presented, which visualizes the activities of a test person based on the models. The tool supports remote testing, which can be performed even on mobile devices.

1 Introduction

It is commonly accepted that software development has to start with an analysis of the problem domain users work in. There are some discussions whether one has to start to analyze objects, tasks or interactions first but at the end there is a common understanding of the importance of all aspects of the problem domain. It is also more and more accepted that the point of view of the user is most important for the software under development. Indeed, a user-centered development process supports this idea perfectly.

Model-based development of software systems becomes more and more popular. Even if it is up to now not used very extensively it is an attractive process with proven record of success, especially in the context of developing multiple user interfaces. There are approaches focusing on object models first like the model-driven architecture of UML [24]. However, we follow task-based approaches like ADEPT [26], CTTE [4] or Cameleon [1]. Typically, such systems are used to model existing or envisioned tasks. They help to understand the tasks a user has to perform in more detail by allowing simulations. Additionally, systems like TERESA [23] support the development of user interfaces.

Our System DiaTask [21] follows a similar approach. Based on task-, object-, user- and environment-models interactive systems are developed. Chapter 2 will

describe this approach in a little bit more details. Afterwards opportunities for remote usability testing will be sketched and at the end we will discuss achieved and further goals.

2 Model-Based Development of Interactive Systems

We strongly believe that software engineers and user interface designers have to base their work on the same models. Fig. 1 visualizes this fact by presenting these models on the left hand side. Task-, business-object-, user- and device models are very much interrelated. The device model is a representative of a general environment model here.

These models are as well the basis for the development of the software developed by software engineers as those for the software developed by user-interface experts. Furthermore, we consider software development as a sequence of transformations of models that is not performed in a fully automated way but by humans using interactive tools.

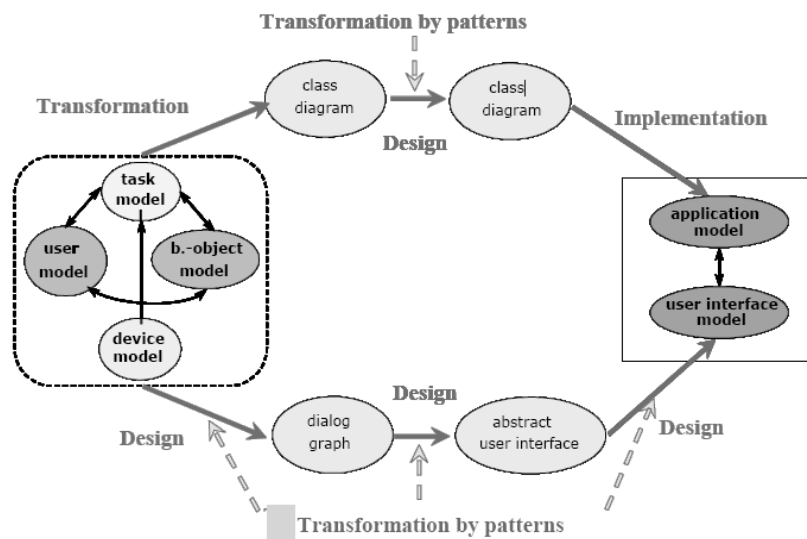


Fig. 1. Model-based development process model

Even if our work is especially focused on methods and tools supporting transformations by patterns (The transformation of class diagrams by patterns using the case tool Rational Rose is described in [11]. In [22], the idea of supporting the development of task models by patterns is presented.) we want to focus this paper on the models, which are within the development process.

In the following, we demonstrate the application of our ideas to a very small example of developing a mail management system. Fig. 2 is the result of the interviews of for the same system. It demonstrates how task models look like.

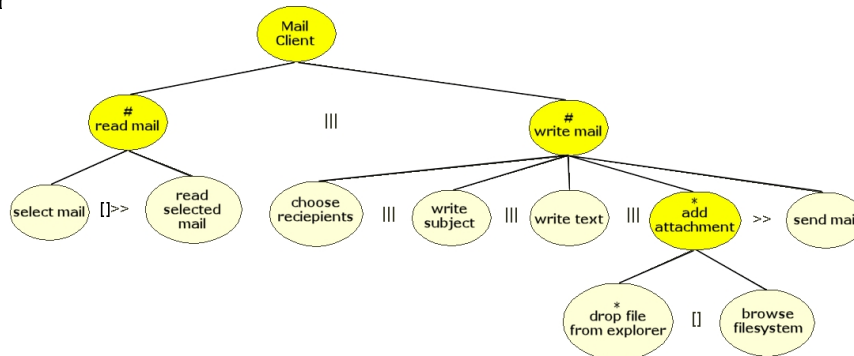


Fig. 2. Task model for the mail management system

According to the task model of Fig. 2, a user may either read his mail, or write a new one. To read his mails, he has to select a specific mail from a list that is generated and presented to him by the application. Once he has selected a mail, he gets its content displayed. “Select” and “display” are consecutive subtasks of an iterative task that can be cancelled at any time.

Writing mails is modeled in a similar manner. After a user decides to write a mail he has to enter the iterative task produce mail, where he is requested to compose a new mail and, after finishing this, the application will send it away. This sub-task may also be cancelled at any time.

Additionally to the classical temporal operations like $>>$ - enabling, $|||$ - in parallel, $[]$ - alternative a new operation symbol $\#$ is introduced. It represents the “instance iteration” operation. In contrast to the classical iteration $*$ it allows to start a new iteration before the old one is finished, a specification feature, which is very helpful in a lot of applications.

With one of our tools called DiaTask [21] we are able to develop dialog graphs that represent the navigation structure of the interactive systems. Such a graph is based on the previously specified task model.

A dialog graph consists of a set of nodes, which are called *views* and a set of transitions. There are 5 types of views: single, multi, modal, complex, and end views. A single view is an abstraction of a single sub-dialog of the user interface that has to be described. A multi view serves to specify a set of similar sub-dialogs. A modal view specifies a sub-dialog, which has to be finished in order to continue other sub-dialogs of the system. Complex views allow a hierarchical description of a user interface model. End views are final points in (sub-) dialogs. Each view is characterized by a set of (navigational) elements. A transition is a directed relation between an element of a view and another view. Transitions reflect navigational aspects of user interfaces. It is distinguished between sequential and concurrent transitions. A sequential

transition from view v1 to view v2 closes the sub-dialog described by v1 and activates the sub-dialog, which corresponds to v2. In contrast, v1 remains open while v2 is activated if v1 and v2 are connected by a concurrent transition. Fig. 3 shows the graphical notation for the different types of views and transitions.

Unlike TERESA [23] the dialog graph is the result of a design process and not the result of automatic transformations.

DiaTask allows assigning several different dialog graphs to one task model. Fig. 3 demonstrates the example of our mailing system with single views (main window, write mail), multiple views (read mail) and the end view (end). The screenshot is produced using the eclipse [9] plug-in for DiaTask.

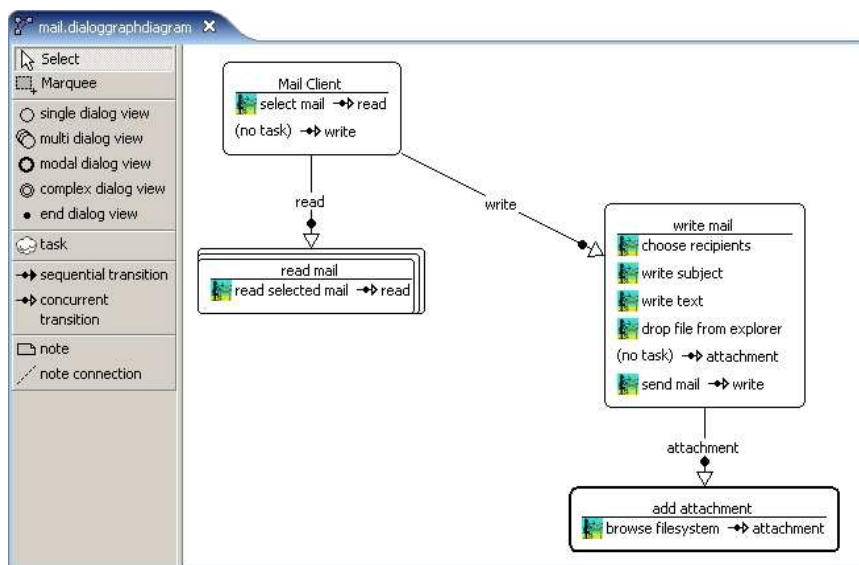


Fig. 3. Dialog graph for the management system of mails

There are concurrent transitions from “mail client” to “read mail” and “write mail”. This means that “main window” stays open and can be activated by a mouse click.

Multiple views are able to instantiate several instances. In the example of a mail system this means that users can read several mails in parallel. Multiple views go together with concurrent transitions. Fig. 3 gives an impression of the abstract prototype generated from the model of Fig. 2. It captures a situation where one mail is read and another one is written. The “main window” is active and as well pressing the corresponding button can activate “end”, “read mail” or “write mail”.

Fig. 4 demonstrates an abstract user interface.

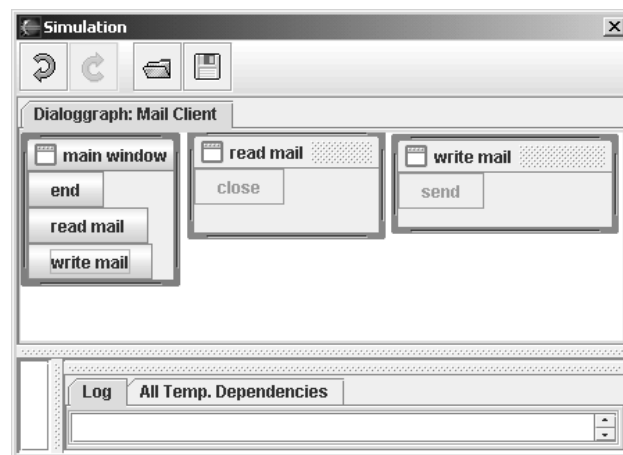


Fig. 4. Canonical abstract prototype of

Fig. 3 in animated mode

The abstract user interface of

Fig. 4 is automatically generated from the specification of the dialog graph and behaves according to the temporal relations of the task model. It is already very good to improve the communication with users during the requirements analysis phase. Unfortunately it has the drawback of a very abstract interface. It was our intention to improve this situation. It was our idea to generate the abstract user interface (e.g.

Fig. 4) in a language, which can be picked up by a GUI-editor for improvements. Our decision was to use XUL [11] for this purpose. This user-interface description language was introduced with the Mozilla project and is part of its success story. Based on an existing GUI plug-in for eclipse an editor for XUL was developed. This editor is able to replace existing elements by other ones. In this way the abstract user interface can be improved to a more useful one while keeping up the reference from the GUI elements to tasks. We are especially working on the problem how patterns can be used for this purpose. Possible tool support is discussed in [22].

The interpreter of the models, which controls the animation, recognizes the existence of improved windows and includes them into the animation process. In this way, the user is able to have a look at a user interface, which is already a candidate for the final interactive system.

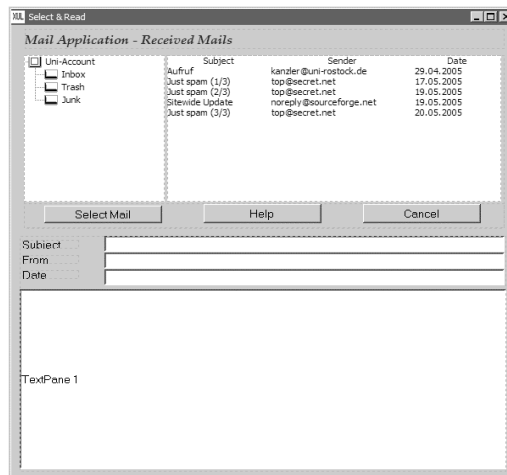


Fig. 5. Designed GUI for Select & Read

Details of the GUI-editor can be found in [27] and [28]. During simulation our system offers a view on the prototype of the user interface and a view on the animated task model. Fig. 6 demonstrates how the animated task model is visualized. Basic tasks (represented by squares) with green circles can be executed. Red crosses represent a status of the task that allows no execution because of restrictions (e.g. temporal relations between tasks). A blue tick signals the successful execution of a task.

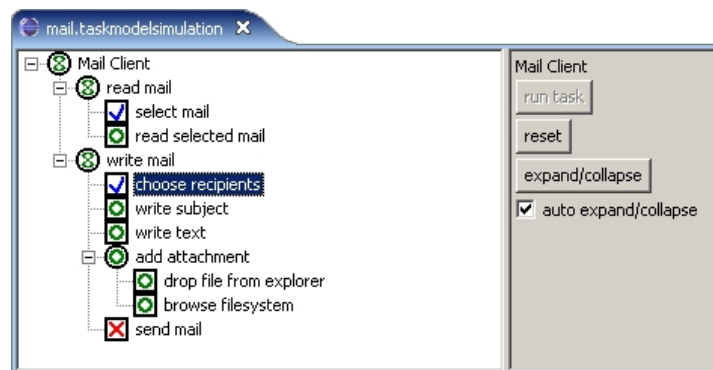


Fig. 6. Visualization of an animated task model

3 Remote Usability Testing

Usability testing is often the first activity, where software engineers come into contact with usability engineers. It is our idea that they start their co-operation already by commonly designing models.

Nevertheless, testing the usability of software is a time consuming process. Test scenarios have to be developed, test persons have to be hired and experienced usability experts have to observe the behavior of the test person. Sometimes it is very difficult to have test person and the usability expert as test supervisor at the same time at the same place. This is especially true for mobile applications.

Sometimes the presence of the test supervisors influences the test persons by executing their tasks.

These problems can be reduced by remote usability tests. This kind of test allows the test person and the test supervisor to work at different places. This is not new but it opens new opportunities based on our model-based development approach.

3.1 Software Architecture

It is possible to run a model-based system on client-server architectures. In this way models are interpreted on servers and the results are delivered to the clients. Fig. 7 gives an impression how this architecture looks like.

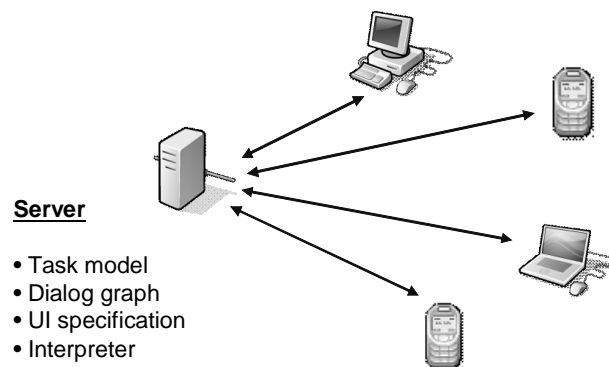


Fig. 7. Software architecture

The architecture of Fig. 7 opens new opportunities for remote usability testing because in addition to videos and the capturing of screens it is possible to observe the status of the interpreted models. Some features of our TMSClient and TMServer (TMS = Task Model System) will be discussed within the following paragraph.

3.2 Tool Support for Remote Usability Tests

Based on our model-based approach a TMSServer was implemented, which allows first remote usability tests. Up to now, the system's input is a generated Java-application (see Fig. 8) that was built using the transformations described in the previous paragraph.

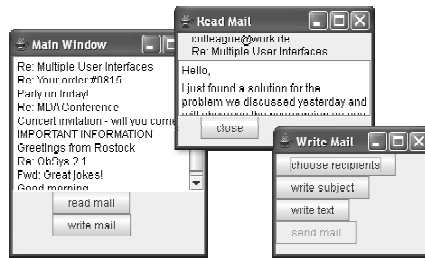


Fig. 8. User interface for the test person

Model related events (creations, state changes) are captured on client side and transmitted to the server where the model is reconstructed and sent to the supervisor's client(s). The usability experts get an impression of the test by looking at the visualization of the actual state of the execution of the task-model instance, which is presented in Fig. 9. We do not want to comment to all the information presented but would like to draw the attention of the reader to the left side of the screen shot, where the activated task model can be seen.

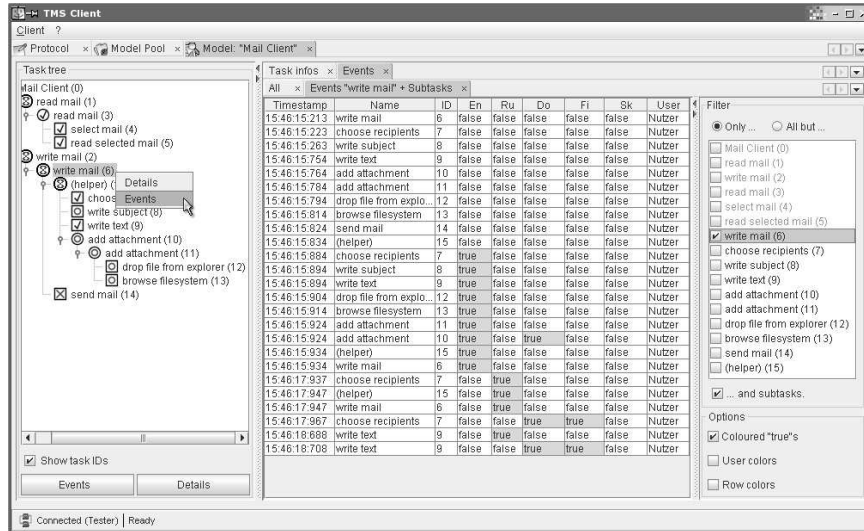


Fig. 9. User interface (animated task model) for usability experts

This user interface of Fig. 9 was technically produced based on a Java implementation, where parameterized cascaded observers were used to observe the states of the models on the server.

At the moment the usability expert has to watch the changes of the task model instance. He has to observe whether the test person behaves as expected or whether tasks are activated that have nothing to do with the actual test scenario. In this last case something is wrong with the system and a usability problem was found. In the future there will be further tool support such tests like automatic extraction of critical actions.

One can imagine that predefined scenarios are stored and as long as the test person acts according to these scenarios nothing happens. Otherwise the usability expert is informed that something happened that was not expected. It is his decision what to do next.

This kind of usability testing offers a new type of information, which is not strongly connected to specific UI-elements like buttons etc but to structural information that was transformed from models to the final software. This allows a much more detailed analysis of the software's logical structure than other test methods, and different versions of the software generated from different versions of the task model can be compared and optimized.

Of course, it is not the intention to replace all other kinds of observations by this method but it is intended to use this idea additionally.

6 Summaries and Outlook

Within this paper a model-based development (Fig. 1) process was proposed. This process model postulates the idea of having the same models as basis of the work of software engineers and usability engineers. This fact is very important for us. Even if not every detail of the models is important for both groups of the different developers there have to be common models as basis of the development process. This is one big first step towards bridging the gap between software engineering and usability engineering. The approach has the following advantages:

- Software engineers are focused on the tasks a user has to perform, on the object he has to manipulate and on the work situation (context of work). In this way it is guaranteed that the developed interactive systems are user-centered.
- Usability experts are involved in the specification of models in very early development stages. In this way they can influence the implementation process of the software engineers and integrate usability test results to ensure the task model and the modeled temporal relations to fulfill the user's needs and support their work in an efficient manner.

- Usability evaluation is supported during all stages of the development process based on the existing models. Especially remote usability testing can be supported.

We have been using our approach in different small projects. However, up to now we did not use models for projects of large scale. Currently we are working in a project for a mobile maintenance system together with 8 partners from university and 9 partners from industry, which follows our model-based approach. The project started 2004 and will be finished 2006. We are sure that the demonstrated approach will lead to a successful result of the project and that we will gain further knowledge to improve our tool for remote usability testing in a mobile environment.

References

1. Cameleon: <http://giove.cnuce.cnr.it/cameleon.html>.
2. Clerxkx, T.; Luyten K.; Conix, K.: The Mapping Problem Back and Forth: Customizing Dynamic Models while preserving Consistency, Proc. TAMODIA 2004, P. 33-42.
3. Constantine L.L: Canonical Abstract Prototypes for Abstract Visual and Interaction Design, in Jorge J. A. et. al (Eds): *Proceedings DSV-IS 2003*, LNCS 2844, Springer Verlag, Berlin, 2003, P. 1-15.
4. CTTE: The ConcurTaskTree Environment. <http://giove.cnuce.cnr.it/ctte.html>.
5. Deakin, N.: XUL Tutorial. XUL Planet. 2000.
6. Dittmar, A., Forbrig, P.: The Influence of Improved Task Models on Dialogues. *Proc. of CADUI 2004*, Madeira, 2004.
7. Dittmar, A., Forbrig, P., Heftberger, S., Sary, C.: Tool Support for Task Modeling – A Constructive Exploration. *Proc. EHCI-DSVIS'04*, 2004.
8. Dittmar, A., Forbrig, P., Reichart, D.: Model-based Development of Nomadic Applications. In *Proc. of 4th International Workshop on Mobile Computing*, Rostock, Germany, 2003.
9. Eclipse: <http://www.eclipse.org>.
10. Elwert, T., Schlunbaum, E.: Dialogue Graphs – A Formal and Visual Specification Technique for Dialogue Modelling. In Siddiqi, J.I., Roast, C.R. (ed.) *Formal Aspects of the Human Computer Interface*, Springer Verlag, 1996.
11. Forbrig, P.; Lämmel, R.; Mannhaupt, D.: Patterns-oriented development with Rational Rose, *Rational Edge*, Vol. 1, No. 1, 2001.
12. Hilbert, D.M., Redmiles, D.F.: Extracting usability information from user interface events, in: *ACM Computer Surveys*, Vol. 32, Issue 4, ACM Press New York.
13. Limbourg, Q., Vanderdonck, J.: Addressing the Mapping Problem in User Interface Design with USIXML, *Proc TAMODIA 2004*, Prague, P. 155-164.

14. López-Jaquero, V.; Montero, F. ; Molina, J.,P.; González, P.: A Seamless Development Process of Adaptive User Interfaces Explicitly Based on Usability Properties, *Proc. EHCI-DSVIS'04*, p. 372-389, 2004.
15. Luyten, K., Clerckx, T., Coninx, K., Vanderdonckt, J.: Derivation of a dialog model from a task model by activity chain extraction. In Jorge, J., Nunes, N.J., e Cunha, J.F. (ed.), *Proc. of DSV-IS 2003*, LNCS 2844, Springer, 2003.
16. Mozilla.org: XUL Programmer's Reference 2001.
17. Nielsen, J.: *Usability Engineering*, Academic Press, Boston, 1993.
18. Paterno, F.; Mancini, C.; Meniconi, S: ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models, *Proc. Interact 97*, Sydney, Chapman & Hall, p362-369, 1997.
19. Paterno, F., Santoro, C.: One Model, Many Interfaces. In *Proc. of the Fourth International Conference on Computer-Aided Design of User Interfaces*, p. 143-154. Kluwer Academics Publishers, 2002.
20. Puerta, A.R. and Eisenstein, J. Towards a General Computational Framework for Model-Based Interface Development Systems. *Proc. of the 4th ACM Conf. On Intelligent User Interfaces IUI'99* (Los Angeles, 5-8 January 1999). ACM Press, New York (1999), 171–178
21. Reichart, D.; Forbrig, P.; Dittmar, A.: Task Models as Basis for Requirements Engineering and Software Execution, *Proc. of Tamodia*, Prague, 2004, p. 51-58
22. Sinnig, D., Gaffar, A., Reichart, D., Forbrig, P., Seffah, A.: Patterns in Model-Based Engineering, *Proc. of CADUI 2004*, Madeira, 2004.
23. TERESA: <http://giove.cnuce.cnr.it/teresa.html>
24. UML: <http://www.uml.org/>
25. UsabilityNet: http://www.usabilitynet.org/tools/r_international.htm
26. Wilson, S.; Johnson, P.: Bridging the generation gap: From work tasks to user interface design, In Vanderdonckt, J. (Ed.), *Proc. of CADUI 96*, Presses Universitaires de Namur, 199, p. 77-94.
27. Wolff, Andreas, Ein Konzept zur Integration von Aufgabenmodellen in das GUI-Design , Master Thesis, University of Rostock, 2004.
28. Wolff, A.; Forbrig, P.; Dittmar, A.: Reichart, D.: Linking GUI Elements to Tasks – Supporting an Evolutionary Design Process, accepted for TAMODIA 2005, Gdansk.
29. Wolff, A.; Forbrig, P.; Dittmar, A.: Reichart, D.: Development of Interactive Systems Based on Patterns, accepted for the workshop “Development of Interactive Systems Based on Patterns” at INTERACT 2005, Rome.
30. XUL: <http://www.xul.org/>