A Methodology for Model-Driven Development of Crisis Management Applications using Solverational

Andreas Petter, Alexander Behring, Max Mühlhäuser

{a_petter,behring,max}@tk.informatik.tu-darmstadt.de

Abstract: Efficient usage of applications often requires context-aware applications. We assume that context-aware adaption can be useful in the case of crisis management tools as well, where fast responses are crucial and can be supported by information technology. A small user study performed with a crisis response team mostly composed of fireman supports our assumptions for the important case of role-aware messaging applications. We demonstrate our way to develop context-aware user interfaces using our model-to-model transformation language with constraint solving and apply it to the case of a crisis management system.

1 Introduction

Crisis management applications provide new, more efficient means to fight crisis and therefore have much potential to save many lives during the crisis when used efficiently. One important factor for efficiency is usability, because higher usability potentially shortens the time to complete the task of the crisis management team members and furthermore can reduce the error rate.

Context-aware applications are being used to provide applications which adapt to the needs of specific users or situations. Therefore, they potentially can provide user interfaces which have higher usability and shorter times to complete tasks for specific users. E.g. our scenario involves the use of context as to adapt to the role of the user as well as of the display resolution.

We therefore investigate the development of context-aware applications for crisis management. We focus on the development of user interfaces, but our concepts may be used for other context adaptions as well - using different transformations and models. Most benefit can be gained by supporting users by their tasks most frequently performed - which is messaging in our case. Our scenario is based on messaging applications for crisis management teams.

This paper provides the following contributions:

- A user study involving real firemen to support our assumptions about to the application of context-aware adaption to the software used by crisis management teams
- A methodology to develop models and applications based on our model-to-model

transformation language called “Solverational”

- Developing context-aware applications using Solverational
- The application of the methodology to the development of crisis management tools

1.1 Motivating User Study

German crisis management teams of fire brigades manage crisis in the event, that the control center is overloaded by a huge amount of small incidents or a several huge incidents which require greater attention. The crisis management team is usually based on 4 or 6 “S-role” team members, who have different responsibilities, e.g. decide on tasks and actions to take, manage the map, organize meals for firemen, or organize reinforcements. The “communication center” communicates with fireman not situated at the current location. The “information manager” gets all messages and distributes them under the staff members. Since he gets all messages, he will be overloaded, when many messages arrive within a small time frame.

To motivate our investigations on context aware computing for crisis management teams we report on a user study performed with 14 users, most of them were firemen trained in the field of crisis management and messaging. Each of the participants performed the study once.

Messaging is an important information channel for the crisis management team. Currently, the team of investigation is using Microsoft Outlook to transfer messages. The firemen believed that it is the right choice for the task of messaging, because most firemen know Outlook. However, we believed that this could be different for the role of the information manager, who was announced to be the “bottleneck” in case many messages arrive within a small time frame. His task is to select recipients for messages from the staff members. Due to provisions of national laws, all messages must pass the information manager and he therefore must be very fast in selecting recipients to circumvent delays in message transfers.

We presented several different user interfaces (see figure 1) for a messaging application and asked the firemen to perform the task of the information manager. Each interface is composed of a list of messages, a message containing the contents of the message, and a panel full of buttons with roles of the recipients. Although the user interfaces in figure 1 are rather small, the figures show a version similar to the interface known from Outlook (list of messages on the left side), a horizontally flipped version (list of messages to the right), and a vertically flipped one (list of messages to the bottom). The time to complete the task and the time to click on a button were recorded. The firemen had to perform 99 tasks with different versions of the user interfaces. In an effort to reduce learning effects these were displayed in random order, though the numbers of the interfaces remained constant over the group of participants.

Results are shown in figure 2. The firemen performed best with the horizontally flipped version. We believe that this is a result from the combination of the reading direction

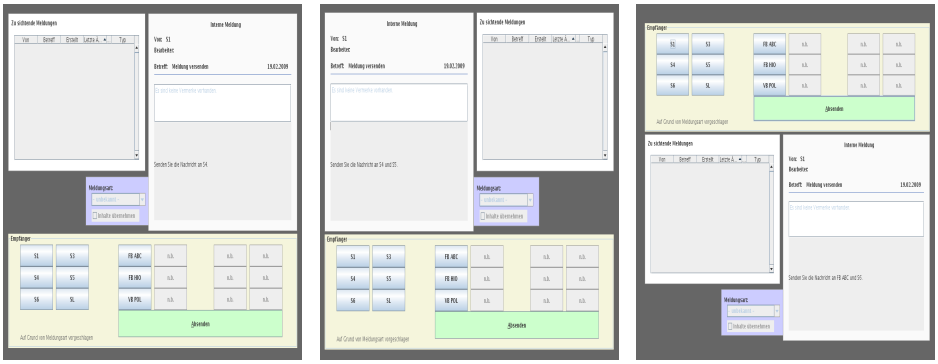

Figure 1: User interfaces of the messaging application.

(left-to-right and top-to-bottom) and the importance of the list of messages. The list of messages is not important for the information manager, because he has to pass every message, anyways. Therefore he performs better when the list of messages does not gain his attention and can be left aside. The horizontally flipped version shows the list of messages on the right side, and the firemen were able to ignore it.

However, it is important to note that this is only the case for the information manager. Most other staff members use the list of messages to organize their messages and will therefore first look at the list instead of the message content itself. The applications is therefore used in two different contexts with different tasks, although both handle messaging. We conclude that the context of use is important for efficient use of messaging in crisis management applications. Still, the user study can only be seen as a hint for the problem given and not as a formal evaluation.

1.2 Paper Outline

Section 2 demonstrates our model-driven methodology used in combination with a model-to-model transformation language called “Solverational” to develop context aware user interfaces for a more general setup, while section 3 specializes our methodology to crisis management. Section 4 summarizes the state of the art of developing user interfaces using model driven development. Section 5 discusses our findings and section 6 summarizes the conclusions of the work being presented.

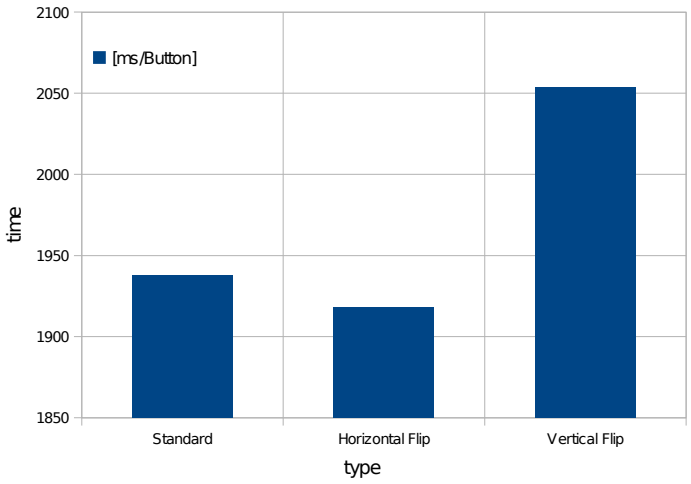

Figure 2: Results of the user study.

2 General Methodology

We present a methodology to develop context-aware applications. The next section specializes this methodology to the development of context-aware crisis management applications.

2.1 Model Driven Architecture

Model driven architecture (MDA) is an approach proposed by the OMG [OMG03] to develop systems and applications using a model driven approach. The OMG outlines that in model driven development there are abstract models (called “platform independent models”, PIM) which are to be refined into concrete models (called “platform specific models”, PSM) through the use of intermediate models (called “platform models”). Furthermore this refinement process can be performed using transformations. It is not specified how this transformation process should look like exactly, but of course, it is highly desirable to use automatic means to support developers developing platform specific models.

This automatism can be provided using a model-to-model transformation language. These languages specialize on the transformation of PIMs to PSMs. The PM (“platform model”) may either be specified in the transformation itself, or provided using another input model (or will be left out, if it is not needed).

2.2 Transformation Language

For that reason the OMG specified the QVT standard [OMG07]. QVT is a standard for model-to-model transformation composed of three different languages. The most interesting, because it is a highly declarative one (declarativity is a property desirable for model-to-model transformation) [JK06], is the QVT Relations language, which is a model-to-model transformation language based on relations between model elements. Therefore, developers do not specify the way the model-to-model transformation process is performed, but the way the result of the transformation should look like. The algorithm to perform the transformation is inferred by the transformation engine.

A QVT Relation transformation is composed of a set of relations. These relations are the transformation rules which specify the result of the transformation. Every relation consists of a set of domains, which specify the types (classes) of model elements to be used by the relation. By specifying the input and output models of a transformation the domains get bound to model elements of the selected types (classes). Because QVT Relations requires that each model element of each domain in each relation is exactly represented once in the other domains of the relation, this results in a relational approach to programming model-to-model transformations. Each domain is composed of several templates, most notably PropertyTemplateItems. PropertyTemplateItems are equations, which require properties of model elements to have or get values assigned by a value expression given in a language called OCL [OMG06].

2.2.1 Solverational

Our model-to-model transformation language called "Solverational" is based on the QVT Relations standard, but adds constraint solving to it [PBM09]. In fact many QVT Relations transformations can be executed using our implementation of the Solverational transformation engine. The approach presented here could probably be used with classical, less expressive QVT Relations transformations as well, if the application of the methodology presented in section 3 would not require the expressiveness of constraint programming.

Transformations written in Solverational are very similar to the ones written in QVT Relations. However, since QVT Relations does not support constraints (other than equalities, which could arguably be called constraints), Solverational is more expressive, although only minor changes to the language were needed. In fact, only the definition of the PropertyTemplateItems had to be changed, such that they handle also inequalities instead of equalities. Additionally, these PropertyTemplateItems may be used several times on the same property using different inequality constraints (this does not make sense in the case of equalities, because the property value is already perfectly specified after the first equality).

In addition to the ability to process constraints, the Solverational transformation engine is able to perform mathematical optimization during the transformation process. This is useful in the case of several solutions (i.e. the system is under-specified) where an optimal solution is required by the transformation developer. This may result in very hard

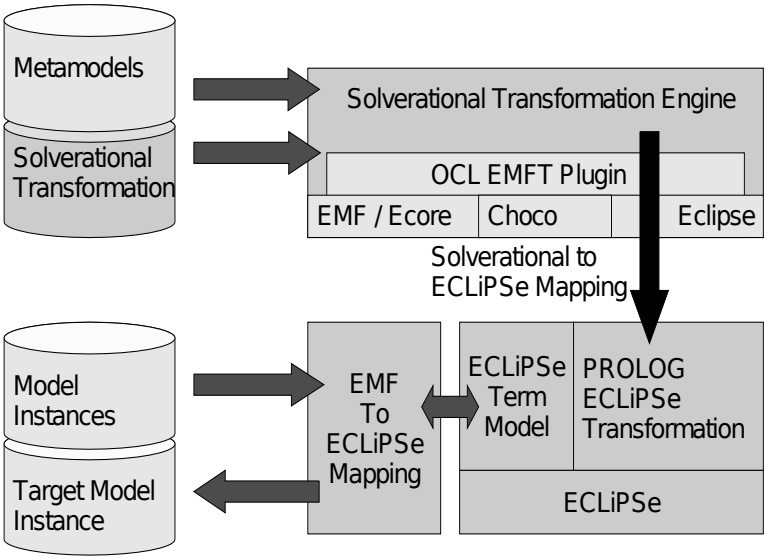

Figure 3: Architecture of the transformation engine.

computational problems called “constraint optimization problems”.

2.2.2 Separation of Concerns using the Solverational Transformation Engine

Figure 3 (which was taken from [PBM09]) illustrates the architecture of our transformation engine for Solverational.

Developing applications using Solverational typically involves three different roles of development. If separation of concerns (SOC) is maximized these roles can be fulfilled by different developers (e.g. a big company has experts for all concerns), but may as well be fulfilled by a single person.

Meta-model developers develop the meta-models used for specifying the models and the transformation definition. Depending on the number of meta-models involved this may be done by several “meta-modelers”. If domain specific languages need to be developed, the meta-modelers will talk to customers about the terminology used by them or people with domain knowledge (e.g. firemen in our case) to derive the meta-models. In addition to the model elements and associations they will also pay attention to constraints commonly used in the languages to support the transformation developers to write the constraint programs.

Model developers develop the models needed during the development process to develop new applications. They carefully talk to customers and developers and retrieve the

specific models which should finally be implemented by the application. In addition to the model elements and associations they will also pay attention to constraints implied by the models to support the transformation developers to write the constraint programs.

Transformation developers develop transformations using Solverational. They specify the transformation. In addition to knowledge needed by common transformation languages developers should have basic knowledge in the field of constraint programming to prevent over/under-specification and unnecessary long runtimes. Transformation developers will carefully listen to the requirements provided by the model developers to automate their development processes. In Solverational the transformation description will most often be used as the platform model, because many platform specific aspects can be encoded using Solverational constraints.

After having developed the meta-models and the transformation definition the transformation engine will transformation these artifacts into an ECLiPSe program (Eclipse Constraint Logic Programming System [AW07]; not to be confused with the Eclispe Java IDE), which can then be used by the modelers.

The modelers, whose work has been automated using the transformation, will then use the models (PIMs) and the transformation and feed them into the ECLiPSe program. The program will return a target model (PSM).

2.3 Context-awareness

"Context" encompasses all data used by the application which is not directly used for input. A "situation" is a set of values of "context variables" and may be values of functions which are computed to get "higher context information" or may be directly measured. Then, context is a set of situations. Each situation is part of at least one specific context.

Reacting to a new "context" is composed of three steps:

1. detecting the context,
2. calculating a reaction to the context, and
3. executing the reaction

Context detection can be done using constraints on values of context variables. The constraints then define the range or the domain for which the situation being measured may still be assumed to be within the context. Therefore, the description of context may best be done using a set of constraints (in our case, which is probably a subset of all possible cases). If, at some time during application runtime, the constraints will be violated, the context will be switched to a context with constraints which can all be satisfied (context change event). To compute higher context information the constraints may also contain complex mathematical functions over the context variables.

In the case of constraint programming, calculating a reaction to the newly detected context can be done by solving constraints. Therefore, a context not only is associated with a set of constraints for context detection, but also with a set of constraints, which are used to compute a reaction to it. In case of constraint programming usually a constraint solver will calculate the solution (and therefore the reaction).

The execution of the reaction then is simply the application of the values gained from the constraint solver to the application itself - and therefore is highly application specific.

2.4 Developing Context-Aware Applications using Solverational

Integrating context-awareness into the models used for application development is called "context-aware modelling". There are several ways to integrate the development of context-aware modelling with a model-driven approach using transformations.

A major difference to the ideas presented in section 2.3 is that context aware adaption is not performed at runtime, but precomputed at design time. This reduces the runtime needed to compute the reaction, but the model-to-code transformation needs to be aware of selecting the right contexts and contexts, which were not foreseeable at design time cannot be used at runtime. However, we believe it is the best choice for model-driven development of applications where all contexts are known in advance and runtime or space is an issue (which we believe is true for the vast majority of applications).

The first way to implement ideas from section 2.3 in model-to-model transformations is to use constraints for the selection of the context. The constraints can be used to select the context by writing them into the pattern matching part of the transformation rules. Each context then gets a set of transformation rules which all contain the constraints for selection of the context in the pattern matching part. The correct transformation rules for the situation are therefore activated by using pattern matching with the constraints from the context. Computing the reaction is as easy as using the constraints for computing the reaction in the domain which should be enforced. The transformation engine will then solve the constraints and therefore compute the reaction.

The second way to implement the context-aware modelling in model transformations is to use a different transformation for each context. The context then is selected at runtime and will select the code and data produced by a subsequent step of the model-to-model transformation process. Computing the reaction is similar to the one presented above.

As the Solverational transformation engine is capable of performing constraint solving in target models and select model elements using constraints it is perfectly suited to implement both ways.

The work presented here is similar to the work presented in [PBS06] and somehow may be seen as an implementation. However, the transformation is not performed at runtime, but at design time. Therefore, the duration of the transformation step is not as important as it would be in the case of [PBS06].

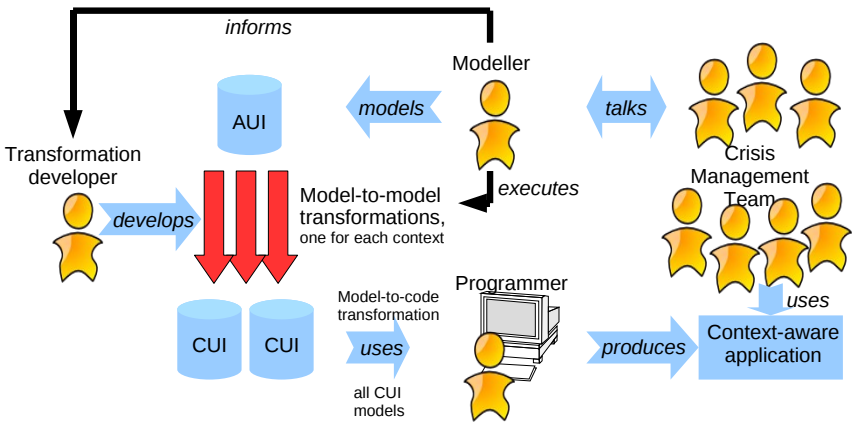

Figure 4: Application of the methodology to crisis management.

3 Application to Crisis Management

The application to developing applications for crisis management is straight forward. As has been motivated by our user study context-awareness is an issue for crisis management messaging applications.

Adhering to the second way presented in the previous section a team of developers will use appropriate model-to-model transformations. For example, our messaging scenario adapts to different screen sizes. Therefore, we develop a model-to-model transformation using Solverational that sets attribute values according to an optimization function based on Fitt's law and the keystroke level model (KLM).

The transformation transforms an abstract user interface model (PIM) into a concrete user interface model (PSM). Abstract user interface models (AUI) abstract from the specific dimensions of the screens used and therefore only define that the application needs a specific set of input variables filled in by the user. The concrete user interface model (CUI) already has all information needed for positions and sizes of the components. The model elements representing the interaction components are organized in a hierarchy that is finally used to display the model on the screen.

The information set by the transformation Fitt's Law relates the positions of user interface components to their sizes. Therefore the transformation pre-computes position and sizes for user interface components (designed to be an implementation of our concepts presented in [PBZ+08]). The KLM defines the time needed to interact with the user interface component.

Currently our implementation lacks the ability to take into account both, screen sizes and roles. In fact the transformations are still very simple (see [PBM09]). Currently we are

working on a transformation that includes roles into the transformation, such that it not only adapts to platform context, but also to role based contexts. However, the methodology is still valid.

3.1 Process

Figure 4 presents an overview of the methodology. The modeller talks to the crises management team to derive terms to be used and user interfaces. He informs the transformation developer (if the transformation has not been developed, yet) that he would like to have context-aware adaptation in model-to-model transformations and provides some constraints. Then, he models the user interface and maybe the rest of the application logic. The transformation developer develops the model-to-model transformations necessary to transform the models into their context-dependent versions. These models are used by the programmer who produces the context-aware application using a model-to-code transformation or an interpreter to interpret the models. The application is tested by the crisis management team to give feedback to all involved software developers, resulting in an iterative process until the final application can be deployed.

4 Related Work

Several methodologies (which may have been explicitly or implicitly given in the literature describing them) for developing context-aware user interfaces exist.

Quentin Limbourg provides a set of models and a methodology to develop multimodal application using model driven development [Lim04]. His methodology is based on UsiXML, which is a language to describe user interfaces and does not use constraint solving.

Mori and Paterno provide a methodology to develop user interfaces with model-driven development starting from a task model [MP05]. Their approach has been extended to adaptive user interfaces and uses model-to-model transformations, but does not use constraint solving and does not use a model-to-model transformation language.

Serco uses model-to-model transformation and constraint solving to transform a set of user interface and task models into concrete user interface models [FBSA08]. They do not use a model-to-model transformation language, but provide a tool to edit the constraints instead. However, the methodology is based on the assumption that the model transformation process is performed at runtime, which is not the case for our scenario.

Banavar et. al provide a methodology which is able to distinguish between developing a single language and a set of platform specific languages [BBG$^+$04]. Their approach does not use constraint solving.

MASTERMIND was an user interface management project at the advent of model based user interface technologies and provides a code generation approach based on models [BDR$^+$97]. The methodology behind MASTERMIND focuses on the generation of code

and does not directly address constraint solving and does not use a model-to-model transformation language.

SUPPLE uses mathematical optimization and constraint solving to develop user interfaces [GW04]. It is not based on model-to-model transformation but is directly implemented in JAVA.

Cassowary is a constraint solver specifically designed for mathematical user interface models [Bad00]. However, it does not use models from software engineering, nor does it use model-to-model transformations.

Thevinin presents ArtStudio, which is an approach based on model-to-model transformation, which uses a simple rule based language to transform the models [The01]. After the transformation process constraints are solved using the Cassowary constraint solver and therefore constraint solving is excluded from the model-to-model transformation process.

Most of the methodologies have different methodologies to develop user interfaces based on model-driven development. Some of them can easily be extended or can be used with adaptive user interfaces. As none of the approaches is using a model-to-model transformation language with constraint solving, the methodologies can not be used for using the promising features of Solverational. Therefore, our methodology focuses on the aspects of using constraint solving and mathematical optimization in the model-to-model transformation process and the language.

5 Discussion

Our methodology is based on MDA and specializes it for our purposes which are threefold:

1. A methodology for model-driven development using Solverational
2. A methodology for model-driven development of context-aware applications
3. A specialization to develop context-aware applications for crisis management using Solverational

Since our methodology is just a specialization of MDA with add-ons for constraint solving, we believe that it is as least as good as developing applications with MDA. Still, the changes to MDA are significant, as MDA is not concerned with constraint solving or mathematical optimization. We therefore believe that our methodology provides a speedup as soon as constraint solving or mathematical optimization can be used within transformations. Since the development of context-aware applications can be perfectly done using constraint solving (see section 2.3) the methodology will very likely provide a speedup.

To perform quantitative analysis of methodologies many projects of appropriate sizes need to be finished to evaluate the effectiveness. Therefore, lacking enough projects of appropriate size our evaluation can only be qualitative.

However, our methodology will be applied to the development of context-aware applications in the SoKNOS project. It focuses on the development of a larger crisis management

system. While we only use our methodology to develop messaging applications it is a start for further evaluation of the methodology in the future.

The user study provided in section 1 motivates the use of context-aware adaptation for roles. Our current transformation (see section 3) can only handle user interface adaptation for platform context (screen and window resolutions). However, we think that our methodology is valid for role based adaption, also. By applying our methodology in the SoKNOS project and extending the transformation to include role based adaptation, we will further investigate if the methodology has to be changed to support role based adaptation. Furthermore we think that our user study indicates that the team members work faster with adapted user interfaces, probably not only with role based ones. Therefore, any user interface adaptation that provides a better user interface may have produced such a result. As we will improve the usability by using our platform context adaptation we still believe to be partially motivated by the user study, even without having a role-based adaptation mechanism, yet.

6 Conclusions and Future Work

A user study motivated the use of context-aware computing for crisis management applications. Our study involved real firemen, which are usually hard to convince to test new software. As a result messaging applications should adapt to the roles of their users.

We developed a methodology, which uses a model-to-model transformation language with constraint solving at its heart. The methodology provides a way to develop models and applications based on our model-to-model transformation language called Solverational and guides developers in developing context-aware applications.

We applied the methodology to the development of crisis management applications. In the future we will integrate several adaptations into the transformation, most notably role based adaptation, as directly motivated by the user study. However, we believe that the user study also motivates more general adaptions which improve usability.

Acknoledgements The work presented is financed by the German Ministry of Education and Research, BMBF. It is part of the SoKNOS project.

References

[AW07] Krzysztof R. Apt and Mark Wallace. *Constraint Logic Programming using Eclipse*. Cambridge University Press, New York, NY, USA, 2007.

[Bad00] Gregory Joseph Badros. *Extending Interactive Graphical Applications with Constraints*. PhD thesis, University of Washington, 2000.

[BBG+04] Guruduth Banavar, Lawrence D. Berman, Yves Gaeremynck, Danny Soroker, and Jeremy Sussman. Tooling and system support for authoring multi-device appliations. *Systems and Software*, 69(3):227–242, December 2004.

[BDR+97] Thomas Browne, David Davila, Spencer Rugaber, , and Kurt Stirewalt. *Formal Methods in Human Computer Interaction*, chapter Using Declarative Descriptions to Model User Interfaces with MASTERMIND. Springer, 1997.

[FBSA08] Sebastian Feuerstack, Marco Blumendorf, Veit Schwartze, and Sahin Albayrak. Model-based layout generation. In *AVI '08: Proceedings of the working conference on Advanced visual interfaces*, pages 217–224, New York, NY, USA, 2008. ACM.

[GW04] Krzysztof Gajos and Daniel S. Weld. SUPPLE: automatically generating user interfaces. In *IUI '04: Proceedings of the 9th international conference on Intelligent user interfaces*, pages 93–100, New York, NY, USA, 2004. ACM Press.

[JK06] Fréderic Jouault and Ivan Kurtev. On the Architectural Alignment of ATL and QVT. In *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*, pages 1188–1195, New York, NY, USA, 2006. ACM Press.

[Lim04] Quentin Limbourg. *Multi-Path Development of Multimodal Applications*. PhD thesis, Université Catolique de Louvin, 2004.

[MP05] Giulio Mori and Fabio Paternó. Automatic semantic platform-dependent redesign. In *sOc-EUSAI '05: Proceedings of the 2005 joint conference on Smart objects and ambient intelligence*, pages 177–182, New York, NY, USA, 2005. ACM Press.

[OMG03] J. Mukerji OMG, J. Miller. MDA Guide Version 1.0.1. OMG, June 2003. document number: omg/2003-06-01.

[OMG06] OMG. Object Constraint Language OMG Available Specification Version 2.0. OMG, May 2006.

[OMG07] OMG. Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. OMG, July 2007. ptc/07-07-07.

[PBM09] Andreas Petter, Alexander Behring, and Max Mühlhäuser. Constraint Solving in Model Transformations. In Richard F. Paige, editor, *International Conference on Model Transformation, ICMT 2009*. Springer, 2009. to appear.

[PBS06] Andreas Petter, Alexander Behring, and Joachim Steinmetz. Efficient Modelling of Highly Adaptive UbiComp Applications. In G. Kortuem, editor, *Workshop on Software Engineering Challenges for Ubiquitous Computing*, pages 47–48, June 2006.

[PBZ+08] Andreas Petter, Alexander Behring, Miroslav Zlatkov, Joachim Steinmetz, and Max Mühlhäuser. Modeling Usability in Model-Transformations. In Marko Bo?kovic, Dragan Ga?evic, Claus Pahl, and Bernhard Schätz, editors, *Proceedings of the 1st International Workshop on Non-functional System Properties in Domain Specific Modeling Languages, NFPinDSML-2008*, volume 394. CEUR, September 2008. ISSN 1613-0073.

[The01] David Thevenin. *Adaptation en Interaction Homme-Machine : le cas de la Plasticité*. PhD thesis, Université de Grenoble, 2001.