Modeling Usability in Model-Transformations

Andreas Petter[1], Alexander Behring[1] Miroslav Zlatkov[1], Joachim Steinmetz[1], and Max Mühlhäuser[1]

Technische Universität Darmstadt, Department of Computer Science, Telecooperation, Hochschulstr. 10, D-64289 Darmstadt, Germany,
{a_petter,behring,zlatkov,joachim,max}@tk.informatik.tu-darmstadt.de

Abstract. Developers of transformation rules for user interface models should have the option to support usability in their transformations. As different aspects of usability highly depend on each other, transformation rules should be able to model these dependencies. We provide an example how this can be done in a transformation language through a QVT Relations dialect.

Keywords: usability, model transformation, constraint solving

1 Introduction

When users assess the quality of software systems, one of their major concerns is the usability of user interfaces. The first impression of users will often be the user interfaces, even before using an application. They will tend to judge the application exactly at this first glance, although the functional features of an application might be completely different from the quality of the user interfaces.

However, in the case of ubiquitous computing creating user interfaces is a problem. Applications must run on a large set of devices, each being equipped with different hardware to interact with users. This possibly results in a large set of user interfaces - in the worst case one set per device. Additionally, applied context-awareness is common in ubiquitous computing. This raises the number of interfaces to be tested for usability even further. Having such a large amount of user interfaces to be tested for usability is hardly feasible. It is of course similarly or even more laborious to develop these many interfaces. As a result automatic means should be used to develop and test the user interfaces. It would be advantageous to develop user interfaces which are usable by design.

Automation of developing and testing for quality of user interfaces relies on a formalization of usability. Furthermore, the formalization must be usable in appropriate development and testing tools. Today, not all aspects of usability can be modelled in a formal way, e.g. "I feel that my software looks bad. I want it to be changed". However, even a partial model of usability can help to shorten development and testing phases. The model will then help in the aspects it covers and prevents developers of reinventing these, giving them more time to think about unhandled aspects or to refine the model, if the transformation is not satisfying.

The next section presents the current state of the art. The third section presents the approach. In the fourth section we show how we used optimization in conjunction with an abstract user interface, followed by a section with an example transformation. Finally, we present our conclusions from the transformations and work to be done in the future.

2 Related Work and Problems Modeling Usability

Developing user interfaces automatically requires tools that can rely on a formalization of usability. Our goal is to demonstrate how to support the developer while he develops transformation rules. These transformation rules consider usability aspects in the transformation declaration. Features are identified, which a model transformation language should support that transformation developers are able to formalize their usability goals in their transformation rules.

First, we looked at different types of definitions of usability to find out which type of support should be provided for developing model transformations. Usability has many different aspects and therefore is hard to define. One of the more commonly used definitions can be found in ISO 9241-11, which defines it to be based on the effectiveness and efficiency to perform a task as well as users' satisfaction with the interface. Several other models of usability [1, 2] have been defined, but most of them are not formal enough to be used in tools. Other mostly qualitative metrics for usability are listed in style guides (e.g. [3]), which usually give a set of guidelines how a "good" user interface should look like. Almost all guidelines are not usable in model transformations or testing tools, because of their informal definition. For formal metric based approaches usability is mostly defined via quantitative measures, e.g. time to complete a task. A well known example is Fitt's law and its multidimensional variants [4]. Fitt's law is used often as a basis for more complex metrics, e.g. the one presented in [5]. It can be applied to perform optimization of user interfaces and therefore we studied it while we designed our model transformation language extension.

To construct user interfaces for many different target devices model driven engineering has been used (e.g. [6–9]). The common approach is to define models, called "abstract user interfaces", which abstract from device specific properties. Thereby, a single abstract user interface allows to describe interfaces for a multitude of devices at once, such that developers do not need to develop each one separately, called "concrete user interfaces". This usually is the case with the classical way to engineer user interfaces. Some of the approaches use model-to-model transformations (e.g. [7, 10–12]) to transform abstract user interfaces to concrete user interfaces, to further automate the process. This would be an ideal point during this automation to take usability aspects into account. Resulting user interface models should have the best quality possible, such that the interfaces only need to be refined as little as possible by hand.

A key challenge when taking this route is, that it is more difficult to model mutually dependent properties of usability within model-to-model transformation rules. This is due to the nature of current model-to-model transformation languages; they perform graph matching (also called "pattern matching") to select elements in source models and usually consider updating the target model to be directly related to the elements of the source models. As rules can not easily take into account the target model elements of other rules, they usually can not perform mutually dependent operations on target model elements such as would be done in constraint solving. Therefore, approaches tend to consider aspects only, which can be handled within those mutual independent rules.

Many rules which describe usability are not mutually independent. This can be illustrated by a simple example: ambient lighting and finger thickness both might affect button sizes for touchscreens of small devices. Because of this reason we concluded that common model transformation languages are not well suited to model usability in model-to-model transformations alone.

A different route is taken by approaches, which use constraints and optimization functions to model usability of user interfaces (e.g. [13–15]). Hereby, a constraint is a mathematical expression, which limits values of variables in the constraint. Especially in the case of modelling mutual dependent aspects, these approaches benefit from the possibility to use variables in several different constraints. The constraint solving algorithm then computes a solution for the complete set of constraints taking into account an optimisation function to get optimum values for the variables. This is especially useful to model mutual dependent aspects of usability. Gajos and Weld implemented an approach, which is able to display an abstract user interface model on many different devices [13] based on constraint solving and optimization. However, constraint solvers are usually not well suited to define model-to-model transformations as their main concern is to calculate a (mathematical) solution to a set of (mathematical) constraints. This means that the process of using models as their input is not self-explanatory as the models need to be transformed into an appropriate set of constraints and optimization functions.

In order to avoid the process to support usability, an appropriate transformation language is needed. We concluded that it would be advantageous to merge a model transformation language, handling models, and a constraint solver, handling constraints. This allows for both, using models and mutual dependent aspects in transformations, finally resulting in a new transformation engine.

3 Approach

Our idea is to support user interface developers creating multiple UIs in their transformations and therefore facilitate to model usability in transformations.

We assume that most model-to-model transformations which try to support usability in the transformation rules will transform an abstract user interface model into a more platform specific model. A model-to-model transformation language therefore needs to provide special support for this pattern. Furthermore a good model-to-model transformation should at least support constraints on user interface component sizes and should at least support Fitt's law, because this is the most common approach taken to mathematically describe usability.

Following this approach, our model-to-model transformations transform abstract user interface models into user interface models, which are more specific with regard to the intended platforms. The transformation is an automatic means to implement a simplification of ”AUI Refinement” [16], which originally has been mostly designed to be used by hand. AUI Refinement is an application of the MDA [17] to the development of user interfaces. The transformations were implemented in QVT Relations [18].

While we developed these transformations [8] we noticed that we cannot use constraints of target platforms within transformations and are not able to set sizes for user interface components. The reason is that component sizes are highly dependent on each other, because positions and spaces usually should not overlap. To model these usability properties in transformations, and especially the mutual dependent aspects, we recognized the need for constraint solving and therefore the need for an extension of model-to-model transformation languages.

Currently our model-transformation language extension is based on QVT Relations. Our choice was motivated because of its declarative nature, as declarative model transformation languages potentially ease developing model transformations due to several reasons [19].

3.1 Metamodel

This work focuses on the transformation of user interface models. It is very similar to the one used in [13]. The user interface models are comprised of a model of components that can be used, and a description of the user interface.

The compoents can be abstract (cannot be rendered directly) or concrete (can be rendered). With increasing level of the hierarchy the components are more concrete. E.g. “multiple entry component” is on the upper level while “list” is a specializiation and therefore situated on the lower level.

A sketch of the user interface is being represented in an abstract user interface model which is consisting of some of the components of the hierarchy. Additionally, the components are being enhanced by the type of input which is implemented by the component.

After the transformation, which is explained later on, the user interface model can be rendered because it will only consist of concrete components.

However, our work does not focus in the development of the metamodels. We see that almost any common widget library can easily be enhanced to fulfill the purpose of the final rendering as well as the generation of the hierarchy of components that can be rendered.

4 Usability Model Using Optimization

To develop a transformation language that can be used as a means to develop transformations that handle usability properties, we first modelled some aspects of usability to determine necessary language constructs. Therefore we created an abstract user interface model which is then mapped on a description of a concrete user interface with an automated process. This has been done using the OPL language, which is implemented in the ILOG CPLEX/CP optimization toolkit [20].

To formally describe usability we used a set of constraints and an optimization function. The mapping process we developed produces a new user interface. During the mapping it determines which user interfaces are "better" in terms of the optimization function and shall be used at last. The cost described by the optimization function is an estimation of the time, which is needed to use the interface. This is often used as a metric for usability testing (time to complete a task).

Starting from a model similar to the one given in [13], we produced a user interface description, which is then optimized using the optimization function. This model is a simplified user interface description, which in fact is nothing more than a tree of types. Every type is associated with a set of different user interface components, e.g. the type string array (basically a "list") is associated with a list of radio buttons or a drop-down list. These user interface components are used as the target description of the resulting user interface. During the constraint solving step the types are transformed into one of the associated user interface components.

Additionally, the associated user interface components have constraints like minimum size or if they can contain other user interface components. If they contain other components, the size of the component limits the size of the inner components.

To model usability all components are used in a metric [5] for an optimization function, as it was done in [13]. As it is the most common class of user interfaces our approach supports GUI components. For each GUI component the optimization function can then be formulated as a sum of the time to navigate to the component (with the mouse) and the time to manipulate the component (e.g. enter information). Determining manipulation time can be modelled fairly easy, it may be determined by the KLM [21]. For navigation time Fitt's law may be applied in a version that supports two dimensions of navigation [4], because GUI components have a width and a height.

Finally, the result of the optimization function is the sum of all costs of all costs of the user interface components. To be able to determine the navigation from component to component it must be determined how the user is going to navigate through the components. Unlike the approach presented in [13], which assumes an existing route (although it may be defined by the designer) through the user interface components, our optimization function takes into account all possible ways to navigate through the components. Because of its independence from a single guessed way or existing ways to navigate, our evaluation function can be applied before the user interface has ever been used. As soon as a path to navigate is known, it can also be used to optimize it to support faster navigation and provide an optimal solution in that specific case. Therefore, both scenarios are supported.

5 Transforming from QVT Relations Extension

After some aspects of usability have been defined we have constructed a transformation language supporting our notion of usability. It is able to at least handle our description of usability presented in the last section. This does not guarantee that other formal definitions of usability can also be used with the transformation language. However, since we based our language on a Fitt's law example, at least a large subset of all formal and quantitative usability definitions can be used in transformations as many definitions are based on it. The transformation language is based on QVT Relations. To execute a transformation, it is partially transformed into the mapping given in the previous section.

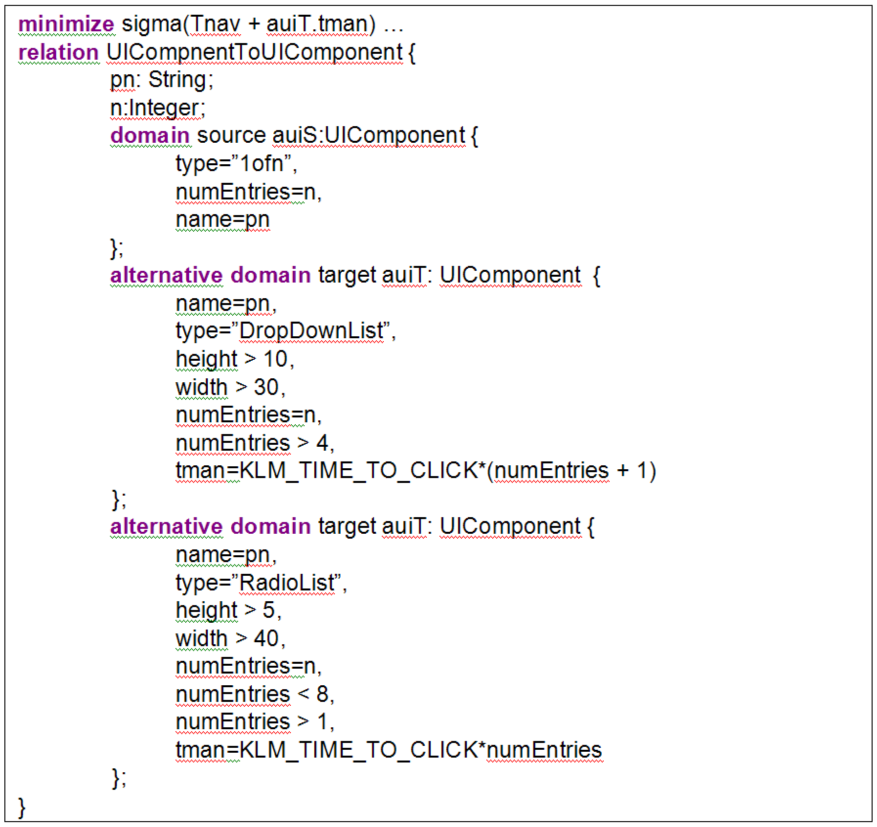
As a start to determine the effectiveness of the language, we tried to perform parts of the transformation from our QVT Relations dialect to the OPL language by hand. We give a small example of our dialect.

A classical QVT Relation transformation consists of several relations, each being a transformation rule. Every relation is a mapping of model elements of the source models to model elements of the target models. Furthermore several variables can be defined, which can store attribute values or model elements and can be used in the relation within the declaration of attributes of model elements.

In the simplified example shown in figure 1 there is a single relation ”UIComponentToUIComponent” which has two domains associated to the ”source” and the ”target” model. In the relation an abstract UIComponent (of type ”1ofn”, the string array type) is transformed into a more concrete UIComponent. There are two types of target components: a drop-down list and a list of radio buttons. Both set different constraints on the width and height of the user interface component to be displayed and both may be used in similar user interfaces and a manipulation time (”tman”). However, the optimization function (see first line of listing) selects appropriate components according to a minimization function. This function, which is simplified for the given example, is essentially the optimization function based on Fitt's law, which we used in OPL. However, it is modified to use attributes instead of variables and one can see that multiple model elements are calculated using a single term, instead of multiple, which would have the case for OPL. Tnav is a shortcut for the term that models Fitt's law.

Constraints on attributes can be directly used in OPL by creating a new variable in OPL for each attribute. Then the new OPL variables can be used in the optimization function, too. The minimization function is then transformed to the OPL function by adding variables and rolling out sigma (summation) terms into a sum of minimization functions for each component.

Currently we are implementing the transformation engine, which is capable of both handling EMOF (Eclipse Modeling Framework) and a subset of possible constraints in QVT Relations transformations. It is based on the MDT Eclipse OCL plugin. Transformations are executed using PROLOG and we are evaluating using ILOG for the optimization within the transformation engine.



```
minimize sigma(Tnav + auiT.tman) ...
relation UICompnentToUIComponent {
            pn: String;
            n:Integer;
            domain source auiS:UIComponent {
                     type="1ofn",
                     numEntries=n,
                     name=pn
            };
            alternative domain target auiT: UIComponent  {
                     name=pn,
                     type="DropDownList",
                     height > 10,
                     width > 30,
                     numEntries=n,
                     numEntries > 4,
                     tman=KLM_TIME_TO_CLICK*(numEntries + 1)
            };
            alternative domain target auiT: UIComponent {
                     name=pn,
                     type="RadioList",
                     height > 5,
                     width > 40,
                     numEntries=n,
                     numEntries < 8,
                     numEntries > 1,
                     tman=KLM_TIME_TO_CLICK*numEntries
            };
}
```

Fig. 1. QVT relation example with optimization

6 Conclusions and Further Research

Our contribution is twofold. First, we identified problems when modeling usability within common transformation languages if transformation rules depend on each other and second we gave an example how to model some aspects of usability within a newly created QVT Relations dialect.

In the future we will further work on the transformation language and its implementation, the transformation engine.

More aspecs of usability should be formalized. We are looking at a style guide to select rules that can be used within the transformation language.

References

1. Abowd, G.D., Coutaz, J., Nigay, L.: Structuring the space of interactive system properties. In: Proceedings of the IFIP TC2/WG2.7 Working Conference on Engineering for Human-Computer Interaction, North-Holland (1992) 113–129

2. Winter, S., Wagner, S., Deissenboeck, F.: A comprehensive model of usability. In: Proc. Engineering Interactive Systems 2007, Salamanca, Spain, Springer (2007)
3. Apple: Apple human interface guidelines (2005)
4. MacKenzie, I.S., Buxton, W.: Extending fitts' law to two-dimensional tasks. In: CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems, New York, NY, USA, ACM (1992) 219–226
5. Sears, A.: Layout appropriateness: A metric for evaluating user interface widget layout. IEEE Trans. Softw. Eng. **19**(7) (1993) 707–719
6. Paternò, F., Santoro, C.: One model, many interfaces. In: Proceedings of CADUI 2002, Valenciennes, France (May 2002)
7. Sottet, J.S., Calvary, G., Favre, J.M.: Towards model driven engineering of plastic user interfaces. In Pleuss, A., Van den Bergh, J., Hussmann, H., Sauer, S., eds.: Proceedings of Model Driven Design of Advanced User Interfaces 2005. Volume 159 of CEUR Workshop Proceedings., Montego Bay, Jamaica, online CEUR-WS.org/Vol-159/paper5.pdf (October 2 2005)
8. EMODE-Consortium: Deliverable d2.3 - modelltransformation. Technical Report TR-5, Telecooperation Research Division, TU Darmstadt, Darmstadt (July 2006) ISSN 1864-0516.
9. Nichols, J.: Automatically Gnerating High-Quality User Interfaces for Appliances. PhD thesis, Carnegie Mellon University (2006)
10. Limbourg, Q.: Multi-Path Development of Multimodal Applications. PhD thesis, Université Catolique de Louvin (2004)
11. Trapp, M., Schmettow, M.: Consistency in use through model based user interface development. In: Proceedings of CHI 2006 Workshop "The Many Faces of Consistency in Cross-Platform Design". (2006)
12. Luyten, K., Clerckx, T., Coninx, K., Vanderdonckt, J.: Derivation of a dialog model from a task model by activity chain extraction. In: DSV-IS. (2003) 203–217
13. Gajos, K., Weld, D.S.: Supple: automatically generating user interfaces. In: IUI '04: Proceedings of the 9th international conference on Intelligent user interfaces, New York, NY, USA, ACM Press (2004) 93–100
14. Badros, G.J.: Extending Interactive Graphical Applications with Constraints. PhD thesis, University of Washington (2000)
15. Browne, T., Davila, D., Rugaber, S., Stirewalt, R.E.K.: The mastermind user interface generation project. Technical report, Georgia Institute of Technology (1996)
16. Behring, A., Petter, A., Flentge, F., Mühlhäuser, M.: Towards multi-level dialogue refinement for user interfaces. In: Workshop on User Interface Description Languages. (Apr 2008) April 5-10, 2008, Florence, Italy.
17. OMG, J. Miller, J.M.: Mda guide version 1.0.1. OMG (June 2003) document number: omg/2003-06-01.
18. OMG: Meta object facility (mof) 2.0 query/view/transformation specification. OMG (July 2007) ptc/07-07-07.
19. Lawley, M., Raymond, K.: Implementing a practical declarative logic-based model transformation engine. In: SAC '07: Proceedings of the 2007 ACM symposium on Applied computing, New York, NY, USA, ACM (2007) 971–977
20. ILOG: Cplex. Internet (2008) http://www.ilog.com/products/cplex/.
21. Kieras, D.: Using the keystroke-level model to estimate execution times. Technical report, University of Michigan (2001)