Model-driven Transformation for Optimizing PSMs
A Case Study of Rule Design for Multi-device GUI Generation

David Raneburger, Roman Popp and Hermann Kaindl
Institute of Computer Technology, Vienna Universtity of Technology, Gusshausstrasse 27-29, Vienna, Austria

Keywords: Platform-specific Model, Transformation Rules, Rule Design, GUI Generation, Optimization.

Abstract: Software design and implementation, in general, have to take many alternatives into account for decision making. Still, current approaches to Model-driven Architecture (MDA) typically transform in one and only one thread from Platform-independent Models (PIMs) to Platform-specific Models (PSMs). Also in the special case of automatically generating graphical user interfaces (GUIs) according to MDA, in most approaches one thread derives a Final User Interface Model from some higher-level model(s). We think that this is one reason for less than optimal usability of automatically generated GUIs.
Our transformation approach (as implemented for GUI generation) allows exploring different design alternatives and evaluating the resulting PSMs according to given optimization objectives. Such a search approach leads to optimal PSMs according to these objectives. In this way, resulting GUIs can be tailored for different devices such as tablet PCs and smartphones. In this context, we design transformation rules for optimizing PSMs independently from concrete device properties, which are given in separate device specifications (Platform Models). We present a related case study for multi-device GUI generation and show that this approach facilitates the automated optimization of PSMs for several devices.

1 BACKGROUND AND INTRODUCTION

The Object Management Group's Model Driven Architecture[1] (MDA) proposes a generic concept to refine models over different levels of abstraction to source code. Between these levels, MDA proposes the use of transformation rules that transform a certain source model to a certain target model, as illustrated in Figure 1 for a transformation from Platform-independent Model (PIM) to Platform-specific Model (PSM). Important promises of MDA are, e.g., faster time-to-market and easier maintenance (Truyen, 2006).

Figure 1: Standard Model-driven Transformation.

Transformation languages like the Atlas Transformation Language[2] (ATL) or Query / View / Transformations[3] (QVT) have been developed to support the specification of transformation rules. These languages can reference any meta-model specified in an appropriate modeling language (typically Meta Object Facility[4] (MOF) compliant languages like the Unified Modeling Language[5] (UML) or Ecore) and are thus also applicable for model-driven UI generation.

Each of these languages comes with its own transformation engine, which supports the execution of the specified transformation rules. These engines do *not* contain conflict resolution mechanisms, thus they allow for exactly one rule to match a certain pattern in the source model. In effect, each source model can be transformed to exactly one target model, as illustrated in Figure 1 for a transformation from PIM to PSM according to MDA.

We consider this a major restriction, since software design (and implementation), in general, have to take many alternatives into account (Eramo et al., 2012). Ideally, the "best" design resulting from all possible alternatives should be taken. While software

[1] http://www.omg.org/mda/
[2] http://www.eclipse.org/atl/
[3] http://www.omg.org/spec/QVT/1.1/
[4] http://www.omg.org/mof/
[5] http://www.uml.org/

Raneburger D., Popp R. and Kaindl H..
Model-driven Transformation for Optimizing PSMs - A Case Study of Rule Design for Multi-device GUI Generation.
DOI: 10.5220/0004490104960503
In *Proceedings of the 8th International Joint Conference on Software Technologies* (ICSOFT-PT-2013), pages 496-503
ISBN: 978-989-8565-68-6

design as specified in practice may typically suggest that there is only the documented solution, many alternatives are typically considered and discussed.

Automated generation of graphical user interfaces (GUIs) can also apply MDA, typically starting from high-level interaction models (e.g., Task Models according to (Paternò et al., 1997) and UsiXML[6] or Discourse-based Communication Models (Falb et al., 2006; Popp and Raneburger, 2011)) and transforming them over different levels of abstraction to the source code of a GUI. Such approaches also apply transformation rules to transform the corresponding models between different levels of abstraction (Calvary et al., 2003).

For automatically generated GUIs according to such an approach, the restriction through a single-threaded transformation is even more obvious, since it is a major reason for the less than optimal *usability* visible to and experienced by the end-user. Multi-device GUI generation from a single source model obviously needs at least alternatives for each generated GUI, e.g., for a PC, a tablet PC and a smartphone.

While several GUIs may be achieved through a specific set of transformation rules for each GUI, we strive for a single set of transformation rules for all supported devices. They should define a kind of search space, where a heuristic search can find optimal PSMs according to given optimization criteria and constraints posed by a given device (Platform). Figure 2 sketches this approach of model-driven transformation with alternatives, where the Optimal PSMs correspond to different Platforms. The optimization happens in the course of the Transformation indicated through the rounded box, which Figure 3 zooms into for further illustration of this optimization approach to be explained below. A transformation engine that allows for matching more than one rule for the same source model pattern as required for this kind of optimization was developed by (Popp et al., 2012). While this engine is specific to GUI generation, we think that this optimization approach is generally applicable to model-driven software generation.

This paper presents our approach to model-driven transformation including PSM optimization. A case study investigates the important aspect of designing transformation rules in such a context. In particular, we show characteristics that need to be considered when designing transformation rules that support multi-device GUI generation, where device specifications are given as separate Platform Models. In this way, the transformation rules can be defined independently from concrete device properties.

[6]USer Interface eXtensible Mark-up Language – `http://www.usixml.eu/`

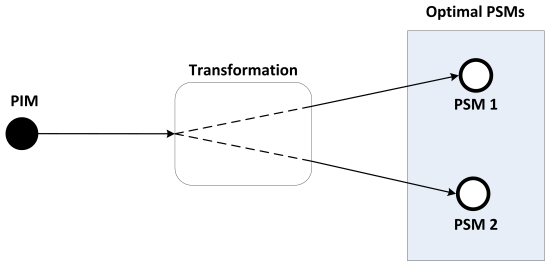

Figure 2: Model-driven Transformation with Alternatives for Optimization.

The remainder of this paper is organized in the following manner. First, we discuss the state of the art, followed by a presentation of our new conceptual approach to PSM optimization. Then we provide some background material for our case study in the context of GUI generation, in order to make this paper self-contained. Based on that, we elaborate on our case study on rule design for multi-device GUI generation and include lessons learned.

2 STATE OF THE ART

The currently available transformation engines for ATL and QVT support only matching one rule per source model pattern. (Wagelaar et al., 2010) present a solution for this problem called Module Superimposition. Their approach is based on ATL and supports overriding transformation rules. By overriding, they mean replacing the original rule with a new one, whereby it is not possible to refer to the original rule anymore. Using module superimposition in our conceptual approach, however, has the drawback that the rule set would need to be changed during the transformation process. If the rule set is modified before the transformation, there is no difference to the standard approach, because there would be exactly one rule for each source model pattern.

A GUI generation framework that supports multi-device GUI generation using ATL is UsiComp (García Frey et al., 2012). This Framework supports transformation rule design at run- and design-time through the designer. Thus, the device information is encoded manually in the transformation rules before the transformation is performed. Another UI generation framework supports semi-automatic multi-device UI generation based on ConcurTaskTree Models (CTT) (Paternò et al., 1997) (Paternò and Santoro, 2002). Semi-automatic means that the high-level CTT model, which is a Computation Independent Model, is tailored manually to a system-task model that already takes device characteristics like

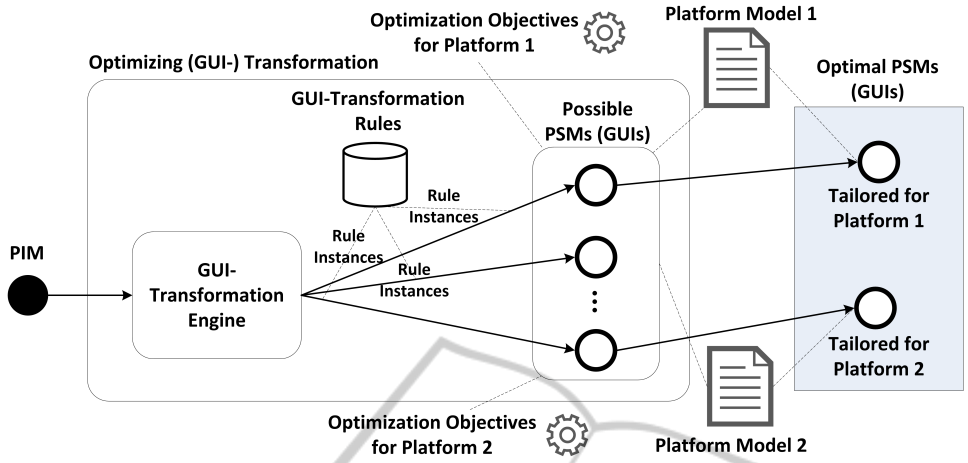

Figure 3: Model-driven Transformation including PSM Optimization.

screen size into account. Other GUI generation approaches specify their transformations in Groovy[7] or Java as part of their transformation tool (Pastor et al., 2008), which makes the modification of the transformations for device-tailoring or GUI beautification difficult (Aquino et al., 2009).

So, multi-device GUI generation typically relies on manual creation of device-dependent transformation rules or manual device-tailoring of the source model, which makes subsequent transformations with ATL or QVT sufficient.

Declarative user interface specifications are used as input for multi-target UI generation in (Gajos and Weld, 2004). The user interface adaption is treated as an optimization problem based on a user- and device-specific cost function. Compared to such user interface specifications, our interaction models are on a higher level of abstraction.

3 CONCEPTUAL APPROACH TO PSM OPTIMIZATION

To allow such a PSM optimization, a new and extended transformation approach is required. In general, some of the objectives can only be evaluated on the whole PSM as already generated, and not on a target pattern of a single rule. Therefore, in contrast to traditional model transformations, it is necessary to allow the generation of different PSMs out of one PIM. This is the case even for a single Platform. From these resulting PSMs, the one fitting best the given optimization objectives is selected.

In addition, our approach covers multi-device optimization. For each Platform, an optimal PSM is generated and selected. This actually happens for one device at a time, and one after the other.

Figure 3 illustrates the new conceptual approach. Let us focus on the case of a single Platform first. The PIM shown on the left side is transformed into several PSMs to be evaluated through an objective function for optimization according to given optimization objectives. This is illustrated inside the rounded box for “Optimizing Transformation” through “Possible PSMs”. In addition, also the concrete values of attributes specified in the Platform Model are taken into account. Some of these attributes constrain the space for optimization. One of the generated PSMs highest ranked by the objective function that fulfills the constraints is selected as the output of the optimizing transformation and is an optimal PSM, shown at the right side of Figure 3.

Our optimization objective for GUI Generation is to fit as much information as possible in a given amount of screen space. Our optimization search employs branch-and-bound techniques and does, therefore, not really explore the whole search space defined by the transformation rules. For more details on our optimization approach for GUI generation see (Raneburger et al., 2011).

For other Platforms, there are other Platform models, and there may also be different objective functions with the purpose of tailoring for different Platforms, so that several optimal PSMs may arise as indicated on the right side of Figure 3.

Note, that all these optimizations are being performed using a single set of transformation rules, even though this happens for one Platform after the other. In order to make this possible, the rules must not specifically depend on any single Platform. More precisely, the rules have to be defined independently of the concrete values of the attributes specified in the

[7] http://groovy.codehaus.org/

various Platform Models, at least for those attributes taken into account for the optimization.

As indicated for the optimization approach, different transformations need to be possible for a single given PIM, in the course of each single optimization run. This is in contrast to traditional MDA transformations. So, the rule set has to provide several rules for a given source pattern in the PIM, at least for some of these patterns. Otherwise, no search space can be generated for optimization. In order to implement the actual firing of several rules for a single pattern, a transformation engine like the one presented in (Popp et al., 2012) is necessary.

Much as in the traditional approach, however, care must be taken that no illegal PSMs are generated, with characteristics not supported by the corresponding Platforms (e.g., widgets not supported by a certain toolkit). For this reason, the transformation rules are usually filtered before their execution. Since our approach optimizes for one Platform at a time, the single rule set for all Platforms can still be filtered for the Platform at hand as usual.

4 CASE STUDY BACKGROUND

For an investigation on how transformation rules should be designed in such a context, we employed our own transformation engine, which allows for more than one rule to match the same source pattern (Popp et al., 2012). It is part of our Unified Communication Platform UI Generation Framework[8] (UCP:UI). This engine has its own transformation language and supports the transformation of Discourse-based Communication Models (Falb et al., 2006; Popp and Raneburger, 2011) to Structural UI Models (Kavaldjian et al., 2008). UCP:UI furthermore allows the optimization of the resulting Structural UI Models according to optimization objectives (Raneburger et al., 2011) and so we selected the domain of GUI generation for our case study and implemented our transformation rules using UCP:UI. We present background information on Discourse-based Communication Models, Structural UI Models, and the Unified Communication Platform GUI Generation Module, in order to make this paper self-contained.

4.1 Discourse-based Communication Models

Discourse-based Communication Models specify high-level communicative interaction of the user with the application, primarily based on discourses in the sense of dialogues. For the purpose of this paper, these models are PIMs in the sense of MDA and do not consider any device or modality characteristics. A small excerpt of a Communication Model of a simple hotel booking application is shown in Figure 4. This excerpt models the interaction between the user and the system during the collection of payment data (i.e., contact person details, billing address and credit card information). The interacting agents (User and System) are depicted in the upper left corner of Figure 4.

The basic building-blocks of Discourse-based Communication Models are Communicative Acts (CAs), depicted as rounded rectangles in Figure 4. Each CA is assigned to an agent, represented through its fill color (green/dark for User, and yellow/light for System). *Adjacency Pairs* model typical turn-takings in a conversation (e.g., Question–Answer or Offer–Accept/Reject). Such Adjacency Pairs are represented through diamonds as shown in Figure 4 and relate one opening and zero to two closing CAs.

Additional Discourse Relations, like the *Ordered-Joint* relation, can be used to link such Adjacency Pairs and to model more complex discourse structures. The OrderedJoint relation, as depicted in Figure 4 links two or more Adjacency Pairs and specifies that all Adjacency Pairs may be executed concurrently. If all branches are performed in the same presentation unit (i.e., a screen) the order is used to identify the order of presentation (i.e., the position of the branches on a single screen). In the case it is not possible to perform all branches concurrently, i.e., the screen is too small to display all branches together, the relation defines the order for displaying the branches, e.g., the order of screens of a GUI.

Discourse-based Communication Models refer to a Domain-of-Discourse Model, which specifies the concepts that the two interacting agents can "talk about". A Domain-of-Discourse Model can be specified by an Ecore or UML class diagram. For the HotelBooking Excerpt it needs to specify a `Person`, an `Address` and a `CreditCard` concept. The connection between the Discourse and the Domain-of-Discourse Model is established through the propositional content specified for a Communicative Act (for details see (Popp and Raneburger, 2011)).

4.2 Structural UI Models

For our case study, *Structural UI Models* are the PSMs. The Structural UI Models are used to define the screens of a GUI on the level of a *Concrete User Interface* according to the Cameleon Reference Model (Calvary et al., 2003). Each of these models is

[8] `http://ucp.ict.tuwien.ac.at`

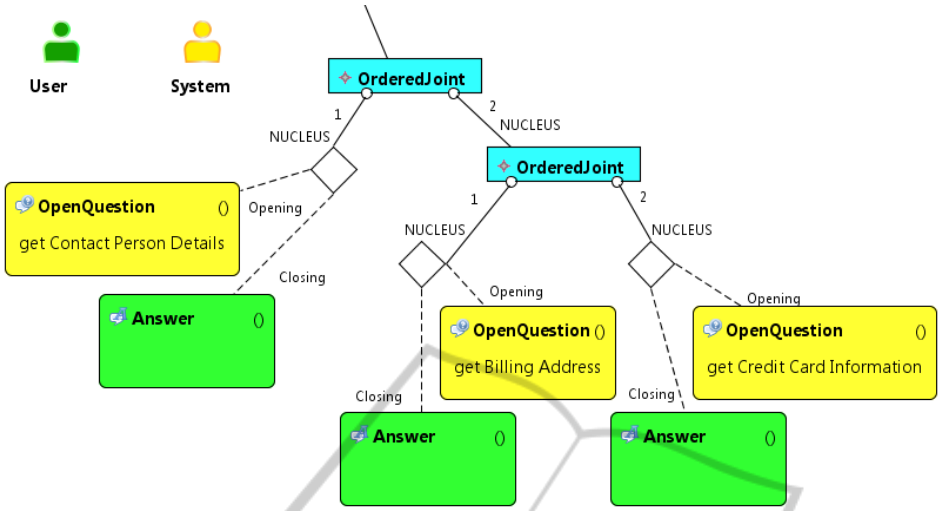

Figure 4: HotelBooking Excerpt.

Platform-specific, but independent of the used graphical toolkit and consists of widgets. As a widget is independent of the used toolkit here, it is an abstraction of an implemented "real world" widget. Such a widget can either be a container, grouping some other widgets, or a simple widget, like a *textbox* or a *label*. We distinguish between two types of containers, one displaying its contained widgets together and one displaying each contained widget at a time. The second one is used to define multiple screens of a GUI. An example for the second one is a *TabControl*, which is used to define a separation into multiple screens in the case that the screen does not provide enough space for displaying all of them together on one screen.

4.3 The Unified Communication Platform GUI Generation Module

The GUI generation module of the Unified Communication Platform UCP:UI transforms Discourse-based Communication Models to device-tailored, toolkit-independent Structural UI Models (Raneburger et al., 2011), using the transformation engine developed by (Popp et al., 2012). Thus, we used the corresponding transformation language for the implementation of a complete transformation rule set.

The left-hand-side (LHS) of these transformation rules is a Communication Model Pattern and the right-hand-side (RHS) is a Structural UI Model pattern. The UCP:UI transformation engine has the unique feature that it allows for more than one rule with the same LHS to match. The selection of which rule is matched for which element depends on target device characteristics (i.e., the Platform Model) and an objective function.

The transformation engine in UCP:UI builds each possible Structural UI Model step by step, controlled by the optimization engine. The optimization engine further evaluates the generated Structural UI Models and selects the highest ranked one. So the two parts of the UCP:UI module together implement such an optimizing transformation. Let us emphasize again, that the optimization search implemented here employs branch-and-bound techniques and does, therefore, not really explore the whole search space defined by the transformation rules.

5 CASE STUDY OF RULE DESIGN

Now let us present our case study of rule design for multi-device GUI generation. In the given context, we designed transformation rules for allowing PSM optimization according to our conceptual approach. Instead of creating Platform-specific transformation rules or tailoring the input model manually to the Platform, we developed and implemented a rule set that supports the transformation of one PIM to several PSMs. These rules are per se not tailored to any specific Platform. Which of these PSMs is optimal only depends on the given optimization objectives, which tailor the PSM to the Platform, and the corresponding constraints specified in the Platform Model.

As usual, the transformation rules enrich the models with information while concretizing them over various levels of abstraction. Normally the transformations are filtered before the transformation according to Platform characteristics, so that exactly one rule matches each source pattern. Thus, they already contain Platform-specific aspects and have to be created

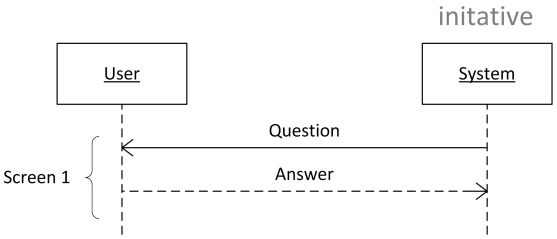

Figure 5: System Initiative.

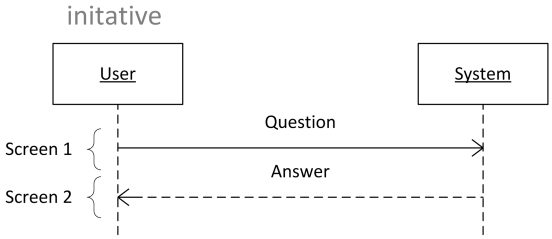

Figure 6: User Initiative.

anew or at least adapted for a new Platform.

A key challenge was to design a "minimal" set of transformation rules that supports the generation of at least one PSM for each *compliant* PIM. This means that the PIM is compliant to its metamodel (e.g., specified in UML or Ecore) and satisfies all constraints that are specified in addition for this metamodel (e.g., in OCL).

Another key challenge regarding the design of transformations for our new conceptual approach as explained above was to design "additional" transformation rules that match the same source model pattern to enable the transformation of one PIM into several PSMs. An example is whether a rule splits the resulting screen or not, which relates to an optimization objective that a minimum number of clicks shall be achieved. More on such optimization objectives can be found in (Raneburger et al., 2011).

5.1 Towards Completeness of the Rule Set

While it is hard to formally define completeness of such a rule set, we tried to design the rule set in such a way, that no rules are obviously lacking. Our first consideration was about the granularity of units to be transformed. This led us to identifying atomic transformation units. In addition, we had to make sure that there were alternative transformations possible.

5.1.1 Identifying Atomic Transformation Units

To ensure that each PIM can be transformed with the available rules, it is important to identify the atomic transformation units, like in traditional transformations. The elements in Discourse-based Communication Models (PIMs) are CAs, Adjacency Pairs and Discourse Relations. We distinguish between relations whose metamodel element has a defined number of child links and relations whose metamodel element has an undefined number of links.

Both the metamodels of the Discourse-based Communication Model (the PIM in our case study) and of the Structural UI Model (the PSM) define these atomic transformation units. Every Discourse Relation is itself an atomic transformation unit, because it can be mapped to a combination of some container widgets.

For Adjacency Pairs, such a definition is more difficult. Let us look at the Adjacency Pair OpenQuestion–Answer at the left side of Figure 4. In this case the System asks the User a question and the User has to answer it. In a GUI the widgets for the two Communicative Acts are usually combined and, therefore, the PSM defines the atomic transformation unit as the whole Adjacency Pair (see Figure 5). In the case the User requests something from the System, the situation is different: the widgets for sending the request and for the response of the System are usually displayed on different screens (see Figure 6). Therefore, it is possible to transform each CA on its own, so that in such a case the CA is the atomic transformation unit. So, the definition of an atomic transformation unit depends here on the initiative in the Adjacency Pair.

5.1.2 Taking Platform Restrictions into Account

In addition, it was necessary to take Platform restrictions into account, since not all the Platforms have the same characteristics, e.g., not all devices have the same widget set. Since the GUI generation (including optimization) will be done for each Platform at a time, we looked into each Platform Model separately for its restrictions as related to the rule set. Before a generation run, rules will be filtered out that would not lead to a legal PSM. In the course of the rule design, therefore, we had to make sure that for each such case still at least one remaining rule exists for each atomic transformation (see above).

Another example for such a restriction is the input method to be used on a given device. On a touchscreen to be used with finger pointing, for instance, widgets too small for that are not to be used. Therefore, all rules generating such widgets will be filtered out, and other rules need to be available (for each compliant PIM, in principle).

5.2 Transformations with Alternatives

As described above, we designed a "minimal" rule set first, for transformations to one PSM per given Platform Model. Table 1 shows the numbers of rules of this set, broken down in subcategories. For optimizing PSMs according to our approach, however, it was necessary to provide additional rules for generating alternatives.

Table 1: Numbers of Transformation Rules.

Category Name	Minimal Rules	Additional Rules
Communicative Act Rules	28	-
Relation Rules	23	10
Adjacency Pair Rules	52	4
Domain-of-Discourse Rules	19	-
	122	14

Table 1 also shows numbers of such additional rules for generating alternatives that we created. We defined ten more rules having the same source pattern for Relations as already defined rules, and four more rules having the same source pattern for Adjacency Pairs as already defined rules. In the following, we elaborate on these additional rules.

5.2.1 Additional Relation Rules

Relation rules typically create containers for their children. According to the behavior definition of a given Relation, its children are either displayed concurrently, leading to a `Panel`, or in different screens, leading to a `Choice` element (e.g., a TabControl). All the ten additional Relation Rules for alternative transformations match Relations whose children may, in principle, be displayed concurrently, but these additional rules split the screen. Thus, their respective RHSs have different impact on the resulting PSM (i.e., Structural UI Model) than the one of the rule in our "minimal" rule set. In fact, a split screen means at least one additional click but reduces the amount of scrolling.

Let us illustrate this with a small example, taking the `OrderedJoint` relation from our Hotelbooking Communication Model. All children of an OrderedJoint may be displayed concurrently, in principle. Figure 7 shows the basic rule that matches an OrderedJoint relation and creates a `Panel` containing the OrderedJoint children. Thus, the resulting GUI will show them concurrently.

Figure 8 shows another rule that matches an OrderedJoint relation, but creates a `TabControl`, which contains its children instead. So, less space is required in the resulting GUI than in the one generated with the

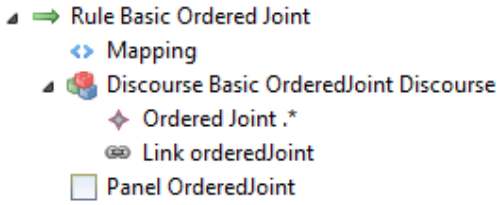

Figure 7: OrderedJoint Rule Large.

rule above, as the screen is split. How much space is saved in the end depends on the children and, therefore, on the Communication Model.

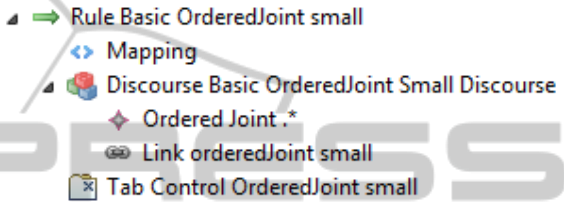

Figure 8: OrderedJoint Rule Small.

5.2.2 Additional Adjacency Pair Rules

For the additional four Adjacency Pair Rules with LHSs matching the same PIM pattern as some other rule each, the approach was analogous. In particular, these rules generate drop-down lists that show only one list entry per default instead of displaying more than one entry at the same time, using radio buttons for their selection. It is important to note that these rules do not discard any information (which would be another way to save space).

As an example, we used the presented rule set to generate GUIs for a simple flight booking application for different devices (available at `http://ontoucp.ict.tuwien.ac.at/UI/FlightBooking`).

5.3 Lessons Learned

Now let us present a few lessons learned that we generalize from our concrete experience gained in the course of this case study:

- Adding additional rules to the "minimal" rule set allows optimization, even when adding a small number of rules (compared to the number of rules in the "minimal" rule set).
- The design of rules independent of a concrete Platform allows the optimization of several PSMs with the same rules.
- When such a rule set works for a certain PSM (for a smartphone), it can at least to a large extent be reused for a similar PSM (for a tablet PC).

6 CONCLUSIONS

In this paper, we propose an approach for optimizing PSMs using model-driven transformations. Since this approach selects from several generated PSMs (according to given optimization objectives), it requires an engine supporting that more than one transformation rule can fire on the same source pattern. Of course, it also requires that at least some rules exist where this actually happens.

In a case study in the context of model-driven generation of graphical user interfaces, we studied rule design for such an approach. Such a rule design has its own intricacies, but we even found possibilities for reuse of rules for generating several PSMs (for GUIs on several devices). On an existing platform that supports this approach, optimization for several such GUIs was actually possible with the rule set designed.

ACKNOWLEDGEMENTS

Part of this research has been carried out in the GENUINE project (No. 830831), another part in the ProREUSE project (No. 834167), both funded by the Austrian FFG.

REFERENCES

Aquino, N., Vanderdonckt, J., Valverde, F., and Pastor, O. (2009). Using profiles to support model transformations in the model-driven development of user interfaces. In Lopez Jaquero, V., Montero Simarro, F., Molina Masso, J. P., and Vanderdonckt, J., editors, *Computer-Aided Design of User Interfaces VI*, pages 35–46. Springer London.

Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J. (2003). A unifying reference framework for multi-target user interfaces. *Interacting with Computers*, 15(3):289 – 308. Computer-Aided Design of User Interface.

Eramo, R., Malavolta, I., Muccini, H., Pelliccione, P., and Pierantonio, A. (2012). A model-driven approach to automate the propagation of changes among architecture description languages. *Software & Systems Modeling*, 11(1):29–53.

Falb, J., Kaindl, H., Horacek, H., Bogdan, C., Popp, R., and Arnautovic, E. (2006). A discourse model for interaction design based on theories of human communication. In *Extended Abstracts on Human Factors in Computing Systems (CHI '06)*, pages 754–759. ACM Press: New York, NY.

Gajos, K. and Weld, D. S. (2004). SUPPLE: Automatically generating user interfaces. In *Proceedings of the 9th International Conference on Intelligent User Interface (IUI '04)*, pages 93–100, New York, NY, USA. ACM Press.

García Frey, A., Céret, E., Dupuy-Chessa, S., Calvary, G., and Gabillon, Y. (2012). Usicomp: an extensible model-driven composer. In *Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems*, EICS '12, pages 263–268, New York, NY, USA. ACM.

Kavaldjian, S., Bogdan, C., Falb, J., and Kaindl, H. (2008). Transforming discourse models to structural user interface models. In *Models in Software Engineering, LNCS 5002*, volume 5002/2008, pages 77–88. Springer, Berlin / Heidelberg.

Pastor, O., España, S., Panach, J. I., and Aquino, N. (2008). Model-driven development. *Informatik Spektrum*, 31(5):394–407.

Paternò, F., Mancini, C., and Meniconi, S. (1997). ConcurTaskTrees: A diagrammatic notation for specifying task models. In *Proceedings of the IFIP TC13 Sixth International Conference on Human-Computer Interaction*, pages 362–369.

Paternò, F. and Santoro, C. (2002). One model, many interfaces. In *Proceedings of the 4th International Conference on Computer-Aided Design of User Interfaces (CADUI 2002)*, pages 143–154.

Popp, R., Falb, J., Raneburger, D., and Kaindl, H. (2012). A transformation engine for model-driven UI generation. In *Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems*, EICS '12, pages 281–286, New York, NY, USA. ACM.

Popp, R. and Raneburger, D. (2011). A High-Level Agent Interaction Protocol Based on a Communication Ontology. In Huemer, C., Setzer, T., Aalst, W., Mylopoulos, J., Sadeh, N. M., Shaw, M. J., and Szyperski, C., editors, *E-Commerce and Web Technologies*, volume 85 of *Lecture Notes in Business Information Processing*, pages 233–245. Springer Berlin Heidelberg. 10.1007/978-3-642-23014-1_20.

Raneburger, D., Popp, R., Kavaldjian, S., Kaindl, H., and Falb, J. (2011). Optimized GUI generation for small screens. In Hussmann, H., Meixner, G., and Zuehlke, D., editors, *Model-Driven Development of Advanced User Interfaces*, volume 340 of *Studies in Computational Intelligence*, pages 107–122. Springer Berlin / Heidelberg.

Truyen, F. (2006). The Fast Guide to Model Driven Architecture - The basics of Model Driven Architecture.

Wagelaar, D., Van Der Straeten, R., and Deridder, D. (2010). Module superimposition: a composition technique for rule-based model transformation languages. *Software and Systems Modeling*, 9:285–309. 10.1007/s10270-009-0134-3.