A Model-Driven Approach to the Engineering of Multiple User Interfaces

Goetz Botterweck
University of Koblenz-Landau
Universitätsstr. 1
D-56070 Koblenz
+49 261 2872531
botterweck@uni-koblenz.de

ABSTRACT

In this paper, we describe MANTRA[1], a model-driven approach to the development of multiple consistent user interfaces for one application. The common essence of these user interfaces is captured in an abstract UI model (AUI) which is annotated with constraints to the dialogue flow. We consider in particular how the user interface can be adapted on the AUI level by deriving and tailoring dialogue structures which take into account constraints imposed by front-end platforms or inexperienced users. With this input we use model transformations described in ATL (Atlas Transformation Language) to derive concrete, platform-specific UI models (CUI). These can be used to generate implementation code for several UI platforms including GUI applications, dynamic websites and mobile applications. The generated user interfaces are integrated with a multi tier application by referencing WSDL-based interface descriptions and communicating with the application core over web service protocols.

1. INTRODUCTION

An elementary problem in user interface engineering is the complexity imposed by the diversity of platforms and devices which can be used as foundations. The complications increase when we develop multiple user interfaces (based on different platforms) which offer access to the same functionality. In that case we have to find a way to resolve the inherent contradiction between redundancy (the user interfaces of one application have something in common) and variance (each user interface should be optimized for its platform and context of use).

Model-driven approaches appear to be a promising solution to this research problem, since we can use models to capture the common features of all user interfaces and model transformations to produce multiple variations from that. The resulting implementations can be specialized (because we can embed platform-specific implementation knowledge into the transformations) as well as consistent (as they are all derived from the same common model and hence share the same logical structure).

2. RELATED WORK

The *mapping problem* [11], a fundamental challenge in model-based approaches, can occur in various forms and can be dealt with by various types of approaches [3]. One instance of this is the question of how we can identify concrete interaction *elements* that match a given abstract element and other constraints [13].

A similar challenge is the derivation of *structures* in a new model based on information given in another existing model. Many task-oriented approaches use requirements given by the task model to determine UI structures; for example, temporal constraints similar to the ones in our approach have been used to derive the structure of an AUI [9] or dialogue model [6].

Florins et al. [5] take an interesting perspective on a similar problem by discussing rules for splitting existing presentations into smaller ones. That approach combines information from the AUI and the underlying task model – similar to our approach using an AUI annotated with temporal constraints which are also derived from a task model.

Many model-driven approaches to UI engineering have proposed a hierarchical organization of interaction elements grouped together into logical units (e.g., [4]). A number of approaches to multiple user interfaces has been collected in [12].

3. ABSTRACT DESCRIPTION OF USER INTERFACES

The MANTRA model flow (cf. Figure 1) is structured vertically by abstraction levels similar to the CAMELEON framework [2]. The goal of our process (in Figure 1 going from top to bottom) is to create several user interfaces (front-ends) for the functionality provided by the core of that application.

Further steps are illustrated by a simple time table application. Figure 2 shows the corresponding AUI model. The user can search for train and bus connections by specifying several search criteria like departure and destination locations, time of travel or the preferred means of transportation (lower part of Figure 2). The matching connections are retrieved by a web service operation and displayed in a separate presentation (upper right part of Figure 2)

At first, this model only contains UI elements (▭) and UI composites (⬭) organized in a simple aggregation hierarchy (indicated by ◇— relations) and the web service operation necessary to retrieve the results. This model is the starting point of our approach (cf. result of ❶ in Figure 1) and captures the common essence of the multiple user interfaces of the application in one abstract UI. This AUI contains platform-independent interaction concepts like “Select one element from a list” or “Enter a date”.

The AUI is then further annotated by dialogue flow constraints based on the temporal relationships of the ConcurTaskTree approach [10]. For instance we can describe that two interaction elements have to be processed sequentially (>>) or have to be processed, but can be processed in any order (|=|).

[1] Model-based engineering of multiple interfaces with transformations

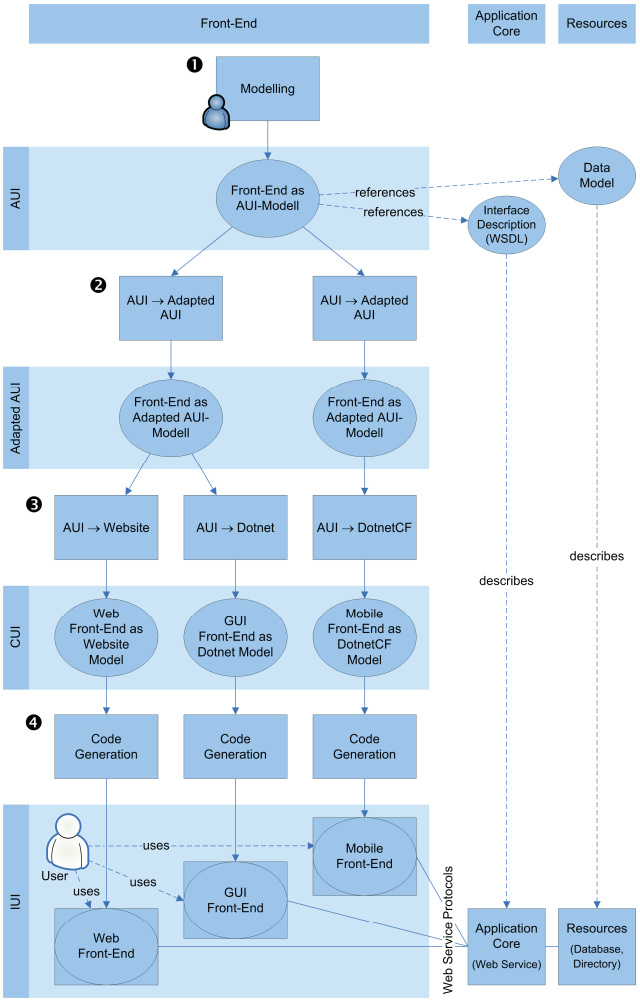

Figure 1. Model flow in the MANTRA approach.

4. ADAPTING ON THE AUI LEVEL

As a next step (❷ in Figure 1) we augment the AUI by deriving dialogue and presentation structures. These structures are still platform-independent. However, they can be adapted and tailored to take into account constraints imposed, for instance, by platforms with limited display size or by inexperienced users.

4.1 Clustering Interaction Elements to Generate Presentation Units

First we cluster UI elements by identifying suitable UI composites. The subtrees starting at these nodes will become presentations in the user interface ( ). For instance we decided that "Time of Travel" and all UI elements below it will be presented coherently. This first automatic clustering is done by heuristics based on metrics like the number of UI elements in each presentation or the nesting level of grouping elements. To further optimize the results the clustering can be refined by the human designer.

4.2 Inserting Control-Oriented Interaction Elements

Secondly, we generate the navigation elements necessary to traverse between the presentations identified in the preceding step. For this we create triggers ( ). These are abstract interaction elements which can start an operation (OperationTrigger) or the transition to a different presentation (NavigationTrigger). In graphical interfaces these can be represented as buttons, in other front-ends they could also be implemented as speech commands.

To generate NavigationTriggers in a presentation p we calculate `dialogueSuccessors(p)` which is the set of all presentations which can "come next" if we observe the temporal constraints. We can then create NavigationTriggers (and related Transitions) so that the user can reach all presentations in `dialogueSuccessors(p)`. In addition to this we have to generate `OperationTriggers` for all presentations which will trigger a web service operation, e.g., "Search" to retrieve matching train connections (lower right corner of Figure 2).

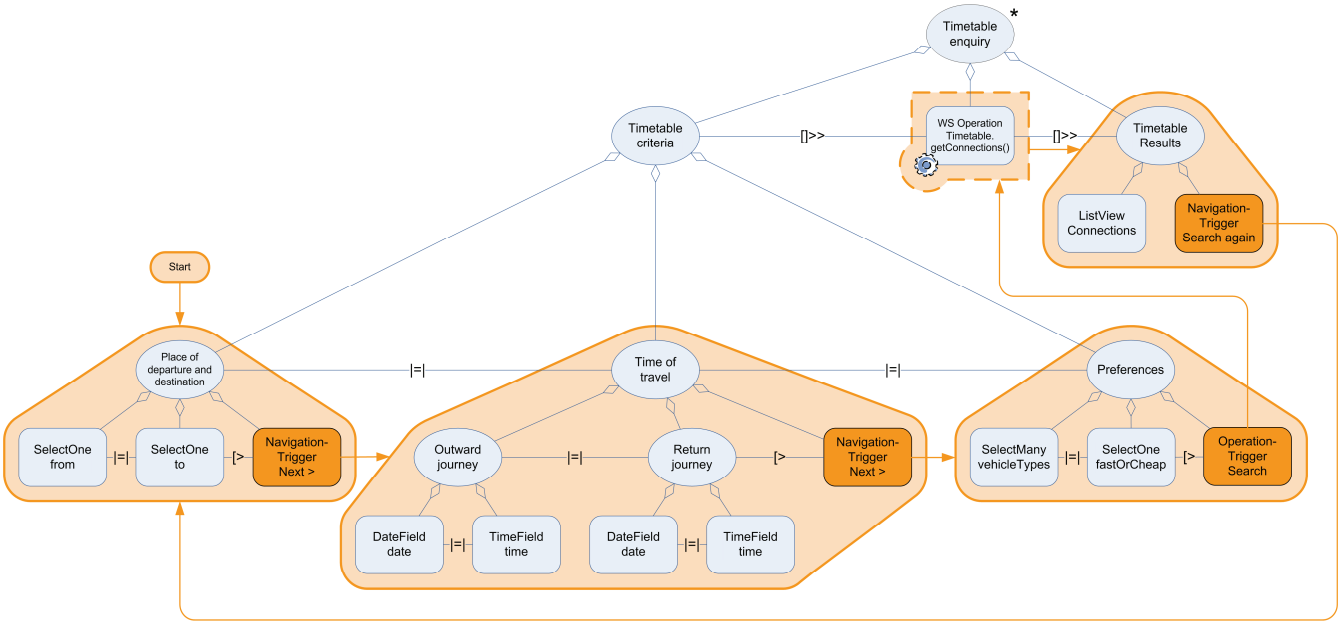

Figure 2. Adopted AUI model of the sample application, already annotated by presentations and triggers.

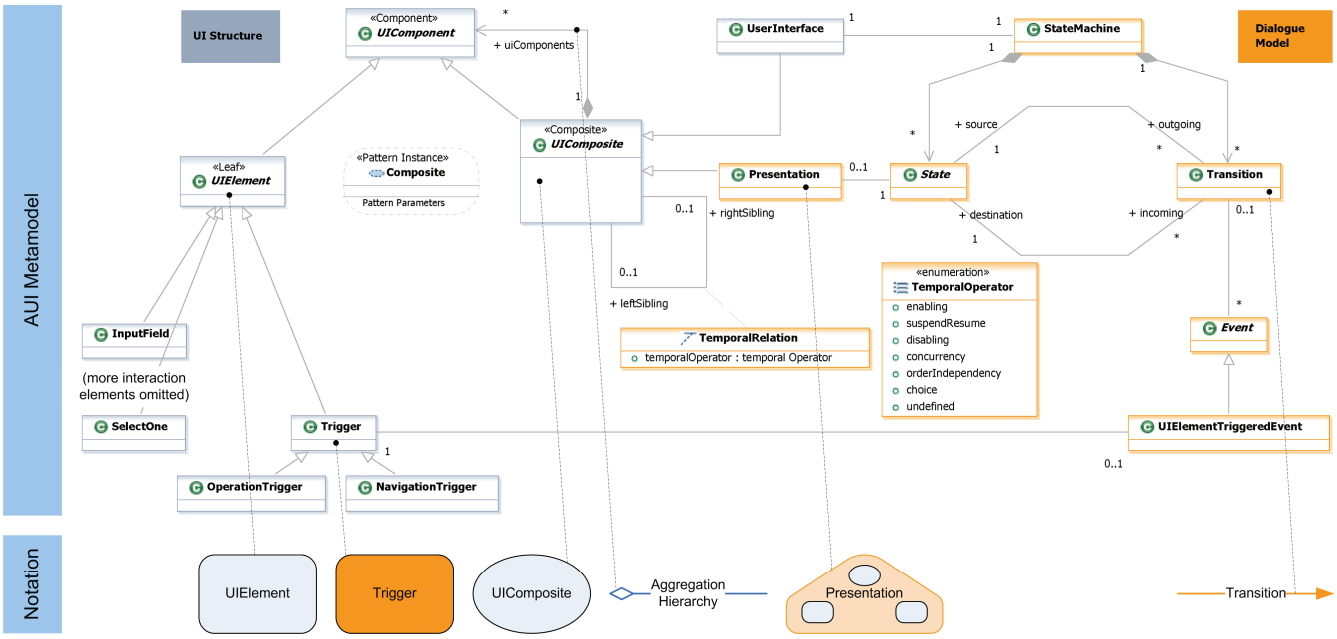

Figure 3. Simplified excerpt from the AUI metamodel and the related notation symbols.

These two adaptation steps (identification of presentations, insertion of triggers) are implemented as ATL model transformations. These result in the AUI (blue symbols in Figure 2) augmented with dialogue structures (orange symbols) which determine the paths a user can take through our application.

It is important to note that the dialogue structures are not fully determined by the AUI. Instead, we can adapt the AUI according to the requirements and create different variants of it (cf. results of step ❷). For instance, we could get more (but smaller) presentations to facilitate viewing on a mobile device – or we could decide to have large coherent presentations, taking the risk that the user has to do lots of scrolling if restricted to a small screen.

4.3 Selecting Content

As an additional adaptation step we can filter content retrieved from the web service based on priorities. For instance, if a user has a choice, higher priority is given to knowing when the train is leaving and where it is going before discovering whether it has a restaurant. This optional information can be factored out to separate "more details" presentations.

A similar concept are substitution rules which provide alternative representations for reoccurring content. A train, for example, might be designated as InterCityExpress, ICE, or by a graphical symbol based on the train category (e.g., ⁂) depending on how much display space is available. These priorities and substitution rules are domain knowledge which cannot be inferred from other models. The necessary information can therefore be stored as annotations to the underlying data model.

5. GENERATING CONCRETE AND IMPLEMENTED USER INTERFACES

Subsequently we transform the adapted AUI models into several CUIs using a specialized model transformation (❸) for each target platform. These transformations encapsulate the knowledge of how the abstract interaction elements are best transformed into platform-specific concepts. Hence, they can be reused for other applications over and over again.

As a result we get platform-specific CUI models. These artefacts are still represented and handled as models, but use platform-specific concepts like "HTML-Submit-Button" or ".NET GroupBox". This makes it easier to use them as a basis for the code generation (❹) which produces the implementations of the desired user interfaces in platform-typical programming or markup languages.

6. APPLIED TECHNOLOGIES

We described the metamodels used in MANTRA (including platform-specific concepts) in UML and then converted these to Ecore, since we use the Eclipse Modeling Framework (EMF) [1] to handle models and metamodels.

The various model transformations (e.g. for steps ❷ and ❸) are described in ATL [8]. On the one hand, the integration of ATL with Eclipse and EMF was helpful as it supported the development in an integrated environment which was well-known to us. On the other hand, the work with ATL model transformations turned out to be time consuming; for instance, ATL was sensitive even to small mistakes and then often did not provide helpful error messages.

We use a combination of Java Emitter Templates and XSLT to generate (❹) arbitrary text-oriented or XML-based implementation languages (e.g., C# or XHTML with embedded PHP).

The coordination of several steps in the model flow is automated by mechanisms provided by the Eclipse IDE and related tools, e.g., we use the software management tool `Apache Ant` [7] (which is integrated in Eclipse) and custom-developed "`Ant Tasks`" to manage the chain of transformations and code generation.

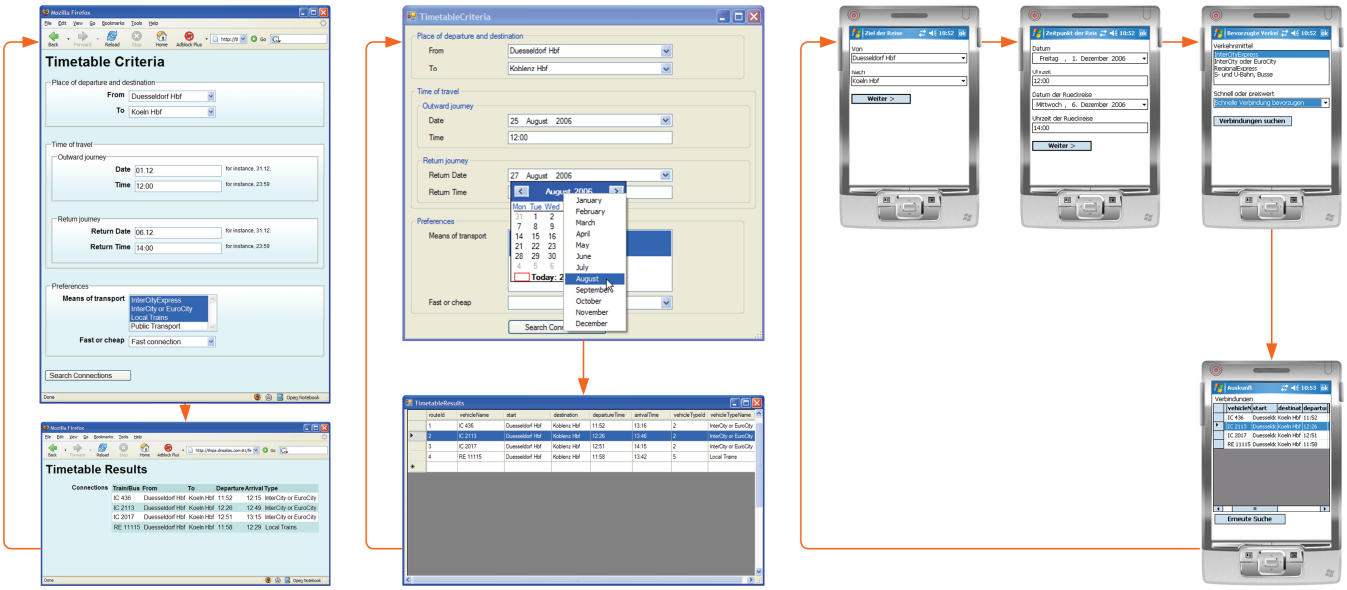

Figure 4. The generated front-ends (Web, GUI, mobile).

We use web services as an interface between the UIs and the application core. Hence, the UI models reference a WSDL based description of operations in the application core. The generated UIs then use web service operations, e.g., to retrieve results for a query specified by the user.

7. CONCLUSION

We have shown how our MANTRA approach can be used to generate several consistent user interfaces for a multi tier application (cf. Figure 4).

At the moment, the *automated* model flow (cf. Figure 1) starts at the AUI level. But nothing prevents us from starting with a task model (e.g., in CTT) and then either manually transferring the task structures into an AUI model, or extending the automated model flow to support task models from which the annotated AUI model can be derived.

We discussed how the user interface can be adapted on the AUI level by tailoring dialogue and logical presentation structures which take into account requirements imposed by front-end platforms or inexperienced users. For this we used the hierarchical structure of interaction elements and constraints on the dialogue flow which can be derived from a task model.

The approach generates fully working prototypes of user-interfaces on three target platforms (GUI, dynamic website, mobile device) which can serve as front-ends to arbitrary web services.

8. ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers for their constructive and valuable feedback.

9. REFERENCES

[1] Budinsky, F., Steinberg, D., Merks, E., Ellersick, R. and Grose, T.J. *Eclipse modeling framework : a developer's guide*. Addison-Wesley, Boston, MA, USA, 2003.

[2] Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L. and Vanderdonckt, J. A unifying reference framework for multi-target user interfaces. *Interacting with Computers*, *15* (3). 289-308.

[3] Clerckx, T., Luyten, K. and Coninx, K., The mapping problem back and forth: customizing dynamic models while preserving consistency. in *TAMODIA '04*, (Prague, Czech Republic, 2004), ACM Press, 33-42.

[4] Eisenstein, J., Vanderdonckt, J. and Puerta, A.R. Applying model-based techniques to the development of UIs for mobile computers. in *IUI '01*, 2001, 69-76.

[5] Florins, M., Simarro, F.M., Vanderdonckt, J. and Michotte, B. Splitting rules for graceful degradation of user interfaces *Intelligent User Interfaces 2006*, 2006, 264-266.

[6] Forbrig, P., Dittmar, A., Reichart, D. and Sinnig, D., From Models to Interactive Systems -- Tool Support and XIML. in *IUI/CADUI 2004 workshop "Making model-based user interface design practical: usable and open methods and tools"*, (Island of Madeira, Portugal, 2004).

[7] Holzner, S. and Tilly, J. *Ant : the definitive guide*. O'Reilly, Sebastopol, CA, USA, 2005.

[8] Jouault, F. and Kurtev, I. Transforming Models with ATL *Model Transformations in Practice (Workshop at MoDELS 2005)*, Montego Bay, Jamaica, 2005.

[9] Paternò, F., One Model, Many Interfaces. in *CADUI'02*, (Valenciennes, France, 2002).

[10] Paternò, F., Mancini, C. and Meniconi, S., ConcurTaskTrees: A diagrammatic notation for specifying task models. in *Interact'97*, (Sydney, 1997), Chapman and Hall, 362-369.

[11] Puerta, A.R. and Eisenstein, J., Interactively Mapping Task Models to Interfaces in MOBI-D. in *DSV-IS 1998*, (Abingdon, UK, 1998), 261-273.

[12] Seffah, A. and Javahery, H. *Multiple user interfaces : cross-platform applications and context-aware interfaces*. J. Wiley, Hoboken, NJ, 2004.

[13] Vanderdonckt, J., Advice-Giving Systems for Selecting Interaction Objects. in *User Interfaces to Data Intensive Systems - UIDIS'99*, (Edinburgh, Scotland, 1999), 152-157.