Capturing the Requirements for Multiple User Interfaces

Return to Published Papers

Goetz Botterweck
Institute for IS Research
University of Koblenz-Landau, Germany
botterwe@uni-koblenz.de

J. Felix Hampe
School of Computing and Information Science
University of South Australia
felix.hampe@unisa.edu.au

Abstract

In this paper we describe MANTRA, a model-driven approach for the development of multiple consistent user interfaces for one application. The common requirements of all these user interfaces are captured in an abstract UI model (AUI) which is annotated with constraints on the dialogue flow.

We exemplify all further steps along a well known application scenario in which a user queries train connections from a simple timetable service.

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 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 user interfaces are integrated with a multi tier application by referencing WSDL-based (Web Service Description Language) interface descriptions.

Finally, we discuss how our approach can be extended to include voice interfaces. This imposes special challenges as these interfaces tend to be structurally different from visual platforms and have to be specified using speech-input grammars.

1. Introduction

Almost everyday we see the emergence and adaptation of new communication technologies and devices. As a result there is a whole spectrum of platforms which offer additional channels to deliver information services. More and more companies use these technology-driven opportunities to implement multi-channel applications, for instance to introduce or intensify self-service approaches where the customer has the freedom to choose which form of communication suits him best.

An elementary problem in user interface engineering related to the context of such multiple user interfaces is the complexity imposed by the diversity of platforms and devices which can be used as foundations. The complexity increases when we develop multiple user interfaces based on different platforms, which offer access to the same functionality.

When describing the requirements for such applications 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* (since 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 describe the differences and produce multiple variations from the common representation. 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.

2. Related work

The *mapping problem* [1], a fundamental challenge in model-based approaches can occur in various forms, such as model derivation, model linking, model update, and can be dealt with by various types of approaches [2]. One instance of this is the question of how we can identify concrete interaction *elements* that match a given abstract element and other constraints [3].

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

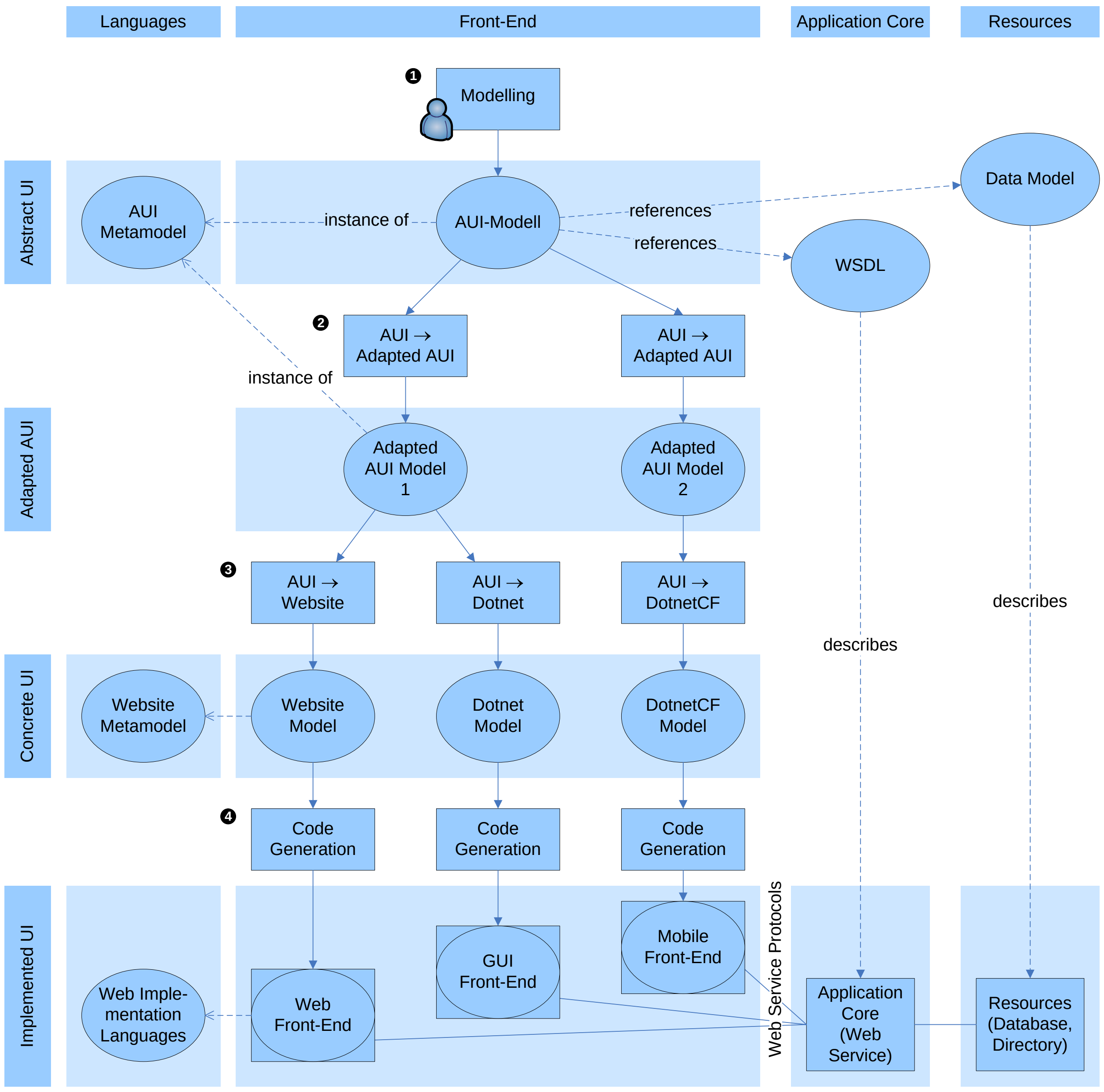


Figure 1. Model flow in the MANTRA approach.

structures; for example, temporal constraints similar to the ones in our approach have been used to derive the structure of an AUI [4] or dialogue model [5].

Florins et al. [6] 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 [7].

A number of approaches to multiple user interfaces has been collected in [8].

3. Abstract description of user interfaces

The MANTRA[1] model flow (cf. Figure 1) is structured vertically by abstraction levels similar to the CAMELEON framework [9]. 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.

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

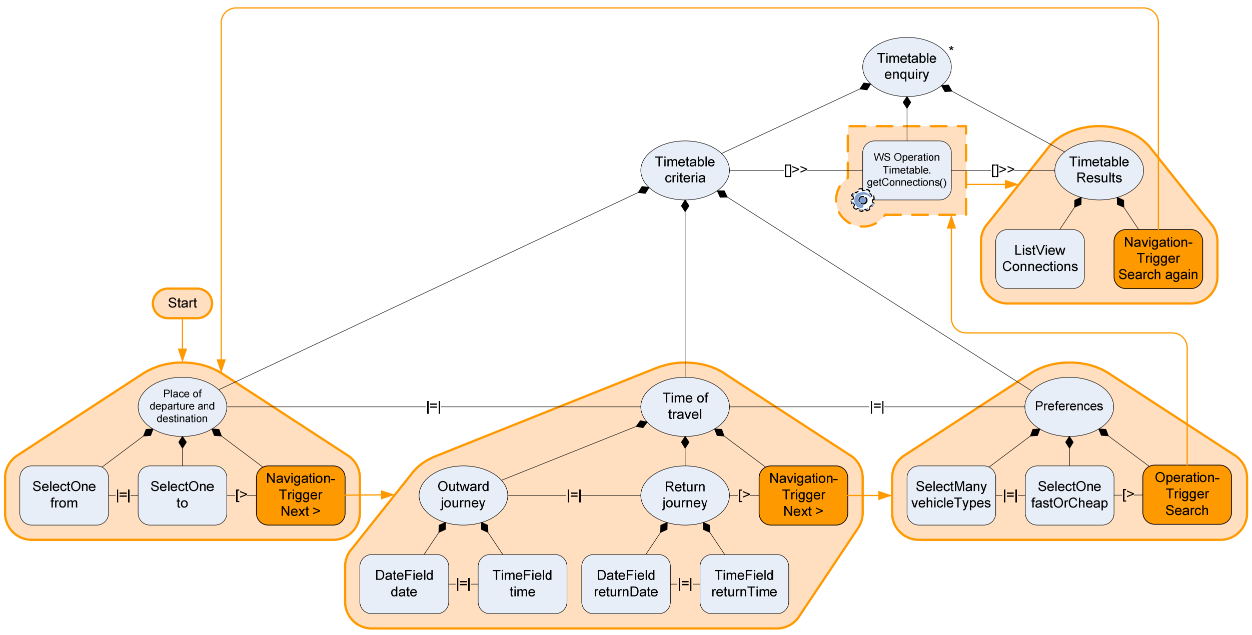


Figure 2. Adopted AUI model of the sample application, already annotated by presentations and triggers.

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 express that two interaction elements have to be processed sequentially (>>) or have to be processed, but in arbitrary order (|=|).

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 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

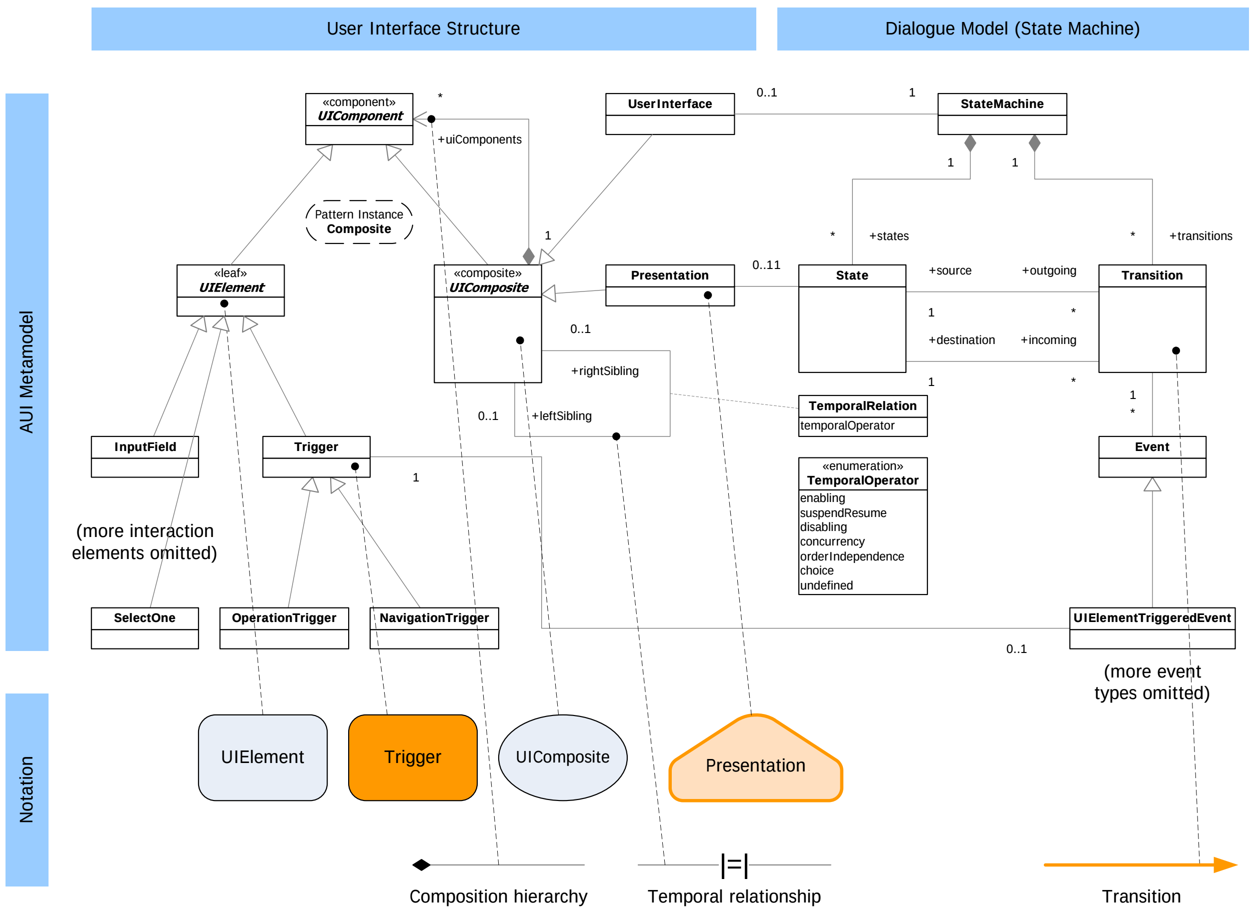


Figure 3. Simplified excerpt from the AUI metamodel and the related notation symbols.

temporal constraints. We can then create Navigation-Triggers (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).

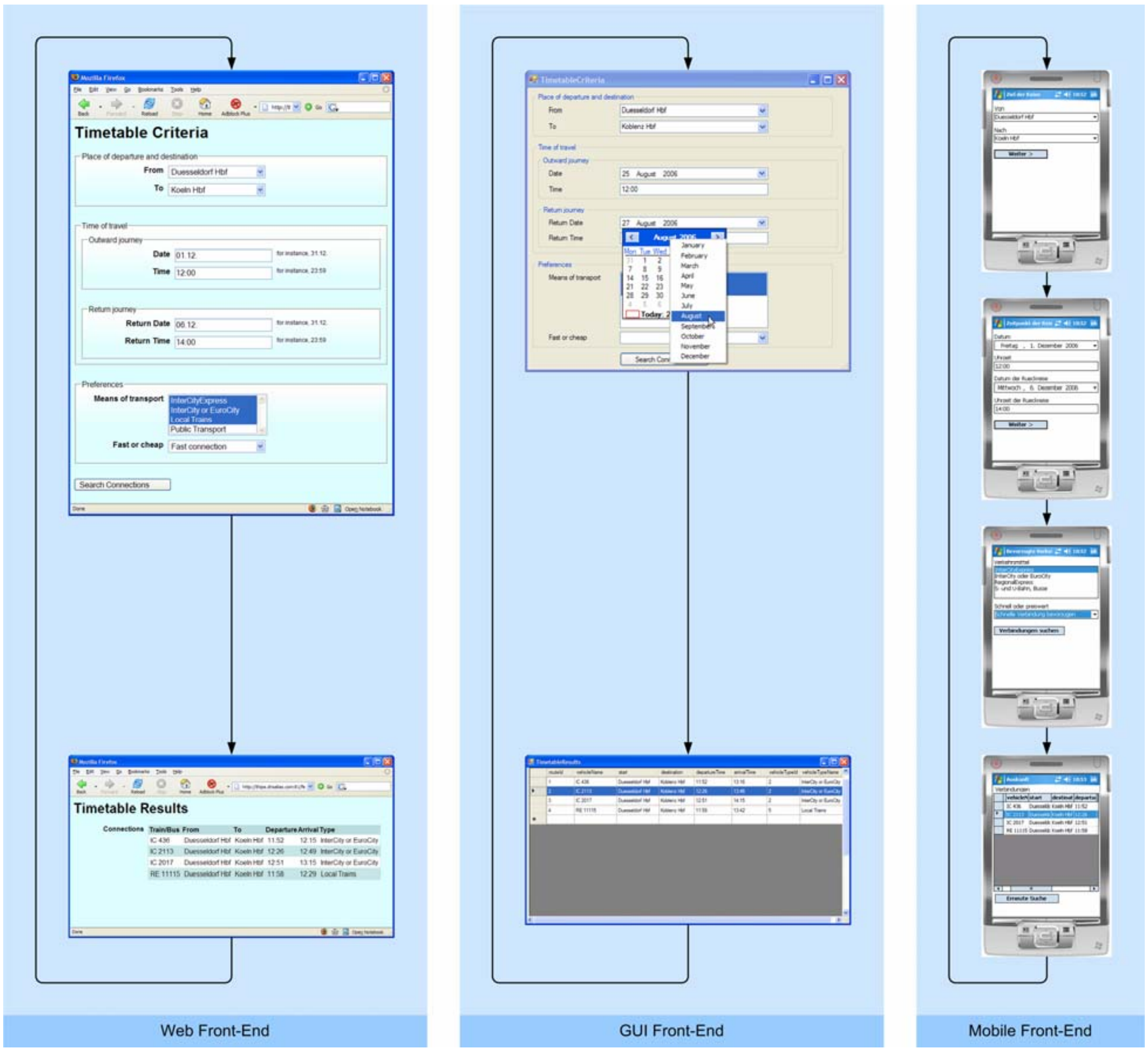
These two adaptation steps (identification of presentations, insertion of triggers) are implemented as model transformations, which are described in the transformation language ATL [11]. These result in the AUI (blue symbols in Figure 2) augmented with dialogue structures (orange symbols) which determine 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, yet 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.



Timetable Criteria

Place of departure and destination

From	Duesseldorf Hbf	
To	Koeln Hbf	

Time of travel

Outward journey

Date	01.12.	for instance, 31.12.
Time	12:00	for instance, 23:59

Return journey

Return Date	06.12.	for instance, 31.12.
Return Time	14:00	for instance, 23:59

Preferences

Means of transport	InterCityExpress, InterCity or EuroCity, Local Trains, Public Transport
Fast or cheap	Fast connection

Search Connections

Timetable Results

Connections

Train/Bus	From	To	Departure	Arrival	Type
IC 436	Duesseldorf Hbf	Koeln Hbf	11:52	12:15	InterCity or EuroCity
IC 2113	Duesseldorf Hbf	Koeln Hbf	12:26	12:49	InterCity or EuroCity
IC 2017	Duesseldorf Hbf	Koeln Hbf	12:51	13:15	InterCity or EuroCity
RE 11115	Duesseldorf Hbf	Koeln Hbf	11:58	12:29	Local Trains

Figure 4. The generated front-ends.

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 base for code generation (❹) which produces the implementations of the desired user interfaces in platform-typical programming or markup languages. Figure 4 shows screenshots of the generated implementations. Please note the congruence between the four screenshots of the mobile front-end in Figure 4 and the four presentation areas (⌓) in Figure 2.

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 EMF [12] to handle models and metamodels.

The various model transformations, such as for steps ❷ and ❸, are described in ATL [11]. 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 [13] (which is integrated in Eclipse) and custom-developed "Ant Tasks" to manage the chain of transformations and code generation.

We use web services as an interface between the UIs and the application core. Hence, the UI models reference a WSDL [14] 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. Extending the approach for voice user interfaces

7.1 Differences between visual user interfaces and voice-based user interfaces

Although our approach supports varying user interface platforms and their different characteristics, the inclusion of voice user interfaces (VUI) imposes additional challenges since there are major differences to the platforms we have considered so far [15]:

VUI are invisible. Hence, the current state of the system and interaction options are not visible to the user.

VUI are limited for one modality (sound) for both input and output. In a visual interface the user is able to enter something, for example with a keyboard, *while* he is perceiving information, like reading a hint which tells him what to enter at the same time. With a voice interface it is impossible to offer similar functionality in a usable way.

A VUI might be used in an environment which competes for a user's attention and cognitive capacity, e.g., when driving a car.

For visual interfaces, there are well-known interaction patterns which have evolved over time, such as main menu, wizard-like-process, homepage, back button and bookmark. For VUIs, however, similar patterns are still evolving and we carefully have to evaluate whether it is beneficial if we simply transfer patterns from visual to voice interfaces by introducing a "home" or "go back" command).

Last but not least, users are used to follow certain structural constraints when filling in forms on a *visual* interface, e.g., they start from the top left corner and continue downwards using visual clues like labels or group boxes to orientate themselves. However, for a *voice* interface such visual structure indicators do not exist: Where is 'top left' in a voice dialog? Hence, a voice interface designer has to provide other clues that structure the interaction: "This is the main menu. To come back here you can always say 'go to main menu'. Now you've got the following options …".

In the following two sections we will discuss two options of how we can structure the voice interaction with the user. As a basis for that discussion we will use the features of VoiceXML [16].

7.2 Strictly structured voice interaction

One option to design a VUI is to strictly structure the interaction. For instance we can closely follow the AUI structure we already presented as a foundation for visual interfaces and use voice dialogs and prompts which resemble the presentations and input controls found in the AUI Model.

In VoiceXML this is supported by forms and fields with so called field-level grammars, which describe the expected syntax of the user's utterances *field-by-field*. For instance for the AUI model in Figure 2 the interaction would start off with a dialogue asking for the place of departure, then a dialogue for the destination and so on, until all values are collected.

To create such a voice interface most of the AUI structure can be transformed similar to the process for visual interfaces (cf. the arrows in Figure 5): The user interface is transformed into a VoiceXML Application, a presentation into a form and a data-oriented UI element into a VoiceXML field.

Although most of this transformation process is pretty much straightforward, we have to handle one special feature of voice interfaces: *grammars*. If we collect required input values by a visual interface, the entered values are, aside from typing errors, already *exact* enough to be used as parameters for further processing. For instance, if the user types "Berlin" for the place of departure and "Paris" as a destination, we can already start the database query, and may have to refine it since there are multiple stations in Berlin and Paris. If we do the same thing with a voice interface and the user says "hem, Berliin" and "I want to go to Paris." it is not enough to just record the audio. Instead the interface has to match the user's utterances to concrete concepts, such as items in a list of all train stations. Therefore, we have to provide a grammar which defines the expected syntax of the text to be spoken and describes how parts of that text can be matched to fill variables with values. In addition, the grammar may contain a lexicon that describes pronunciation information for tokens within the enclosing grammar.

If we want to generate a voice interface from an AUI model, we will have to find a way to assign grammars to the generated fields. One solution to this

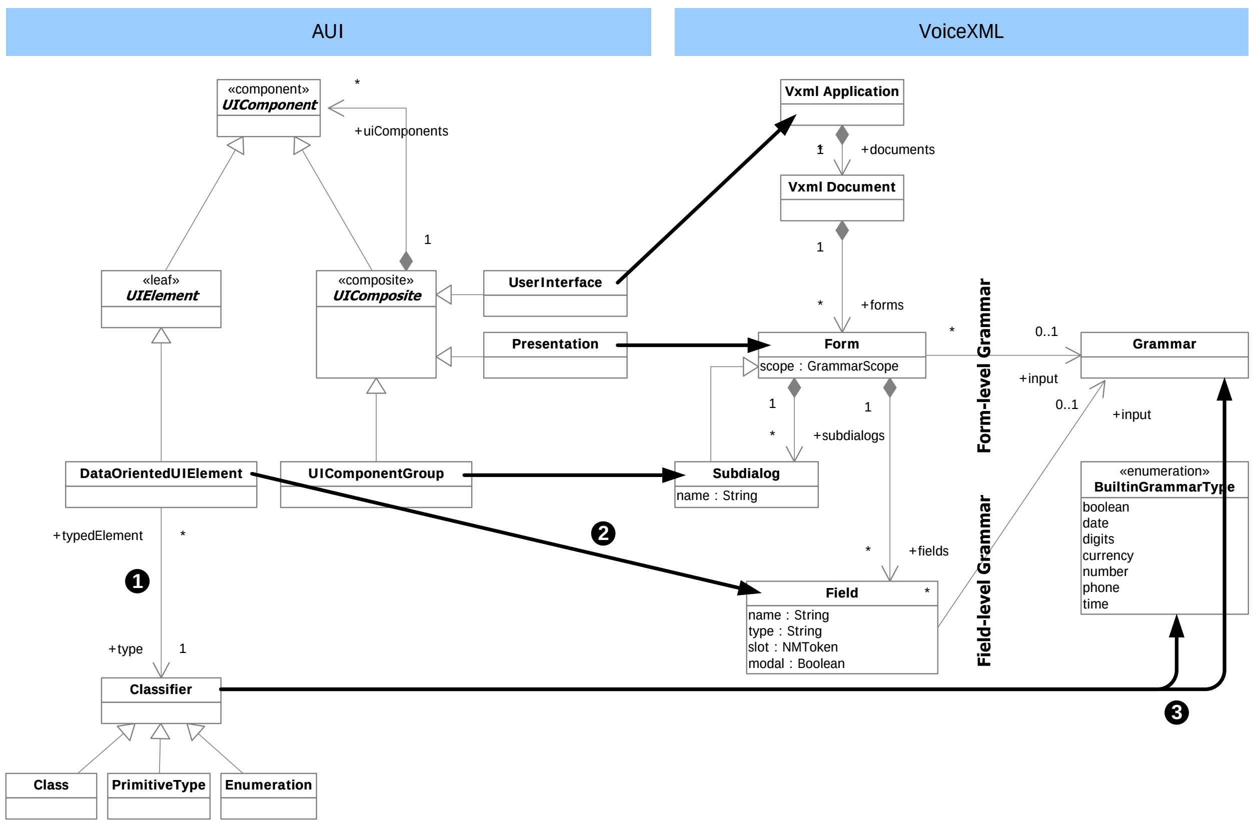


Figure 5. Transforming AUI elements to VoiceXML.

are types that are assigned to data-oriented user interface elements in the AUI model (cf. ❶ in Figure 5). If a user interface element is transformed into an VoiceXML field (❷) we can associate it with a field-level grammar, that is selected on the basis of the type of the user interface element.

For this, however, a classification consisting of just a few types like String, Integer and Boolean, which might be sufficient for a visual interface, is not detailed enough. Instead we have to use a more sophisticated classification with specific types, e.g., Time, or domain-specific concepts, such as TrainStation or MeansOfTransportation. Then we are able to assign grammars that are specific enough for speech input. Depending on the type we can use grammars that are predefined in VoiceXML, like time or currency, or reference domain-specific grammars (❸).

7.3 Mixed initiative voice interaction

In the last section we discussed a model of speech interaction which is rather rigid and is organized in a field-by-field structure. Hence, one might say, it is a *computer-directed* "conversation" [16].

If we strive for a more natural interaction where the user can enter information in a flexible way, we can use so called *form-level grammars*, which enable the user to (1) enter more than one item in one utterance and (2) do that in any order. In that case both the human and the computer direct the conversation. Hence, this model is sometimes called "*mixed initiative*". If we use form-level grammars in our timetable example, we have one grammar per presentation (cf. the ☁ areas in Figure 2). For instance, with a suitable grammar for the presentation "Place of departure and destination" the user could say "I want to go from Berlin to Paris" or "To Paris, from Berlin".

This model is more flexible and better resembles the natural interaction, which is expected by users unfamiliar with interactive voice response (IVR) systems. On the other hand, it takes more effort to create interfaces with form-level grammars, since these more complex grammars cannot be derived from the AUI model and have to be developed manually. Time and effort can be saved by using one form several times in the same application or by identifying building blocks which are common to multiple forms.

Hence, we have to trade off between the cost-effective generation of a voice interface with field-level grammars or a voice interface with form-level grammars which is more flexible and natural but requires the manual creation of grammars.

8. Conclusion

We have shown how our approach MANTRA can be used to generate several consistent user interfaces for a multi tier application. The approach presented in this paper offers four contributions:

First of all, we discussed how dialogue structures can be derived from the static hierarchical structure of interaction elements and constraints on the dialogue flow taken from a task model. This enables us to tailor the dialogue and logical presentation structures to take into account requirements imposed by front-end platforms or inexperienced users.

Secondly, the approach utilises novel model-driven technologies, i.e. model transformations described in ATL combined with code-generation technologies, and therefore shows the applicability of these mechanisms to the engineering of multiple user interfaces.

Thirdly, the approach generates prototypes of user-interfaces on three target platforms (GUI, dynamic website, mobile device) which can serve as front-ends to arbitrary web services. These front-ends are not just visual mock-ups, but working, functional prototypes which can by used to run through use cases and application scenarios. By this means we could then evaluate how the user reacts if the system *behaves* differently, for instance when the user interface has to handle large data sets or when the connection between front-end and application core gets slow due to network limitations.

Last, but not least, we discussed how our approach could be extended to support voice user interfaces (VUI) and why we have to trade off between VUIs which can be generated easily, and VUIs which offer a natural interaction but take more effort to develop, since they are structurally different from other interfaces.

9. Acknowledgements

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

References

[1] A. R. Puerta and J. Eisenstein, "Interactively Mapping Task Models to Interfaces in MOBI-D," DSV-IS 1998 (Design, Specification and Verification of Interactive Systems), Abingdon, UK, 1998

[2] T. Clerckx, K. Luyten, and K. Coninx, "The mapping problem back and forth: customizing dynamic models while preserving consistency," TAMODIA '04, Prague, Czech Republic, 2004

[3] J. Vanderdonckt, "Advice-Giving Systems for Selecting Interaction Objects," UIDIS'99, Edinburgh, Scotland, 1999

[4] F. Paternò and C. Santoro, "One Model, Many Interfaces," CADUI'02, Valenciennes, France, 2002

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

[6] M. Florins, F. M. Simarro, J. Vanderdonckt, and B. Michotte, "Splitting rules for graceful degradation of user interfaces," IUI'06, Sydney, Australia, 2006

[7] J. Eisenstein, J. Vanderdonckt, and A. R. Puerta, "Applying model-based techniques to the development of UIs for mobile computers," IUI '01, Santa Fe, NM, USA, 2001

[8] A. Seffah and H. Javahery, *Multiple user interfaces : cross-platform applications and context-aware interfaces*. New York, NY, USA: John Wiley & Sons, 2004

[9] G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon, and J. Vanderdonckt, "A unifying reference framework for multi-target user interfaces," *Interacting with Computers*, vol. 15, pp. 289-308, 2003

[10] F. Paternò, C. Mancini, and S. Meniconi, "ConcurTaskTrees: A diagrammatic notation for specifying task models," Interact'97, Sydney, Australia, 1997

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

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

[13] S. Holzner and J. Tilly, *Ant : the definitive guide*, 2nd ed. Sebastopol, CA, USA: O'Reilly, 2005

[14] W3C, "Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language," World Wide Web Consortium, 2006

[15] R. Beasley, *Voice application development with VoiceXML*. Indianapolis, IN, USA: Sams, 2002

[16] S. McGlashan, D. C. Burnett, J. Carter, P. Danielsen, J. Ferrans, A. Hunt, B. Lucas, B. Porter, K. Rehor, and S. Tryphonas, "Voice Extensible Markup Language (VoiceXML) Version 2.0," World Wide Web Consortium, 2004