A Domain-specific Language for the Model-driven Construction of Advanced Web-based Dialogs

Patrick Freudenstein[1], Martin Nussbaumer[1], Florian Allerding[2], Martin Gaedke[3]

[1]University of Karlsruhe (TH)
Institute of Telematics
D-76128 Karlsruhe, Germany
{freudenstein, nussbaumer}@tm.uka.de

[2]University of Karlsruhe (TH)
Institute of Applied Informatics and Formal Description Methods
D-76128 Karlsruhe, Germany
florian.allerding@aifb.uni-karlsruhe.de

[3]Chemnitz University of Technology
Distributed and Self-organizing Computer Systems Group
D-09107 Chemnitz, Germany
gaedke@informatik.tu-chemnitz.de

ABSTRACT
Complex dialogs with comprehensive underlying data models are gaining increasing importance in today's Web applications. This in turn accelerates the need for highly dynamic dialogs offering guidance to the users and thus reducing cognitive overload. Beyond that, requirements from the fields of aesthetics, Web accessibility, platform-independence, and Web service integration arise. To this end, we present an evolutionary, extensible approach for the model-driven construction of advanced dialogs. It is based on a Domain-specific Language (DSL) focusing on simplicity and fostering collaboration with stakeholders.

Categories and Subject Descriptors
D.2.2 [**Software Engineering**]: Design Tools and Techniques – *User Interfaces, Evolutionary Prototyping, Petri nets.* D.2.13 [**Software Engineering**]: Reusable Software – *Domain Engineering.*

General Terms
Design, Human Factors, Languages.

Keywords
Web Engineering, User Interaction, DSL, Model-driven.

1. INTRODUCTION
Considering the significant complexity of tasks performed within advanced enterprise Web applications as well as their comprehensive underlying data models, highly dynamic dialogs reducing cognitive overload and offering guidance to the users are required. Such usability aspects have a major influence on the efficiency and efficacy of users [2]. Beyond that, aspects from the fields of accessibility, platform independence, and adaptivity have to be considered. From a technical point of view, the integration of Web service communication for retrieving updates of the dialog's data model or for submitting it is a common requirement for this new generation of dialogs. Besides these application type-specific requirements, a systematic Web Engineering approach should also treat key factors like agility, strong stakeholder involvement and clear business objectives arising from a project management perspective as guiding principles [4].

Facing these requirements, we present an evolutionary, model-driven approach for the construction of rich dialogs. The presented Dialog DSL empowers stakeholders and domain experts having no experience in software development to directly contribute to the development effort by validating, adapting, and even developing dialog models. Thus, the collaboration with stakeholders throughout the development process can be intensified and the possibility of misunderstandings be lowered.

WWW 2008, April 21--25, 2008, Beijing, China.
ACM 978-1-60558-085-2/08/04.

2. THE DIALOG DSL
Based on our research towards DSL-based Web Engineering [3], the Dialog DSL consists of three elements:

Firstly, the *Domain-specific Model (DSM)* serving as formal schema for all dialogs that can be developed with the DSL. Having analyzed the dialog domain, we coined the DSM as an extensible combination of Petri net-based *Interaction Structures* for describing a dialog's dynamic behavior and XForms-based *Interaction Elements* for specifying user interaction primitives.

Secondly, the *Domain Interaction Model (DIM)* which is strictly based on the DSM and defines simple and intuitive graphical notations for the concepts defined therein. Focusing on simplicity and supplemented by a dedicated Web-based editor, it is used by stakeholders to validate, adapt or even create dialog models.

Thirdly, the *Solution Building Block (SBB)* which is a dedicated software component being capable of executing (XML-based) dialog models by adapting its behavior accordingly.

The Dialog DSL is used in a three-phase process model in the course of a continuous evolution. In the first phase *Data Design*, the data model for the aspired dialog in terms of an XML Schema document is either designed from scratch, extracted from a WSDL file or retrieved from the Dialog Reuse Repository. Based on this schema, the Dialog DSL's technical framework is already able to construct a running dialog. The second and third phases, *Partition Design* and *Appearance Design*, deal with the modeling of the dialog's dynamic behavior and concrete appearance. With respect to *evolution*, our approach allows for modifications at each stage of the process model while preserving model consistency. All changes can be applied at runtime and thus are visible instantly.

2.1 The Modeling Notation
The modeling notation consists of two tiers, thus fostering reuse and separation of concerns. On the first tier, the elements from the dialog's data model are distributed on various partitions and dynamic behavior between them using *Interaction Structures* is modeled. Dialog partitions are represented by Petri net places containing elements from the dialog's data model. At runtime, if a Petri net place is marked, its elements are visible. Petri net transitions correspond to the performed user interaction, i.e. changing a value in the dialog's data model. Focusing simplicity,

we defined graphical Petri net transition templates for well-known Interaction Structures like *Choice* and *Sequence* (Figure 1).

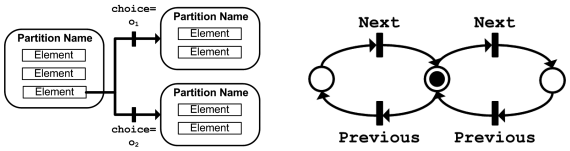

Figure 1. Interaction Structures 'Choice' and 'Sequence'

The Choice transition template is connected to the element whose value decides on which transition is fired and to the various target places. The transitions are labeled with the various values the element in the source place can take. At runtime, if a place becomes marked, all elements become marked. When the user changes the value of an element connected with a Choice transition, the mark of the element flows to the target partition, thus making it and its elements visible. The source partition's mark, however, is still there, meaning that both partitions are visible. If the source partition should become invisible, the transition can be annotated with a *[Replace]* tag.

On the second tier, the concrete appearance of each partition employing *Interaction Elements* is specified (Figure 2). An XForms user control represented by a corresponding graphical symbol is assigned to each data element. With respect to the device-dependent model adaptations at runtime, dedicated graphical symbols allow for marking partitions and groups of interaction elements as non-dividable. This 'pen and paper' modeling approach can be augmented by a property editor allowing for a detailed configuration of interaction elements.

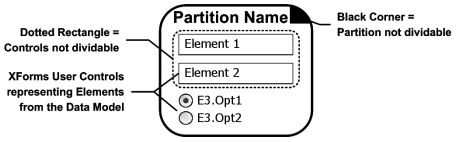

Figure 2. User controls binding and semantic grouping.

2.2 The Editor

Figure 3 shows the editor's user interface for *Partition & Transition Design* (1) and *Appearance Design* (2). Regarding the former, the editor displays a list of elements from the data model that have not yet been assigned to a partition (left panel). The partition assignment can be done via drag & drop. The top panel provides buttons for adding new partitions and defining Sequence or Choice transitions. Each partition contains an *Appearance Design* button leading to the Appearance Design view of the respective partition. There, an interaction element type for each data element can be selected; a default interaction element has already been assigned based on the data element's type. Furthermore, markup, e.g. for headings, can be inserted and the relative layout of the interaction elements can be defined. Beyond that, a partition can be tagged as non-dividable and semantically cohesive element groups be defined. A *Property Editor* supports the detailed configuration of each interaction element, e.g. its label, access key, hint text or appearance. Additional properties allow for defining input validations or calculations.

2.3 Model Transformations

Our approach uses two kinds of model transformations: *user-agent related transformations* and *model-to-code* transformations. The former are performed directly on the Petri net model, e.g. for decomposing partitions into smaller device-specific partitions. Model-to-code transformations transform the dialog models into

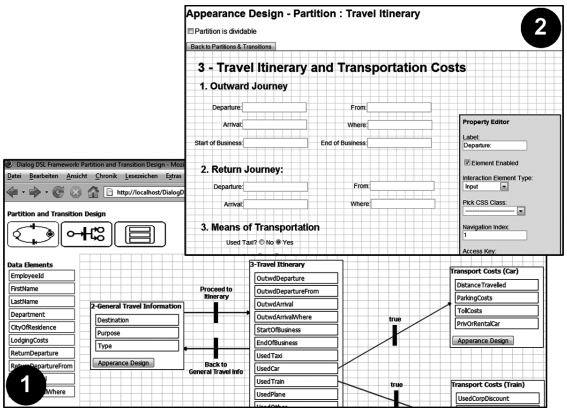

Figure 3. The Web-based editor.

executable markup and vice versa. So far, we have developed bidirectional transformations to XForms based on term rewriting. Other markup languages could be easily incorporated.

2.4 Technical Platform

The *WebComposition Service Linking System (WSLS)* [1] serves as technical platform for the *Dialog Solution Building Block (SBB)*. WSLS aims at facilitating the systematic construction and evolution of Web applications by reusing software artifacts and emphasizing the "configuration instead of programming" paradigm. The Dialog SBB communicates with a Dialog Web Service for reusing dialogs and initiating the generation of raw dialog models based on a given data schema. Moreover, it links to the Web-based editor for creating and adapting dialogs. Finally, the SBB identifies requesting user agents, performs corresponding dialog adaptations and generates executable markup.

3. EVALUATION

The Dialog DSL was successfully used for the construction of several complex dialogs in a large-scale Enterprise Application Integration project. The observed improvements regarding the efficiency and efficacy of the construction process are promising. Due to the model-driven approach, the construction time could be considerably decreased. Moreover, the simple template-based modeling notation and the associated editor as well as short iteration cycles combined with immediate previews allowed for an intensified stakeholder collaboration. Compared to similar dialogs developed without the Dialog DSL, we observed an increase in the dialog's usability caused by the adoption of the introduced Interaction Structure patterns and their intuitive application. Currently, we are working on a comprehensive empirical study on the assets and drawbacks of the Dialog DSL based on diverse scenarios and stakeholder groups.

4. REFERENCES

[1] Gaedke, M., Nussbaumer, M., and Meinecke, J., WSLS: An Agile System Facilitating the Production of Service-Oriented Web Applications, in Engineering Advanced Web Applications, S.C. M. Matera, Editor. 2005, Rinton Press.

[2] Nielsen, J., Forms vs. Applications, in Jakob Nielsen's Alertbox. 2005

[3] Nussbaumer, M., Freudenstein, P., and Gaedke, M., The Impact of DSLs for Assembling Web Applications. Engineering Letters, 2006. 13(2006): p. 387-396.

[4] The Standish Group International, CHAOS Research - Research Reports (1994-2005).