Constructing Advanced Web-based Dialog Components
with Stakeholders – a DSL Approach

Patrick Freudenstein and Martin Nussbaumer
Karlsruhe Institute of Technology - University of Karlsruhe (TH)
Department of Telematics, IT Management and Web Engineering Research Group,
Engesserstr. 4, 76128 Karlsruhe, Germany
{patrick.freudenstein, martin.nussbaumer}@kit.edu

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 reducing cognitive overload. Beyond that, requirements from the fields of Web accessibility, platform-independence and Web service integration arise. Considering the resulting complexity, a systematic engineering approach becomes important. Besides addressing the specific characteristics of these dialogs, key success factors from a communication perspective like strong user involvement and clear business objectives must be taken into account. To this end, we present an evolutionary, extensible approach for the model-driven construction of advanced dialogs which is based on a Domain-specific Language (DSL). We introduce a modeling notation based on Petri net constructs and XForms as well as a supporting Web-based editor, both focusing on simplicity and fostering communications. The technical framework allows for quick prototyping and flexible changes. In conclusion, complex, device-independent dialogs with rich behavior and appearance can be constructed and evolved with intense stakeholder collaboration.

1. Introduction

The World Wide Web is currently performing its next evolution cycle towards a platform for sophisticated enterprise applications and portals with a high intensity and complexity of user interaction aspects [14, 17]. Considering the significant complexity of tasks performed within these new types of applications as well as the 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 [12]. Beyond that, aspects from the fields of accessibility, platform independence, adaptivity, and Web service communication have to be considered. Besides these application type-specific requirements, key factors arising from a project management perspective have to be taken into account. To this end, agility, strong stakeholder involvement and clear business objectives have been identified as key success factors [21].

This being the situation, we present the Dialog DSL – an evolutionary, model-driven approach for the construction of rich dialogs which meets these requirements. The Dialog DSL is part of our research towards the model-driven construction of workflow-based Web applications using Domain-specific Languages for their various aspects [6]. It is based on our previous work, the WebComposition Service Linking System (WSLS) approach [7] and our latest research towards DSL-based Web Engineering [13]. By using this DSL-based approach, stakeholders and domain experts having no experience in software development can directly contribute to the development effort by understanding, validating, adapting, and even developing dialog models. Moreover, the Dialog DSL allows for rapid iteration cycles with running versions of the aspired dialog being available very early. In conclusion, our DSL approach enables an intense collaboration throughout the development process and lowers the possibility of misunderstandings.

Section 2 introduces a core set of requirements for a systematic dialog engineering approach. In section 3, an overview of the Dialog DSL is given, whereas a detailed presentation of its core pillars follows in section 4: The modeling notation based on Petri nets [16] and XForms [2], a supporting Web-based model

IEEE
computer
society

editor, and the involved model transformations. Experiences from the application of the Dialog DSL in real-world projects are outlined in section 5. Section 6 discusses related work and section 7 concludes the paper and presents future work.

2. Problem Scope

Based on challenges experienced in several real-world projects as well as from general requirements for dialog engineering methods found in literature, e.g. [9, 10], we identified the following key requirements. The first three requirements aim at vital characteristics of advanced dialogs a suitable engineering approach must address, whereas the last three concern the development process and the methodology itself. These requirements served as starting point for the design of our approach and will be used for the evaluation of related work and the method itself (section 6).

Usability: The engineering approach should treat dynamic behavior, user guidance and feedback, and adaptivity as vital usability features of advanced dialogs.

Accessibility: The accessibility of the resulting dialogs for everyone is a key requirement [3]. Especially in business applications or in the public sector, no potential users must be passed over.

Platform-independence: Particularly dialogs in business-related Web applications should be accessible not only from a desktop or notebook computer, but also from mobile devices like PDAs.

Agility & Evolution: A dedicated dialog engineering approach should be agile and evolution-oriented in terms of supporting short revision lifecycles and the efficient adoption of changes [4, 19].

Strong Stakeholder Involvement: Strongly emphasizing stakeholder involvement and supporting efficient and efficacious communications by focusing on simplicity and supported by rapid prototyping [22] is, particularly for the construction of dialogs, crucial.

Reuse: With respect to requirements from the fields of agility, software quality, and development efficiency, systematic reuse of all kinds of artifacts [11] should be incorporated as a guiding principle throughout the development process.

3. The Dialog DSL at a Glance

The Dialog DSL is part of our research in the context of DSL-based Web Engineering [13] in general and workflow-based Web Applications [6] in particular. The overall goal of DSL-based Web Engineering is to foster communication and collaboration with stakeholders by emphasizing simplicity. Based on the DSL-based Web Engineering approach, Web applications are constructed by assembling components for various concerns (e.g. dialogs, workflows or data presentation) and configuring them with DSL programs at runtime. These DSL programs in turn are obtained through transformations of visual models tailored both to individual stakeholder groups and the problem domain.

3.1 Elements of the Dialog DSL

Domain-specific Model (DSM): The Dialog DSL's DSM specifies the formal schema for all dialogs that can be designed with the DSL. A DSM should be tailored to the problem domain, not the solution domain, i.e. the DSM must abstract from the final implementation. Exploring the domain of Web-based dialogs, we identified two necessary groups of concepts to be integrated in an appropriate DSM: Concepts for describing *interaction elements* and concepts for specifying dynamic behavior of a dialog, so-called *interaction structures*.
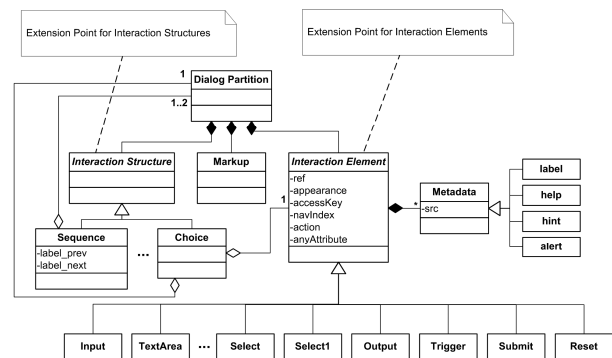

Figure 1. Simplified excerpt from the DSL's DSM.

Figure 1 shows an excerpt from the DSM starting from a *dialog partition*, i.e. a semantically cohesive part of a dialog, which can contain *interaction structures* and *interaction elements*. Regarding the latter, we chose to integrate the concepts for specifying interaction elements from the W3C XForms standard [2]. They are a good means for expressing interaction elements within a DSL as they are based on high-level user interaction primitives [18]. Thus, they separate the expression of the intent underlying a particular form control from its presentational and behavioral aspects. The DSM can be extended by additional interaction elements as indicated in the figure by the corresponding extension point. Regarding *interaction structures*, we defined an extensible core set representing common dynamic behaviors in dialogs: *Sequence* represents a wizard-like sequence of dialog

partitions, each of them being presented to the user one at a time and connected via previous / next navigation facilities, thus allowing for semantic grouping and reducing complexity. *Choice* represents the dynamic display of a dialog partition in response to a selection made by the user. As indicated in the figure, this initial set of interaction structures can also be extended.

Domain Interaction Model (DIM): We propose a two-tiered modeling notation based on Petri nets and XForms. On the first tier, the elements from the 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. Petri net transitions correspond to the performed user interaction, i.e. changing a value in the dialog's data model. Interaction structures are represented by predefined graphical Petri net templates. On the second tier, the concrete appearance of each partition employing interaction elements is specified. This two-tiered modeling approach fosters reuse and allows for separation of concerns - thus again putting emphasis on simplicity.

Solution Building Block (SBB): A SBB is a dedicated software component being capable of executing DSL programs by adapting its behavior accordingly. The Dialog SBB runs on the WSLS framework [7] and represents the core of the technical platform. It communicates with a Dialog Web Service for initiating the generation of raw dialog models based on a XML Schema definition or for reusing dialog models. In each case, a running dialog is obtained without any manual modeling. Moreover, the SBB links to a Web-based editor for creating and adapting dialogs at runtime, i.e. no (re)compilation or (re)deployment is required. Finally, the SBB identifies requesting user agents at runtime based on the HTTP user agent string or by evaluating *User Agent Profile (UAProf)* information [15], performs corresponding dialog adaptations as well as ultimately transforms dialog models into executable markup, e.g. XForms. Submissions of the dialog model in whole or part are received by the SBB and processed, e.g. in the context of a workflow, or forwarded to a Web service the dialog communicates with. In the latter case, the SBB receives the response from the Web service and forwards it to the corresponding client.

3.2 The Dialog DSL Process Model

The Dialog DSL's associated process model for the construction of advanced dialogs consists of three phases in the course of a continuous evolution.

Data Design: In this phase, the data model (i.e. an XML schema) for the aspired dialog is either retrieved from the reuse repository, extracted from the WSDL specification of a Web service the dialog shall communicate with or developed from scratch.

Partition Design: This phase addresses the modeling of dialog partitions and dynamic behavior and is ideally supported by a visual drag & drop editor. Therefore, in the first step, the elements from the dialog's data schema are distributed on several dialog partitions, each of them representing a semantically cohesive dialog unit. Then, employing predefined interaction structures like *Sequence* or *Choice*, dynamic transitions between them are defined.

Appearance Design: In this phase, the concrete appearance of each dialog partition is designed, again supported by the Web-based editor. Therefore, an interaction element is assigned to each element from the data model. Based on the type of a data element, a possible interaction element was already assigned at dialog generation time (e.g. input for string, select1 for enumerations etc.) and can be modified. Furthermore, style sheets can be applied and additional markup be inserted. In order to provide additional guidance to the user, input validations or dynamic features like hints or auto completion can be defined. Special notations allow influencing the device-adaptive rendering of a dialog at runtime. Here again, the visual editor supports strong collaboration with stakeholders.

Evolution: In case of extensions or modifications in the data model, the technical framework regenerates only those elements affected by the change while preserving the rest. These new or modified elements can then be designed in detail with respect to partition membership, dynamic behavior and appearance in the succeeding phases. For changes not affecting the data model, the Data Design phase can be skipped.

4. Dialog DSL Building Blocks

This section describes the core pillars of the Dialog DSL approach in detail: The modeling notation for specifying dynamic behavior and concrete appearance of a dialog (4.1), a corresponding Web-based editor (4.2) and the employed model transformations (4.3).

4.1 The Modeling Notation

The DSM of the Dialog DSL defines two major groups of concepts: *Interaction elements* and *interaction structures*. Interaction elements represent high-level user interaction primitives following the W3C XForms user controls, whereas interaction

structures stand for common dynamic behaviors in dialogs like *Sequence* or *Choice*. The Dialog DSL's modeling notation defines corresponding notations for the concepts defined in the DSM.

With regard to *interaction elements*, employing well-known dialog user controls turned out to be a good choice. For example, an *input* interaction element is represented by an input field, a *select1* interaction element by a dropdown list control, and a *trigger* interaction element by a button. This way we defined a graphical symbol for each interaction element in the DSM. The fact that almost all symbols in the DIM notation were already known to stakeholders made it rather intuitive. Regarding the modeling of dynamic behavior by *interaction structures*, we decided to employ predefined Petri net constructs. Petri nets are very suitable for modeling dynamic behavior, parallelism and the state of a system. These characteristics can all be found in advanced dialogs, thus making Petri nets a good choice. In order to reduce complexity, we predefined a transition template for each interaction structure, thereby simplifying the modeling process. In order to achieve *separation of concerns*, the modeling notation is divided into two tiers: The first tier addresses the modeling of dialog *partitions and transitions*, whereas the second tier focuses on the *appearance design* of each partition.

4.1.1 Partitions & Transitions Modeling Tier

On this tier, semantically cohesive elements from the dialog's data model are grouped into dialog partitions which are represented by Petri net places. At runtime, if a place is marked, its elements are visible. Afterwards, transitions between these dialog partitions are defined using predefined transition templates according to the DSL's interaction structures.
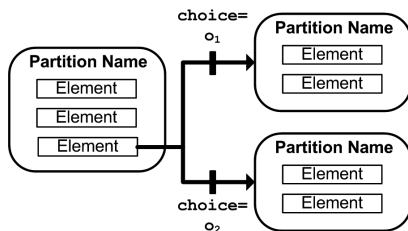

Figure 2. A 'Choice' interaction structure as Petri net transition template.

Figure 2 shows the Petri net representation of a *Choice* interaction structure. We consider elements in a Petri net place again as Petri net places, thus resulting in hierarchical Petri nets. Accordingly, 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. To this end, it is advisable to map such an element to an interaction element with a discrete value range (e.g. *select1*), which can be done on the Appearance Modeling Tier. At runtime, if a place becomes marked, all elements become marked. When the user changes the value of an element connected to 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 this is not the desired behavior, i.e. the source partition should become invisible and only the target partition become visible, the transition would have to be connected to the source partition instead of the concrete element. As we are trying to emphasize simplicity in the modeling notation, we decided to connect a Choice transition always to the respective element. In case the source partition shall become invisible when a transition fires, the transition can be annotated with a [Replace] tag. It should be mentioned that, when a partition becomes invisible, its state is preserved by the marking of its encapsulated elements and thus is restored when it becomes visible again.

The Petri net representation of a *Sequence* Interaction Structure is shown in Figure 3. It represents a wizard-like navigation through a linear space of dialog partitions.
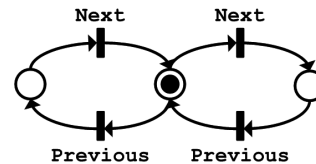
Figure 3. A 'Sequence' interaction structure.

4.1.2 Appearance Modeling Tier

Based on the dialog partitions defined on the superordinate tier, this tier focuses on their concrete appearance design. Figure 4 illustrates a core set of the possible modeling options. First, an XForms user control represented by a corresponding graphical symbol has to be assigned to each data element. Moreover, labels can be defined and markup, e.g. for headings, be inserted. Furthermore, a partition can be semantically tagged as 'not dividable', indicated by a black corner. This means that possible runtime model adaptations for clients with small displays should attempt to keep the partition's elements together. In case a partition is possibly dividable, this can also be done on a more fine-grained level for interaction elements, indicated by a dotted rectangle. Supported by a corresponding editor, this 'pen and paper' modeling approach can be augmented by configuring interaction elements in detail using a property editor.
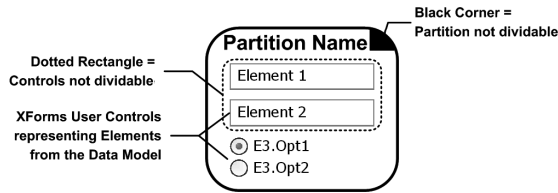
Figure 4. Binding XForms User Controls to data elements and defining semantic groups.

4.2 The Editor

In order to support the model-driven construction and evolution of dialogs using the modeling notation described above, we developed a corresponding Web-based editor. Figure 5 shows screenshots of the editor's user interfaces for *Partition & Transition Design* (1) and *Appearance Design* (2). Regarding the former, the editor displays a list of the elements from the data model that have not yet been assigned to a partition (left panel). In the top panel, graphical buttons for adding new partitions and defining Sequence or Choice transitions are available. Data model elements from the left panel can be assigned to partitions via drag & drop. After having clicked on a Sequence or Choice transition button, the user can connect two partitions or an element from one partition with another partition respectively via clicking on them. Thereupon, the editor draws the transition and allows the user to annotate it.
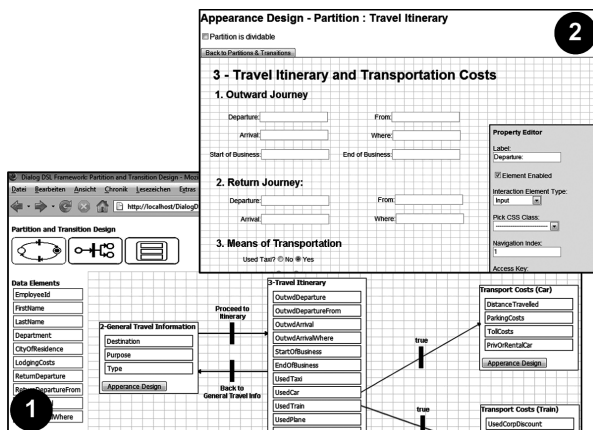
Figure 5. Partition & Transition Design (1) and Appearance Design (2) in the Web-based editor.

Each partition contains a button labeled 'Appearance Design' leading to its Appearance Design view (Figure 5-2). There, an interaction element 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 be defined. Beyond that, a partition can be tagged as non-dividable and semantically cohesive element groups can be specified. A Property Editor allows for the detailed configuration of each interaction element like e.g. its label, navigation index, access key or appearance, hint, help and alert texts, input validations or calculations. A screenshot of the rendered dialog resulting from the models edited in Figure 5 is shown in Figure 7.

4.3 Model Transformations

Within the presented DSL approach, two kinds of model transformations are required: user-agent-related transformations and model-to-code transformations.

4.3.1 User-Agent-related Transformations

In our approach, dialogs and their decomposition into partitions should be modeled with respect to a regular desktop terminal. For user agents with smaller screens, they have to be further decomposed into suitable device-specific partitions.
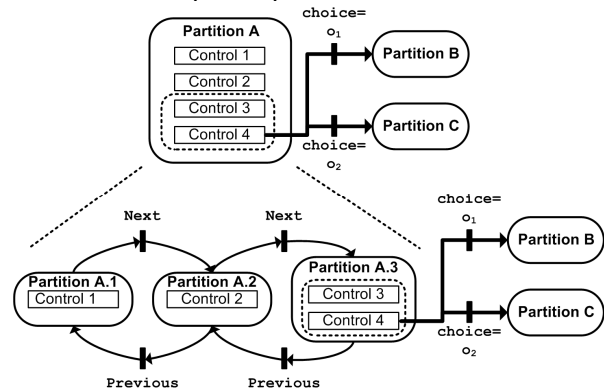
Figure 6. Pagination of a large dialog partition.

Figure 6 illustrates the model transformation for decomposing Partition A into several smaller partitions, i.e. Partion A.1-A.3. The pagination algorithm fills a partition with controls until their combined estimated size on the user agent exceeds the given maximum screen size. In that case, an additional partition is created and filled. As far as possible, semantic groupings like the grouping of Control 3 and 4 are preserved. The resulting micro-partitions are connected via the Sequence interaction structure.

4.3.2 Model-Code Transformations

On the one hand, after potential model adaptations have been conducted by the SBB, it has to translate the user agent-specific dialog model into executable markup. On the other hand, in order to enable the import of existing markup code from third parties and

its subsequent adaptation using the Web-based editor, also transformations in the backward direction have to be provided. So far, we developed such bidirectional transformations between the Dialog DSL's formal schema, i.e. the DSM, and XForms.

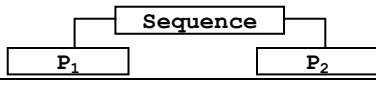**Table 1.** Multi-step transformation of dialog models into final markup.

(1)	DSM-based pattern	Sequence / P₁ / P₂
(2)	Context-free grammar rule	`Sequence:=P₁ P₂`
(3)	Extended rule	`Sequence:=seq(P₁, P₂)`

(4)	Term rewriting rule	`seq(t1,t2)→`
		`  <switch>`
		`    <case id="t1">eval(t1)</case>`
		`    <case id="t2">eval(t2)</case>`
		`  </switch>`

Table 1 illustrates the multi-step transformation process. In the first step, a DSM-based model element (1) is mapped to a context-free grammar-based expression (2). Then, this expression is extended by a term-algebraic operation (3) allowing for their processing within a term rewriting-based compiler. In the last step, term rewriting rules are applied to translate the expressions into the final markup code (4). Here, term rewriting rules to other markup languages (e.g. XAML) could be flexibly incorporated.
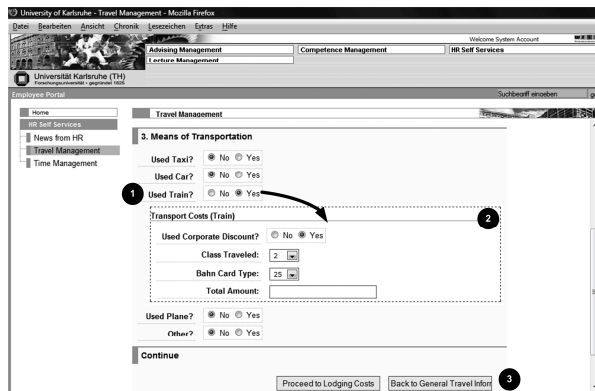
Figure 7. Rendered dialog with dynamic Choice (1+2) and Sequence (3) behavior.

5. Practical Experiences

The Dialog DSL was successfully used for several complex dialogs within the KIM project [1]. 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 intensified stakeholder collaboration. For example, adapting the model and immediately seeing the impact on the running dialog eased the collaboration a lot. Beyond that, the modeling notation in combination with the editor turned out to be rather intuitive, even for stakeholders with few technical skills. 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.

6. Related Work

In the following, we outline two representative approaches and point out the differences based on the requirements presented in section 2.1.

The Object-Oriented Hypermedia (OO-H) method [8] supports the model-driven construction of dialogs and their direct transformation into executable source code. Thus, it fosters agility, even though evolution cycles with OO-H seem to be longer than with our approach. Regarding rich dynamic behavior and user guidance, OO-H defines a valuable interaction pattern catalogue including static and dynamic navigation patterns as well as command control patterns. Although the patterns were defined from a user's perspective, they lack an intuitive graphical representation. Thus, the modeling process for the experienced designer is eased and the quality of the resulting interfaces improved. However, regarding the integration of stakeholders in the development process, detailed OO-H dialog models still remain quite complex. The OO-H model compiler is able to produce markup for various platforms like ASP, JSP, PHP or WML. Dialog-specific markup languages like XForms are not included so far.

The Object-Oriented Hypermedia Design Method (OOHDM) [20] employs Abstract Data View (ADV) models for the specification of dialogs and their dynamic behavior [5]. While ADV seem to be suitable for the formal specification of a dialog's static and dynamic aspects, they are rather unintuitive for stakeholders with few technical skills. Client-specific dialog adaptations as well as accessibility concerns have not been addressed yet. Recently, the OOHDM group proposed an interesting approach towards enriching hypermedia application interfaces by animating navigational transitions and thereby emphasizing semantically important information [10]. With regard to the dynamic transitions employed in

our approach, it would be interesting to further investigate how both approaches can be integrated, thus offering additional guidance to the user.

7. Conclusion & Future Work

Facing the challenges found in the development and evolution of advanced Web-based dialogs, we presented a DSL-based engineering approach - the Dialog DSL. It puts strong emphasis on simplicity, thereby enabling stakeholders to intensely participate in the development process. The DSL is formally based on *interaction primitives* derived from the W3C XForms standard and an extensible set of common *dynamic interaction structures* like 'Sequence' or 'Choice'. The proposed modeling notation employs Petri net semantics for the decomposition of dialog elements into dialog partitions and the modeling of dynamic transitions between them. Dedicated notations allow influencing the *device-adaptive rendering* at runtime. A *Web-based editor* supports the easy yet detailed creation and adaption of dialog models. Modifications to the dialog model can be performed at runtime, thus enabling *rapid evolution cycles*. The DSL's *technical framework* realizes the generation of raw dialogs from a data schema and facilitates *reuse* of dialog models. Moreover, it *adapts the dialog model* according to the requesting client's capabilities and transforms it into executable markup (e.g. XForms).

We successfully applied the presented approach in several real-world scenarios and observed promising improvements. A comprehensive empirical evaluation of the Dialog DSL will be the next step in our research agenda. Beyond that, we are striving for identifying and conceptualizing additional interaction structures from existing dialogs. Moreover, we are planning to integrate a proactive rule-based usability validation supporting the modeler already at design time.

8. References

1. KIM Project Homepage - 2005), University of Karlsruhe: http://www.kim.uni-karlsruhe.de/
2. Boyer, J.M., et al.: XForms 1.0 (Third Edition) - W3C Recommendation (2007)
3. Consortium, W.W.W.: Web Accessibility Initiative (WAI) Homepage - 2006): http://www.w3.org/WAI/
4. Deshpande, Y., et al.: Web Engineering. Journal of Web Engineering, 2002. 1(1): p. 3-17
5. Fialho, A. and Schwabe, D.: Enriching Hypermedia Application Interfaces. in Proceedings of 6th International Workshop on Web-Oriented Software Technologies (IWWOST'07). 2007. Como, Italy
6. Freudenstein, P., et al.: Model-driven Construction of Workflow-based Web Applications with Domain-specific Languages. In Proceedings of the 3rd International Workshop on Model-Driven Web Engineering. 2007: CEUR Workshop Proceedings, ISSN 1613-0073.
7. 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. p. 26-37
8. Gómez, J. and Cachero, C.: OO-H Method: Extending UML to Model Web Interfaces, in Information modeling for internet applications, P.v. Bommel, Editor. 2003, IGI Publishing: Hershey, PA, USA. p. 144 - 173
9. Kappel, G., et al.: Web Engineering: The Discipline of Systematic Development. 1 ed. 2006: Wiley
10. Matera, M., Rizzo, F., and Carughi, G.T.: Web Usability: Principles and Evaluation Methods, in Web Engineering, E. Mendes and N. Mosley, Editors. 2006, Springer: Heidelberg. p. 143-180
11. Mcllroy, M.D.: Mass Produced Software Components. in Sofiware Engineering; Report on a conference by the NATO Science Committee. 1968. Garmisch, Germany: NATO Scientific Affairs Division, Brussels, Belgium
12. Nielsen, J.: Forms vs. Applications, in Jakob Nielsen's Alertbox. 2005
13. Nussbaumer, M., Freudenstein, P., and Gaedke, M.: The Impact of DSLs for Assembling Web Applications. Engineering Letters, 2006. 13(2006): p. 387-396
14. O'reilly, T.: What Is Web 2.0 - Design Patterns and Business Models for the Next Generation of Software - Online Article (2005): http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html (18.10.2005)
15. Open Mobile Alliance: User Agent Profile Specification - 2003): http://www.openmobilealliance.org/release%5F program/uap_v2_0.html
16. Petri, C.A.: Kommunikation mit Automaten. 1962, Technischen Universität Darmstadt: Darmstadt
17. Phifer, G., et al.: Hype Cycle for Web and User Interaction Technologies, 2007, in Gartner Reports. 2007, Gartner, Inc.: Stanford, CT, USA
18. Raman, T.V.: Auditory User Interfaces--Toward The Speaking Computer. 1997: Kluwer Academic Publishers
19. Roger S. Pressman: Part Three: Applying Web Engineering, in Software Engineering: A Practioner's Approach. 2005, McGraw-Hill: New York. p. 499-626
20. Schwabe, D. and Rossi, G.: An Object Oriented Approach to Web-Based Application Design, in Theory and Practice of Object Systems 1998, Wiley and Sons: New York, USA
21. The Standish Group International: CHAOS Research - Research Reports (1994-2005): http://www.standishgroup.com
22. Wiegers, K.E.: Software Requirements. Second ed. 2003: Microsoft Press. 430 pages