

**The Dialog DSL: Rapid Development of
Advanced Web-based Dialogs
with Stakeholders**

**Patrick Freudenstein
and
Martin Nussbaumer**

Interner Bericht 2008-6



Universität Karlsruhe (TH)
Forschungsuniversität • gegründet 1825

ISSN 1432-7864



Fakultät für **Informatik**

The Dialog DSL: Rapid Development of Advanced Web-based Dialogs with Stakeholders

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 [22, 25]. 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 [20]. Beyond that, aspects from the fields of accessibility, platform independence, and adaptivity have to be considered. From the technical point of view, the integration of Web service communication for retrieving updates of the dialog's data model or for submitting the final user input is a common requirement found in this new generation of Web applications. Facing the immense complexity in development and evolution of such advanced dialogs, a dedicated engineering approach is desirable.

Besides these application type-specific requirements, such a dialog engineering approach must also consider key factors arising from a project management perspective. To this end, agility, strong stakeholder involvement and clear business objectives have been identified as key success factors in comprehensive empirical studies [28] and taken on in agile software development methods [3]. Against this background, a systematic Web Engineering approach should also treat them as guiding principles.

This being the situation, we present an evolutionary, model-driven approach for the construction of rich dialogs meeting both the dialog- and the project management-specific requirements stated above. Our approach comprises a Domain-specific Language (DSL), the Dialog DSL, and an associated technical framework. 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 [11]. It is based on our previous work, the WebComposition Service Linking System (WSLS) approach [12] and our latest research towards DSL-based Web Engineering [21]. 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 short 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 the particular requirements for a systematic dialog engineering approach and illustrates them in the context of a real-world EAI scenario. In section 3, we outline the idea of DSL-based Web Engineering to facilitate the understanding of the succeeding sections. Afterwards, in section 4, an overview of the Dialog DSL, its extension points and the associated process model is given. Subsequently, in section 5, a detailed presentation of its core pillars based on the example scenario follows: The modeling notation based on Petri nets [24] and XForms [4], a supporting Web-based model editor, the underlying technical platform, and the involved model transformations. The latter are used for runtime adaptations of dialog models according to characteristics of requesting client devices as well as their transformation into executable markup, e.g. XForms code. Experiences from the application of the Dialog DSL in real-world projects are outlined in section 6. Section 7 discusses related work and section 8 concludes the paper and outlines future work.

2. Problem Scope

In this section, we first introduce a general core set of requirements a systematic engineering approach for the construction of advanced dialogs should fulfill. Afterwards, these requirements are illustrated in the context of a representative scenario, which will also serve as a running example throughout the paper.

2.1 Requirements Catalogue

Based on the challenges experienced in several real-world projects as well as from general requirements for dialog engineering methods found in literature, e.g. [15, 18], 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 7).

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. 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.

Strong Stakeholder Involvement: Strongly emphasizing stakeholder involvement and supporting efficient and efficacious communications is, particularly for the specification and 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 should be incorporated as a guiding principle throughout the development process.

2.2 Challenges in a Real-World Scenario

The project “Karlsruhe’s Integrated Information Management (KIM)” [1] is a university-wide Enterprise Application Integration (EAI) project in which we have been collaborating in for several years now. The KIM project aims at integrating the great diversity of heterogeneous systems and at establishing an IT platform for the integrated and uniform execution of university-spanning business processes.

Within the KIM project, we addressed the travel management business process starting from the traveler filling out her travel request up to receiving the final travel expense statement from the travel management department. In the course of the development of an corresponding workflow-based Web application using our methodology presented in [11], several advanced dialogs had to be constructed. One of them, the travel expense report filled out by travelers after the journey, will serve as a representative example.

The travel expense report includes personal data, the detailed travel itinerary as well as all incurred expenses. Thus, the associated data model is quite comprehensive and includes several context-dependent elements. For example, the required information

differs depending on the travel destination (national / international) and the used means of transport (e.g. train, plane, car). Thus, a dialog should reduce complexity by dynamically showing only relevant elements to the user. Beyond that, dynamic features like hints, data validation, automatic calculation etc. can further improve the *usability* of the dialog.

Regarding the dialog's future users and their skill-levels, two major groups can be identified: On the one hand, experienced faculty secretaries handling travel reports for several senior researches and thus using the dialog at a regular basis. On the other hand, regular employees using the dialog only a few times a year. The user group of frequent users is likely to prefer filling out the dialog in an "expert mode", having all input fields on one screen, thereby supporting fast processing. In contrast, infrequent users need much more guidance and support reducing cognitive overload, e.g. like the 'Wizard' interaction pattern [29].

In addition to these usability requirements, the dialog should also be usable *platform-independently*. For example, travelers might want to fill out parts of the expense report during their journey using a mobile device (e.g. a PDA). Furthermore, due to recent legal regulations in the public sector, dialog must be *accessible* for users with disabilities [6].

Besides these dialog-specific requirements, further challenges arising from a development- and project management perspective exist. Web applications in general [8] and their dialogs in particular [27] underlie a continuous *evolution*, for example due to adaptations in the data model, design or layout changes, or completely new requirements. Thus, a suitable dialog engineering approach should be *agile* in terms of support short revision cycles and the efficient adoption of changes.

Throughout the development process of the travel expense dialog, a multitude of *stakeholders* from different organizational units with diverse professional backgrounds and skill levels has to be involved. In order to improve the efficiency and efficacy of the collaboration, the employed modeling notations have to be as simple and intuitive as possible and should focus on relevant dialog-specific aspects while hiding unwanted complexity. An associated, easy-to-use editor for creating and adapting dialog models is desirable. Early prototypes can help to achieve a common understanding and to identify discrepancies between requirements and their realization [30]. Design alternatives, e.g. targeting usability aspects, can be explored and misunderstandings can be resolved at an early, yet cost-efficient point of time.

Considering the immense number of Web-based dialogs being developed at a University over time, the strong integration of *reuse* in all phases of the development process is desirable. The systematic reuse of various artifacts, e.g. data models, dialog models (in part or whole), as well as software components contributes particularly to development efficiency and software quality [19].

3. DSL-based Web Engineering

The Dialog DSL is part of our research in the context of DSL-based Web Engineering [21] in general and Workflow-based Web Applications in particular [11]. The overall goal of our DSL-based Web Engineering approach is fostering communication and collaboration with stakeholders by emphasizing simplicity. A DSL can be seen as "a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain" [9]. With respect to the development of advanced Web applications, we propose using various DSLs, each of them being tailored to a specific problem domain. Each DSL employs well-known concepts, abstractions and notations derived from the problem domain. Considering simplicity as a key factor for usability and efficacy, a DSL might cover only about 80% of a problem domain's complexity, if in return all stakeholders are enabled to learn and use it. By providing various graphical notations and accompanying editors, each of them being as intuitive as possible for a particular stakeholder group, the usability of a DSL can be further improved. In conclusion, stakeholders without software development skills are enabled to directly contribute to the development effort by understanding, validating, adapting and even specifying parts of the solution.

In our approach, a DSL consists of a *Domain-specific Model (DSM)*, one or more *Domain Interaction Models (DIMs)* and a *Solution Building Block (SBB)*. The *DSM*, usually an XML Schema document, specifies the formal schema for all solutions that can be described with the DSL. Based on the *DSM*, a *DIM* defines dedicated (graphical) notations, being as intuitive as possible for a particular stakeholder group. Ideally supported by associated editors, stakeholders use one or more *DIMs* to validate, adapt or create DSL programs. Thereby, they are not confronted with complicated source code, but rather employ concepts, abstractions and notations from the problem domain. A *SBB* is a dedicated software

component being capable of executing DSL programs by adapting its behavior accordingly. The *WebComposition Service Linking System (WSLS)* [12] serves as technical platform for the SBBs. WSLS aims at facilitating the systematic construction and evolution of Web applications by reusing software artifacts and emphasizing the “configuration instead of programming” paradigm. Thus, Web applications can be constructed by assembling SBBs for their various concerns and configuring them with DSL programs at runtime.

4. The Dialog DSL at a Glance

In this section, an overview of the Dialog DSL’s elements, its extension points and the associated process model is given. An in-depth description of these building blocks follows in section 5.

4.1 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. Although DIM notations serve for simplifying and tailoring a DSL to a specific stakeholder group, choosing well-known concepts and abstractions from the problem domain already in the DSM is advisable. 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*.

structures and *interaction elements*. Regarding the interaction elements, we chose to integrate the concepts for specifying interaction elements from the W3C XForms standard [4]. They are a good means for expressing interaction elements within a DSL as they are based on high-level user interaction primitives [26]. 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 by the corresponding extension point, this initial set of interaction structures can also be easily extended.

Domain Interaction Model (DIM): So far, we have developed a two-tiered, Petri net-based DIM notation in accordance to the DSM. 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. With respect to the device-dependent model adaptations at runtime, dedicated symbols allow for marking partitions and groups of interaction elements as non-dividable.

This two-tiered modeling approach fosters reuse and allows for separation of concerns - thus again putting emphasis on usability and simplicity. In addition, we developed a supporting Web-based editor which allows for the comfortable creation and adaption of these models.

Solution Building Block (SBB): The Dialog DSL’s SBB 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 dialogs. Moreover, it links to the Web-based editor for creating and adapting dialogs. Finally, the SBB identifies requesting user

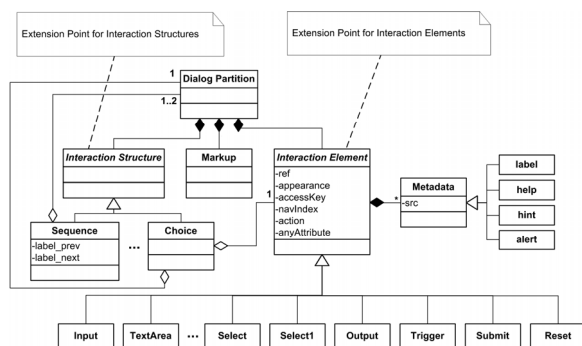


Figure 1. Simplified excerpt from the DSL’s DSM.

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*

agents at runtime and performs corresponding dialog adaptations as well as ultimately transforms dialog models into executable markup, e.g. XForms or XAML [17].

4.2 The Dialog DSL Process Model

Figure 2 shows the Dialog DSL's associated process model for the construction of advanced dialogs. It consists of three phases in the course of a continuous evolution.

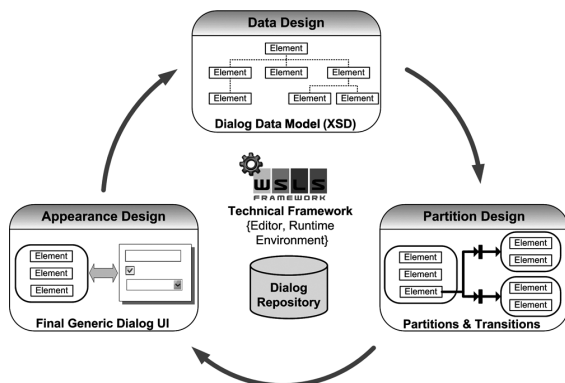


Figure 2. Overview of the evolutionary dialog engineering approach.

Data Design: In this phase, the data model for the dialog being constructed is developed. This can be achieved in several ways: Firstly, supported by sophisticated search mechanisms considering criteria from the development context like the type of application being developed or the workflow context the dialog is used in, a suitable data model can be retrieved from the reuse repository. Secondly, if the data entered in the dialog shall be submitted to a Web service, the target data schema can be extracted from the Web service's WSDL document. Thirdly, the data schema can be elaborated from scratch in strong collaboration with the involved stakeholders, ideally supported by an elicitation tool. The output of this phase is an XML Schema document specifying the dialog's data schema. Based on this schema, the Dialog DSL's technical framework is already able to construct a running dialog that can be directly used in production or further refined using the Web-based editor at runtime.

Partition Design: This phase addresses the modeling of dialog partitions and dynamic behavior. 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, e.g. personal data, travel itinerary

or expenses. Then, employing predefined interaction structures like *Sequence* or *Choice*, dynamic transitions between these partitions are defined. Due to this template-based modeling approach, this phase is ideally supported by a visual drag & drop editor, thereby again emphasizing simplicity and enabling the strong participation of stakeholders.

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. This can be done by either selecting the appearance, i.e. how shall the interaction element being rendered (e.g. select1 either as radio buttons or dropdown list) or exchanging the interaction element type. Considering the final rendering for diverse clients with possibly smaller screen sizes, a dialog partition may be split up into several smaller partitions. To this end, partitions as well as groups of elements therein can be marked as non-dividable. 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. Due to the visual editor, this phase can also be performed in strong collaboration with stakeholders.

Evolution: In case of extensions or modifications in the data model, the technical framework regenerates only these elements that are affected by the change while preserving the rest of the dialog. 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.

5. Dialog DSL Building Blocks

Based on the example scenario presented in section 2.2, this section describes the core pillars of the Dialog DSL approach in detail: First of all, the modeling notation for specifying dynamic behavior and concrete appearance of a dialog and its partitions (section 5.1) as well as a corresponding Web-based editor (section 5.2) are presented. Afterwards, section 5.3 gives an overview of the technical framework for generating, reusing, designing, and rendering dialogs. Finally, the model transformations involved in user-agent related runtime adaptations as well as term rewritings for

transferring DSL programs into executable dialogs and vice versa are described (section 5.4).

5.1 The Modeling Notation

The Domain-specific Model (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*. According to our DSL approach, the Dialog DSL's *Domain Interaction Model (DIM)*, i.e. *the 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 which could arise in complex Petri nets, we predefined a transition template for each interaction structure, thereby simplifying the modeling process.

In order to achieve a good *separation of concerns*, the modeling notation is divided into two tiers: The first tier addresses the modeling of dialog *partitions and transitions* by means of the Petri net transition templates mentioned above. The second tier focuses on the *appearance design* of a partition.

5.1.1 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 Petri net place is marked, its elements are visible. Subsequently, the transitions between these dialog partitions are defined using predefined Petri net transition templates according to the DSL's interaction structures.

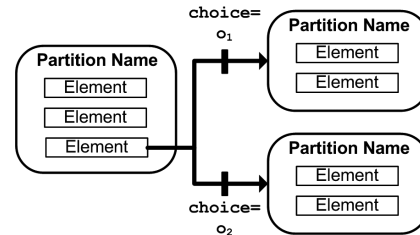


Figure 3. A 'Choice' interaction structure as Petri net transition template.

Figure 3 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 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 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 always connect a Choice transition 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 the partition becomes visible again.

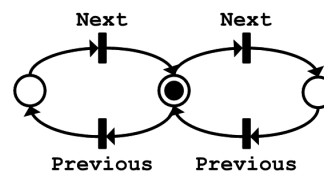


Figure 4. A 'Sequence' interaction structure as Petri net transition template.

The Petri net representation of a *Sequence* Interaction Structure is shown in Figure 4. Here, the

transitions are always connected to the Petri net places as this interaction structure is independent from the data model. It rather represents a wizard-like navigation through a linear space of dialog partitions. When the model is rendered into an executable dialog, corresponding interaction elements (e.g. buttons) allowing the activation of a transition are added to the source partition. Thereby, the labels annotated at the transitions are taken as labels for the interaction elements.

5.1.2 Appearance Modeling Tier

Based on the dialog partitions defined on the superordinate tier, this tier focuses the concrete appearance design of each of these partitions. Figure 5 illustrates a core set of the possible modeling options.

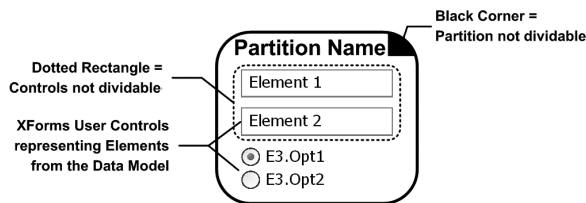


Figure 5. Binding XForms User Controls to data elements and defining semantic groups.

First of all, an XForms user control represented by a corresponding graphical symbol has to be assigned to each data element. Moreover, labels can be defined for each interaction element and additional 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 elements of the partition 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.

5.2 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 6 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.

Each partition contains a button labeled ‘Appearance Design’, which leads the user to the Appearance Design view of the respective partition (Figure 6-2). There, the user can select an interaction element type for each data element. Depending on the element’s data type, a default interaction element has already been assigned. Furthermore, additional markup, e.g. for headings, can be inserted and the relative layout of the interaction elements be defined. Beyond that, a checkbox in the upper left allows for tagging a partition as non-dividable and semantically cohesive element groups can be specified using the floatable Property Editor. In addition, the 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 6 is shown in Figure 9.

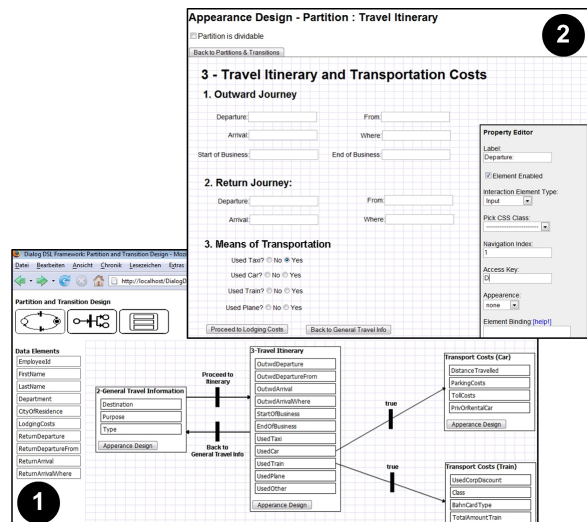


Figure 6. Partition & Transition Design (1) and Appearance Design (2) in the Web-based editor.

Based on our work in the field of Web Accessibility, e.g. [16], we have begun to equip the editor with a proactive accessibility engine advising the user of potentials of improvements already at design time. Beyond that, our technical platform - *the*

WLSLS framework - already allows for runtime accessibility checks of a Web application.

5.3 Technical Platform

Figure 7 gives an overview of the Dialog DSL's technical platform consisting of the Solution Building Block (SBB), the Dialog Web Service, the Web-based editor as well as the underlying WebComposition Service Linking System (WLSLS).

The SBB running on the WLSLS framework can be considered as the main component and thus is in charge of various functions during the development process. In the beginning, it can be either configured with a data schema and submission information or an URL to a WSDL file and the name of the operation the dialog shall be submitted to. The SBB passes the respective data schema to the Dialog Web Service which generates a corresponding raw dialog model and returns it to the SBB. Thereby, an appropriate interaction element is assigned to each element of the data model depending on its data type. If requested, a decomposition of the dialog into partitions derived from the data model's structure is performed. Alternatively, a dialog model from the repository can be searched and reused. From that moment on, a running dialog is already available – without having performed any manual modeling. The dialog can now either be modeled in detail using the Web-based editor or used in production. In either case, no (re)compilation or (re)deployment is required.

If a client requests the Web page containing the dialog, its screen characteristics are identified based on the user agent string contained in the HTTP request or, in case of a mobile device, by evaluating *User Agent Profile (UAProf)* information [23] provided in terms of the W3C *Composite Capability Preferences Profile (CC/PP)* standard. By applying the model transformations presented in the next subsection, the SBB adapts the dialog model accordingly, translates it into executable markup, e.g. XForms, and returns it to the client. In order to achieve an adequate performance, the final markup is cached until the dialog model gets changed and can thus be reused for identical requests.

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 asynchronously communicates with. In the latter case, the SBB receives the response from the Web service and forwards it to the corresponding client.

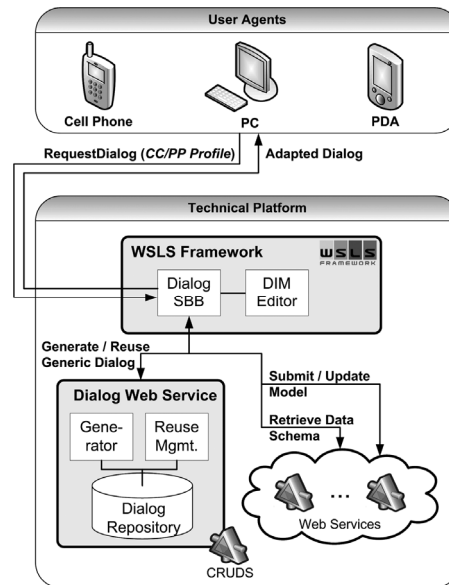


Figure 7. Overview of the technical platform.

5.4 Model Transformations

Within the presented DSL approach, two kinds of model transformations are required. On the one hand, transformations are applied to the dialog model according to the capabilities of the requesting user agent. On the other hand, the dialog model has to be transformed into executable markup, e.g. XForms code.

5.4.1 User-Agent-related Transformations

In our approach, dialogs and their decomposition into partitions are 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, also referred to as 'Pagination'.

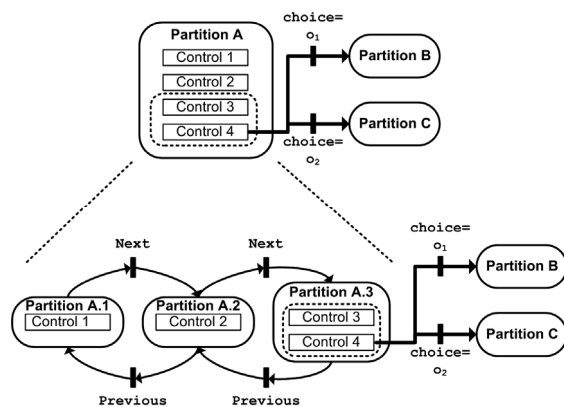


Figure 8. Pagination of a large dialog partition.

Figure 8 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 interconnection of the resulting micro-partitions is realized via the Sequence interaction structure.

5.4.2 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 transformation 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 *Domain-specific Model (DSM)*, and XForms.

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

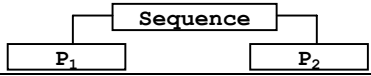
(1) DSM-based pattern	
(2) Context-free grammar rule	Sequence:=P ₁ P ₂
(3) Extended rule	Sequence:=seq(P ₁ , P ₂)
(4) Term rewriting rule	seq(t1, t2) → <pre><switch> <case id="t1">eval(t1)</case> <case id="t2">eval(t2)</case> </switch></pre>

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 like e.g. XAML could be flexibly incorporated.

Figure 9 shows the finally rendered dialog, whose modeling is depicted in Figure 6. When the value of the interaction element labeled ‘Used Train?’ (1) is changed to ‘yes’, it triggers a *Choice* interaction structure, dynamically making the dialog partition

‘Transport Costs (Train)’ (2) visible. At (3), two buttons realizing the *Sequence* interaction structure are located.

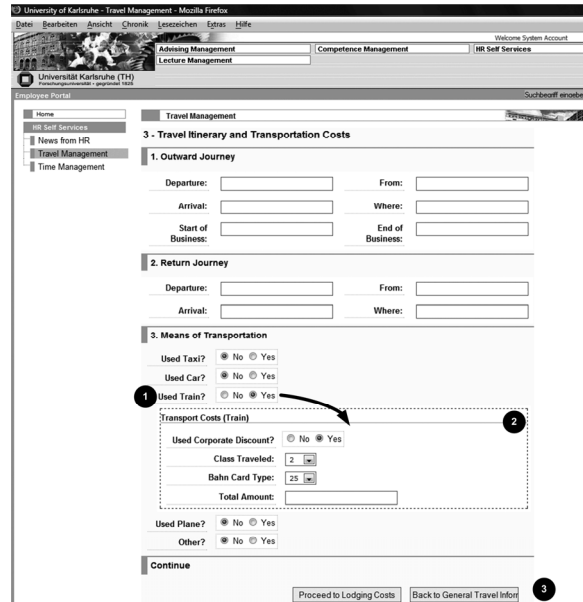


Figure 9. Rendered dialog with dynamic Choice (1+2) and Sequence (3) behavior.

6. Practical Experiences

The Dialog DSL was successfully used for several complex dialogs within the KIM 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. 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.

7. Related Work

In the following, we outline four representative approaches and point out the differences compared with our approach based on the requirements presented in section 2.1.

The UML-based Web Engineering methodology (UWE) [14] allows for the model-driven construction of dialogs using dedicated UML stereotypes. Regarding the modeling of dynamic behavior, first ideas based on UML state charts were proposed [2], but seem not to have been pursued in more detail so far. Due to the implementation-independent modeling approach, dialogs for diverse platforms could be generated by dedicated model transformations. However, platform-independent dialog markup languages like e.g. XForms as well as automatic adaptations according to requesting devices have not been realized so far. Also the consideration of accessibility concerns at design time has not been addressed explicitly. Regarding the requirement of agility, i.e. short evolution cycles and flexible adoptions of changes, UWE models cannot be directly transformed into running applications as there is still some manual source code development required. This also hinders stakeholders without development skills to modify existing dialogs. However, UWE dialog models look rather intuitive, thus improving communications during the development process. An explicit reuse strategy or infrastructure has not been presented yet, even though reusing UWE dialog models is basically possible.

The Object-Oriented Hypermedia (OO-H) method [13] 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. An explicit consideration of accessibility concerns at design time is not apparent.

The Web Modeling Language (WebML) recently presented a dedicated extension towards Rich Internet Applications [5], augmenting the existing WebML modeling notation by the possibility to model dynamic behavior on pages. So far, as WebML is a methodology addressing particularly data-intensive Web applications, the focus lies on dynamic filtering and ordering of data in response to a user's input. More general dynamic behavior as well as reducing cognitive overload and providing user guidance in dialogs have not been considered yet. WebML generally supports mobile devices, even though it does not employ dialog-specific markup languages like XForms or automated dialog adaptations according to the requesting client device. The WebML modeling notation does not address the concrete appearance of a dialog, but rather provides tool-support via a style editor.

The Object-Oriented Hypermedia Design Method (OOHDM) [7] employs Abstract Data View (ADV) models for the specification of dialogs and their dynamic behavior [10]. 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 at runtime as well as the consideration of 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.

8. Conclusion & Future Work

Facing the challenges found in the development and evolution of advanced Web-based dialogs, we presented a systematic engineering approach which is based on a Domain-specific Language (DSL) - the Dialog DSL - and an associated technical framework. This DSL-based approach puts strong emphasis on simplicity, thereby enabling stakeholders to intensely participate in the development process by validating, modifying or even creating dialog models.

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 two-tiered modeling notation employs Petri net

semantics for the decomposition of dialog elements into dialog partitions and the modeling of dynamic transitions between them. Due to a template-based modeling approach, the notation remains rather intuitive. Regarding the modeling of a partition's concrete appearance, well-known dialog control symbols are employed. Dedicated notations allow influencing the *device-adaptive rendering* at runtime.

Furthermore, we presented a corresponding *Web-based editor* supporting the easy yet detailed creation and adaption of dialog models. Modifications to the dialog model can be performed at runtime, thus enabling *short evolution cycles* while preserving model consistency. The Dialog DSL's *technical framework* realizes the sophisticated generation of raw dialogs based on a data schema and facilitates the *reuse* of dialog models. Moreover, it identifies requesting client devices, *adapts the dialog model* accordingly and finally transforms it into executable platform-independent markup (e.g. XForms) employing term rewriting techniques.

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.

9. References

1. KIM Project Homepage - 2005), University of Karlsruhe: <http://www.kim.uni-karlsruhe.de/>
2. Baumeister, H., Koch, N., and Mandel, L.: Towards a UML Extension for Hypermedia Design. in Proceedings of UML '99: The Unified Modeling Language - Beyond the Standard. 1999. Fort Collins, USA: Springer Verlag
3. Beck, K., et al.: Manifesto for Agile Software Development - 2001): <http://www.agilemanifesto.org> (10.11.2006)
4. Boyer, J.M., et al.: XForms 1.0 (Third Edition) - W3C Recommendation (2007):
5. Bozzon, A., et al.: Conceptual Modeling and Code Generation for Rich Internet Applications. in Proceedings of International Conference on Web Engineering 2006. 2006. Menlo Park, USA
6. Consortium, W.W.W.: Web Accessibility Initiative (WAI) Homepage - 2006): <http://www.w3.org/WAI/>
7. Daniel Schwabe, G.R.: An Object Oriented Approach to Web-Based Application Design, in Theory and Practice of Object Systems 1998, Wiley and Sons: New York, USA
8. Deshpande, Y., et al.: Web Engineering. Journal of Web Engineering, 2002. 1(1): p. 3-17
9. Deursen, A.V., Klint, P., and Visser, J.: Domain-Specific Languages: An Annotated Bibliography. ACM SIGPLAN Notices, 2000. 35(6): p. 26-36
10. 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
11. 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.
12. 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
13. 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
14. Hennicker, R. and Koch, N.: Modeling the User Interface of Web Applications with UML. in Proceedings of Practical UML-Based Rigorous Development Methods Workshop at the UML 2001, . 2001: Köllen Druck+Verlag
15. Kappel, G., et al.: Web Engineering: The Discipline of Systematic Development. 1 ed. 2006: Wiley
16. Luque Centeno, V., et al.: Web Composition with WCAG in Mind. in 14th International World Wide Web Conference, International Cross-Disciplinary Workshop on Web Accessibility (W4A). 2005. Chiba, Japan
17. Macvittie, L.A.: XAML in a Nutshell. 2006: O'Reilly Media
18. 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
19. McIlroy, M.D.: Mass Produced Software Components. in Software Engineering; Report on a conference by the NATO Science Committee. 1968. Garmisch, Germany: NATO Scientific Affairs Division, Brussels, Belgium
20. Nielsen, J.: Forms vs. Applications, in Jakob Nielsen's Alertbox. 2005
21. Nussbaumer, M., Freudenstein, P., and Gaedke, M.: The Impact of DSLs for Assembling Web Applications. Engineering Letters, 2006. 13(2006): p. 387-396
22. 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)
23. Open Mobile Alliance: User Agent Profile Specification - 2003): http://www.openmobilealliance.org/release%5Fprogram/uap_v2_0.html
24. Petri, C.A.: Kommunikation mit Automaten. 1962, Technischen Universität Darmstadt: Darmstadt

25. Phifer, G., et al.: Hype Cycle for Web and User Interaction Technologies, 2007, in Gartner Reports. 2007, Gartner, Inc.: Stanford, CT, USA
26. Raman, T.V.: Auditory User Interfaces--Toward The Speaking Computer. 1997: Kluwer Academic Publishers
27. Roger S. Pressman: Part Three: Applying Web Engineering, in Software Engineering: A Practitioner's Approach. 2005, McGraw-Hill: New York. p. 499-626
28. The Standish Group International: CHAOS Research - Research Reports (1994-2005): <http://www.standishgroup.com>
29. Welie, M.V. and Trtteberg, H.: Interaction Patterns in User Interfaces. in Proceedings of 7th Pattern Languages of Programs Conference. 2000. Illinois, USA
30. Wiegers, K.E.: Software Requirements. Second ed. 2003: Microsoft Press. 430 pages