Experiences with a Dialog-Driven Process Model for Web Application Development

Matthias Book, Volker Gruhn
Chair of Applied Telematics/e-Business, Dept. of Computer Science, University of Leipzig
Klostergasse 3, 04109 Leipzig, Germany, Tel. +49-341-97-32337, Fax +49-341-97-32339
{book, gruhn}@ebus.informatik.uni-leipzig.de

Abstract

We present a dialog-driven process model for the development of web-based applications that uses a graphical notation to model and iteratively refine the application's dialog flow, and communicate with non-technical stakeholders in the development process. This way, the user interface can drive the design and implementation of the application logic and data model instead of being dictated by it. After an introduction of the underlying notation and dialog control framework, we present how these tools can support the phases of the development process and discuss experiences gained from the implementation of a web application that was built using this approach.

1. Introduction

Many enterprises are currently connecting their back-end systems to the World Wide Web in order to provide their employees and clients with access to important data and business processes. This way, field staff of insurances, truck drivers of logistics companies and clients of banks (to name just a few) can request information and do business online using a web browser, PDA or mobile phone [9].

We therefore present a dialog-driven process model for the development of web applications that relies on two tools: a dialog flow notation that allows developers to specify a web-based user interface with hierarchical, modular dialog sequences and thus let users work with familiar navigation structures; and a dialog control framework that decouples the device-independent application logic from the interaction patterns required by different devices. We hypothesize that the dialog-driven process model increases web applications' usability while at the same time reducing the development effort by eliminating a number of redundant activities. This supports an agile development process in which the user experience of the final product is driving the design and implementation of the other tiers instead of being dictated by them.

After a brief introduction to the Dialog Flow Notation and Dialog Control Framework in Sect. 2, we will present the phases of the dialog-driven process model in Sect. 3. These stages are illustrated with examples from the development of a web application built to evaluate the feasibility of this approach. We will discuss the experiences gained from that project in more detail in Sect. 4 before giving an overview of the related work (Sect. 5) and drawing conclusions from the project (Sect. 6).

2. Dialog flow specification and control

The Dialog Flow Notation (DFN) [3] models a web-based application's dialog flow as a transition network, i.e. a directed graph of states connected by transitions called a *dialog graph*. The notation refers to the transitions as *events* and to the states as *dialog elements*. Two of the most important dialog elements are *masks* (hypertext pages symbolized by dog-eared sheets) and *actions* (application logic operations symbolized by circles). Every dialog element can generate and receive multiple events (symbolized by arrows). Which element will receive an event depends both on the event and the generating element. Events can carry parameters such as form data or processing results, and thus facilitate communication between dialog elements.

In the DFN, dialog graphs are never free-standing, but always encapsulated in *dialog modules* that facilitate the connection and nesting of dialog graphs, as illustrated e.g. in Fig. 3 using the example of a "Create Account" module. A module typically encapsulates one or more dialog masks, actions and possibly sub-modules implementing a certain functionality, process or behavior in the system (e.g. creating an account, logging in, searching for hotels, booking a room etc.). Any module's dialog graph can contain sub-modules, and any module can itself be embedded in the dialog graphs of various super-modules. The modules' definitions are decoupled from each other, but through the inter-

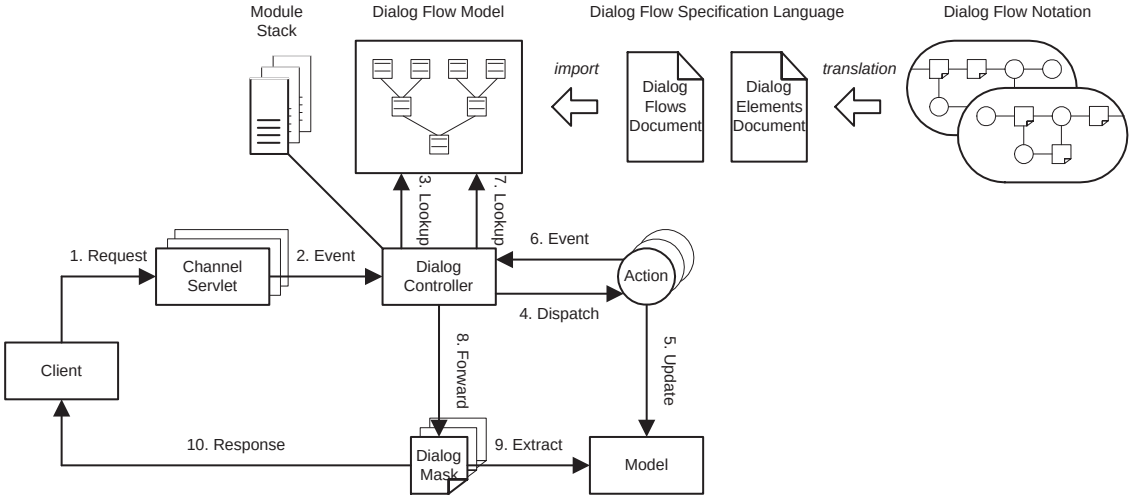

Figure 1. Architecture of the Dialog Control Framework

faces of their initial and terminal events, they can call and return results to each other. This facilitates easy reuse of dialog sequences and enables developers to model the complete dialog flow of an application with a set of dialog modules that are nested into and connected with each other. To model different interaction patterns that may be required by different devices, developers can specify variants of a module's dialog flow by assigning *presentation channel* labels to them (noted in square brackets after the module name).

To enable a smooth transition from models to code, developers can model dialog flows using a graphical editing plug-in for the Eclipse IDE that validates the models and auto-generates machine-readable specifications in the XML-based Dialog Flow Specification Language (DFSL) for interpretation by the Dialog Control Framework (DCF).

Following the MVC paradigm, the DCF architecture (Fig. 1) [2] enforces a strict separation of presentation, application and dialog control logic since neither the masks nor the actions determine the next step in the dialog flow directly. Instead, this decision is made by the dialog controller according to the dialog flow specification.

Upon initialization of the application, the DCF parses the dialog flow specification expressed in the DFSL documents and builds an object-oriented dialog flow model from it. Every time an event comes in from a client through one of the presentation channel servlets (steps 1 and 2), the dialog controller looks up the receiver for this event in the dialog flow model (3). If the receiver is an action, the dialog controller will dispatch the event to the respective object (4), which may update the applications' data model (5) and then returns a new event indicating the result of the operation (6). The dialog controller again looks this event up in the dialog flow model (7), where it may find that it leads to another action (in which case the cycle repeats) or to a module or mask. If the event receiver is a module, the dialog controller will push a reference to it onto the user's module stack in order to reflect the user's updated position in the dialog flow, and then look up the receiver of that module's initial event. If the event receiver is a mask, the dialog controller will dispatch the request to the implementation for the corresponding presentation channel (e.g. a JavaServer Page, step 8), which can read information from the data model (9) to build a response that is finally sent back to the client (10).

3. Dialog-driven process model

Since the DFN does not require technical knowledge of a web application's underlying protocols and technologies, but uses concepts that can be grasped quite intuitively ("masks are what the user sees; actions are what the server does; modules contain processes that the user executes"), it should also be understandable by non-programmers (e.g. audience representatives, usability experts and user interface designers) and thus support communication between the various stakeholders and roles involved in the software development process. The notation lends itself naturally to an iterative and incremental development process: In a top-down approach, coarse drafts of the dialog structure created in early stages of the development process can be refined step by step in subsequent iterations.

In the following subsections, we will introduce a dialog-driven process model (DDPM) that suggests how the DFN and DCF may be employed in the phases of a typical web application development processes. We illustrate the concepts presented for each phase with examples from the development of the ARGuS Travel Guide, a complex web ap-

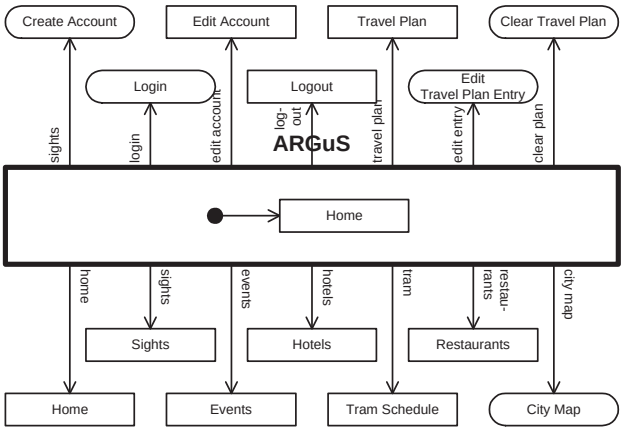

Figure 2. High-level dialog structure of the ARGuS portal

plication built using the DFN and DCF in order to examine the feasibility of this approach.

3.1. Requirements analysis phase

Early in the requirements analysis phase, a very coarse, high-level view of the application's dialog flow, showing just the relationships between the most important compounds, should be drafted in order to visualize the overall structure and scope of the project. This early draft can be derived from use cases and thus initiate the transition from informal requirements to a more formal and detailed specification of the application's look and feel.

The ARGuS Travel Guide, for example, bundles information on Leipzig's sights, restaurants, hotels, and public transportation schedules and makes it available through the web-based user interface of a portal system. Users can search for points of interest and save them in a personal travel planner to create their individual itinerary for a visit to the city. Most features of the system can be accessed either through a desktop browser, PDA or WAP-enabled mobile phone. Consequently, the first step in the dialog flow design was to identify the dialog modules that would later contain these use cases, without specifying their actual dialog graphs yet (Fig. 2).

3.2. Specification phase

The details of the dialog flow should then be worked out in the specification phase, preferably in incremental fashion: In order to ensure that the whole development process is driven by the users' needs, the coarse dialog graphs should be populated with an emphasis on dialog masks first, since these are the entities that will ultimately determine the usability experience. The actions can be specified coarsely at this early stage, and be refined in subsequent iterations.

If the application logic shall serve multiple presentation channels, their dialog flows should be specified in parallel to ensure that the dialog structure is as similar as possible on all channels, allowing users to switch channels without having to rebuild their conceptual model of the application. In this case, giving preference to the masks during the specification and design phase is especially important to prevent the design of the actions from becoming presentation channel-dependent. For example, to create a new account in the system, the user needs to provide quite a bit of data (address, travel preferences and desired password) — while this can be all acquired in one form on the HTML channel, we use a sequence of three pages on the WML channel to cater to mobile devices' smaller screens (Fig. 3).

3.3. Design phase

By the end of the specification phase, the user experience should have been worked out in terms of which masks exist on various channels of the application and how the user can navigate between them. In the design stage, when the structure of the underlying application logic and data model is also designed, it is then time to refine the dialog graphs by inserting all necessary actions between the masks to process the users' input and prepare the system's output.

To complete the dialog graphs of the "Create Account" module, for example, we need to add logic for validating the data entered by the user. In order to keep this logic channel-independent, we implement it in three actions that are executed subsequently on the HTML channel, but interspersed with the masks on the WML channel (Fig. 4). The necessity to distribute the input processing over several actions only becomes obvious if the channels are designed in parallel, and preference is given to the masks — had we designed for the HTML channel only, all the processing might have been implemented in one action that would have been unsuit-

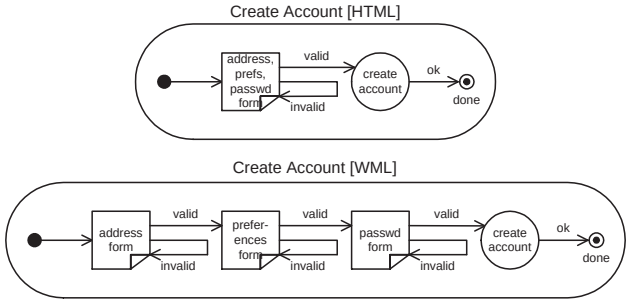

Figure 3. Early sketches of the "Create Account" dialog module

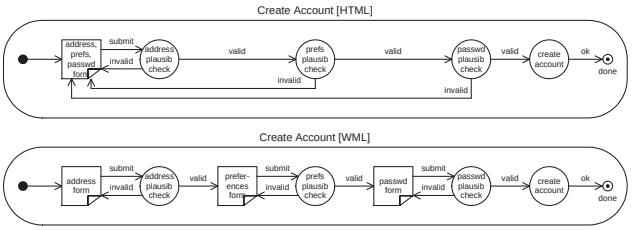

Figure 4. Refined dialog flows of the "Create Account" dialog module

able for reuse on the WML channel. Therefore, if we want to avoid redundant implementation of application logic on multiple channels, the structure of the logic must be flexible enough to serve all channels — and the required degree of flexibility can only be gauged if the channels' user interfaces are specified first.

As the application's data model matures in the design phase, this is also the time for specifying the data flow between the application logic and the user interface. This includes the detailed definition of the masks' contents, i.e. the data that the user should input and the system should output. This stage will likely be characterized by mutual feedback between the dialog flow specification and the application design: The dialog flow defines what the application logic should do in response to user interactions, guiding the design of the logic and data model, which in turn define the data structures that the masks and actions will process.

Obviously, since the dialog flow specification is continuously evolving from the first coarse sketches to the final detailed graphs, no clear lines can be drawn between these three phases. In practice, there will always be some degree of overlap and iteration, depending on the type of application and the actual process employed.

3.4. Implementation and testing phase

In the implementation phase, the refined dialog flow model serves as essential input for the DCF, which will manage the users' interaction with the application accordingly. Since a machine-readable specification can be generated from the dialog flow model automatically, there is no need for developers to implement the specified dialog flows manually. As an example, Figure 5 shows an excerpt from the DFSL representation of the "Create Account" module.

Because of the modular nature of the dialog flow, modules may be added to the dialog flow model incrementally as the implementation of their constituent masks and actions progresses. Developers can also test the interaction of their masks and actions with the application logic incrementally on a relatively low level by embedding them into simple dialog graphs that merely serve as test drivers.

Since the DFN encourages modularity in the dialog flow and the DCF enforces the separation of dialog flow specification, user interface design and application logic, any errors found in tests of one of these tiers should be fixable without affecting other tiers or even other elements of the same tier. In contrast, the intermingling of application logic and dialog control logic in actions that is allowed by approaches such as Apache Struts [1] bears the risk of introducing side effects when changing one aspect of the combined logic.

4. Project experience

To evaluate the practical applicability of the DDPM and the suitability of the DFN and DCF in this context, the ARGuS Travel Guide was implemented by a team of three students with a background in common web engineering technologies, but no prior experience in using the DFN or DCF. Over the course of a half year, the team built a portal system comprising the core application logic and user interface on three presentation channels: rich HTML for desktop browsers, WML for mobile phones, and later, light HTML for PDAs (some special features such as hotel booking and tram routing were not fully implemented due to their integrative or algorithmic complexity that were beyond the focus of this project). Following the DDPM, the team first specified the complete dialog flow in the DFN before translating it into DFSL and implementing the application logic.

```
<dfs-flows>
 <in-module name="create account">
  <channel name="html">
   ...
   <ex-action name="passwd plausib check">
    <on-event name="valid">
     <call-action>create account</call-action>
    </on-event>
    <on-event name="invalid">
     <call-mask>
      address, prefs, passwd form
     </call-mask>
    </on-event>
   </ex-action>
   <ex-action name="create account">
    <on-event name="ok">
     <term-event>done</term-event>
    </on-event>
   </ex-action>
  </channel>
  ...
 </in-module>
 ...
</dfs-flows>
```

Figure 5. Excerpt of the "Create Account" module's dialog flow specification

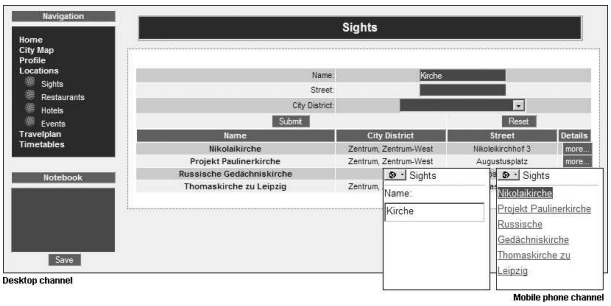

Figure 6. Searching for sights on the desktop and mobile phone channel of ARGuS

While the developers found the basic DFN and DCF concepts easy to learn, they initially found it difficult to differentiate between control flow, data flow and dialog flow. By consistently placing all application logic into the actions, using a mix of dialog events and session variables for data flow and restricting Java code in the masks to simple presentation issues, the team later managed to cut the size of the dialog flow specification in half.

The total time for the specification and design of the application logic and the dialog flows for the HTML and WML channel was about two months. In the first part of the implementation phase, the complete application logic and 24 rich HTML masks were then implemented along with 29 action classes that serve as the interface between the application logic and user interface. This increment took about two months to complete. After a successful system test of the HTML channel, 22 masks for the WML channel were implemented. No additional application logic had to be implemented for this increment since the WML channel's dialog flow had been coordinated with the HTML channel in such a way that the whole application logic could be reused. Consequently, its implementation took only two days.

Late in the project, the team decided to add a light HTML channel serving virtually text-only pages was conceived for use by PDAs and similar devices. This channel completely reused all application logic, as well as the desktop channel's dialog flows as the only difference was in markup, but not in pagination. Since the DFN and DFSL support reuse of specifications with minimal redundancy, this channel required only one day for the implementation of the 24 masks.

The observed implementation times reflect what would be intuitive expectations for the development efforts of these channels: Since the rich and light HTML channels are very similar, one would expect the effort for adding the PDA channel to be quite low. The observed implementation effort of just one day confirms this and suggests that the notation and framework did not introduce any additional overhead. For the implementation of the WML channel, the observed implementation time of two days is a bit higher than for the PDA channel, but still lower than expected. We consider this a pay-off of the preceding two months of specification and design, where the dialog flows were iteratively revised up to a maturity that later actually enabled the developers to simply implement the WML masks without having to deal with any application logic in order to obtain a working mobile channel.

Obviously, these implementation times cannot be generalized — more empiric evidence and a comparison to development projects that do not use the approach suggested here is necessary to draw valid conclusions on the efficiency of this method vs. others. Also, since the rich HTML and WML dialog flows were designed in parallel, it is not possible to quantify how much effort went into which channel. It would be an interesting experiment to add a channel for a device with restricted I/O capabilities late in the project and observe the development effort this incurs. Still, the initial experiences gained from the ARGuS project look promising and show that it is feasible to build complex web-based applications for multiple devices using the DDPM.

5. Related Work

The dialog-driven process model draws part of its efficiency from the compatibility between the Dialog Flow Notation and Dialog Control Framework that allow a seamless transition from models to code.

Other notations suggested for modeling web-based UIs initially focused on data-intensive information systems, but not interaction-intensive applications [5]: For example, the RMM development process [8] allows the definition of navigable relationships between data entities, and the OOHDM [10] process provides classes like node, link and index to represent different forms of navigation; however, they do not provide explicit support for modeling different presentation channels. The language WebML [4] is capable of modeling the layout and appearance of web pages independently of the output device using an abstract XML language for its presentation model, but does not seem to provide an overriding mechanism for the extension of generic dialog modules with channel-specific fragments that enables the easy reuse of dialog sequences across presentation channels. Similarly, the Web Composition Language [6] focuses on the specification of the dialog mask's contents, but seems to lacks tool support for handling navigation patterns.

Most tools offering dialog control implementation support for web applications follow the Front Controller design pattern to facilitate easier dialog control. However, since they lack accompanying notations, they still require developers to manually implement dialog flows that were specified using unrelated notations (if at all). The Apache Jakarta Project's Struts framework [1] is the most popular solution today, however, it forces developers to combine application

logic and dialog control logic in its actions: The Struts controller only decides which action should receive incoming requests, but the actions then decide which view to display next. Since the application logic is thus not completely decoupled from the dialog flow, reusing it on different channels is not always possible, making device-independent design cumbersome.

The need to spread complex forms over multiple interaction steps on small-screen devices instead of presenting them as a whole is addressed by the Renderer-Independent Markup Language (RIML) [11], an extension of XHTML 2.0 which contains semantic information for an automatic pagination engine. Collecting the data fragments coming in from the split-up forms is the task of a proxy between the client and server in that approach. In contrast, we are working on an extension to the DCF that will enable it to manage the necessary micro-dialog flows directly.

6. Conclusion

We presented a dialog-driven process model for the development of web-based applications. Using this approach, developers can derive early drafts of the application's dialog flow from the requirements and then refine them continually using a graphical dialog flow notation that can serves as a communication tool between all involved stakeholders throughout the process. Since the dialog flows of the application are specified first, they are driving the design of the application logic instead of being dictated by it. This encourages compliance with the ISO dialog principles of consistency with user expectations and suitably for the task [7] and thus contributes to the usability of the application.

Based on our experiences gained from the development of the ARGuS web application, we predict that a dialog-driven process using the Dialog Flow Notation and Dialog Control Framework can reduce the development effort and cost for web applications in a number of ways: Firstly, the development effort for application-specific dialog control logic is eliminated since the DCF that contains all this logic can be reused as a black box. Secondly, the redundant effort of first specifying an application's dialog flow in some notation and then manually implementing it is eliminated since the DFN/DFSL specifications serve as direct input to the DCF without the need for further programming. Thirdly, redundant development effort for multiple presentation channels is eliminated since the DFN and DCF encourage reuse of the device-independent application logic and provide means to reuse similar parts of the dialog flow across channels. Fourthly, the risk of late discovery of flaws in the user interface and business processes is reduced since the intuitively understandable DFN allows the involvement of non-technical project stakeholders throughout the project, and the seamless transition from specification to implementation of dialog flows using the DCF allows rapid prototyping. We hypothesize that in combination, these effects will lead to a reduced overall cost of web application development and a shorter time to market, while at the same time providing developers with means to structure web applications' user interfaces on various devices in a more user-friendly way.

7. Acknowledgments

The Chair of Applied Telematics/e-Business is endowed by Deutsche Telekom AG.

References

[1] Apache Software Foundation. Struts. http://struts.apache.org.

[2] M. Book and V. Gruhn. A dialog control framework for hypertext-based applications. In H. Lin and H. Ehrich, editors, *Proceedings of the 3rd International Conference on Quality Software (QSIC 2003)*, pages 170–177. IEEE Computer Society Press, 2003.

[3] M. Book and V. Gruhn. Modeling web-based dialog flows for automatic dialog control. In *19th IEEE International Conference on Automated Software Engineering (ASE 2004)*, pages 100–109. IEEE Computer Society Press, 2004.

[4] S. Ceri, P. Fraternali, and A. Bongio. Web Modeling Language (WebML): a modeling language for designing Web sites. *Computer Networks*, 33:137–157, 2000.

[5] P. Fraternali. Tools and approaches for developing data-intensive web applications: A survey. *ACM Computing Surveys*, 31(3):227–263, Sep 1999.

[6] M. Gaedke, M. Beigl, H. Gellersen, and C. Segor. Web content delivery to heterogeneous mobile platforms. *Advances in Database Technologies, LNCS*, 1552, 1998.

[7] Intl Organization for Standardization. Ergonomic requirements for office work with visual display terminals — Part 10: Dialogue principles. ISO 9241-10, 1996.

[8] T. Isakowitz, E. Stohr, and P. Balasubramanian. RMM: a methodology for structured hypermedia design. *Communications of the ACM*, 38(8):34–44, Aug 1995.

[9] A. Köhler and V. Gruhn. Analysis of mobile business processes for the design of mobile information systems. *5th International Conference on Electronic Commerce and Web Technologies (EC-Web '04), LNCS*, 3182:238–247, 2004.

[10] D. Schwabe and G. Rossi. The object-oriented hypermedia design model. *Communications of the ACM*, 38(8):45–46, Aug 1995.

[11] T. Ziegert, M. Lauff, and L. Heuser. Device independent web applications — the author once – display everywhere approach. *Proceedings of the 4th International Conference on Web Engineering (ICWE 2004), LNCS*, 3140:244–255, 2004.