A Notation and Framework for Dialog Flow Control in Web Applications

Matthias Book and Volker Gruhn

Chair of Applied Telematics / e-Business,[⋆] Department of Computer Science
University of Leipzig, Klostergasse 3, 04109 Leipzig, Germany
{book, gruhn}@ebus.informatik.uni-leipzig.de

Abstract. The usability of web applications today often suffers from the page-based medium's lack of intrinsic support for hierarchical dialog sequences mirroring the parent-child relationships between dialog boxes in window-based user interfaces. For multi-channel applications, an additional challenge lies in reconciling the device-independent business logic with the device-specific interaction patterns necessitated by different clients' input/output capabilities. We therefore present a graphical Dialog Flow Notation that allows the specification of nestable dialog sequences for different presentation channels. These specifications serve as input for a Dialog Control Framework that controls the dialog flows of complex web applications.

1 Introduction

Web engineers are faced with two major challanges today: The first is the difference between page-based and window-based user interface (UI) paradigms: In window-based applications, any window can spawn "child windows", and the completion of a dialog in a child window returns the user to the dialog in the parent window. Users can rely on this predictable behavior that reinforces their conceptual model and thus increases applications' usability [14]. In web applications, however, only simple linear and branched dialog sequences can be implemented with basic session state management techniques, while hierarchical dialog sequences require more complex dialog control logic. Secondly, if an application shall be accessed through a variety of devices, their different input/output (I/O) capabilities affect how users work with an application: A dialog that may be completed in a single step on a desktop browser may have to be broken up into multiple interaction steps on a mobile device. Yet, the server-side business logic should remain independent of such client-side specifics [3, 8]. This obviously calls for a separation of presentation and business logic – however, that is not as trivial as it sounds since the dialog control logic tends to get mixed up with the other tiers. To address the issues of nestable dialogs and device-dependent interaction patterns, we introduce a Dialog Flow Notation that allows the specification of complex dialog flows (section 2), and present a Dialog Control Framework

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

that provides the corresponding dialog control logic for black-box reuse in any application (section 3).

2 The Dialog Flow Notation

The Dialog Flow Notation (DFN) specifies the sequence of UI pages and processing steps in an application, and the data exchanged between them. It models the dialog flow as a transition network called a **dialog graph**. The notation refers to the transitions as **events** and to the states as **dialog elements**. These elements are further divided into hypertext pages (symbolized by dog-eared sheets and referred to by the more generic term **masks** in the DFN) and business logic operations (symbolized by circles and called **actions** here). Every dialog element can generate and receive multiple events. Which element will receive an event depends both on the event and the generating element (e.g., an event e may be received by action *1* if it was generated by mask A, but be received by action *2* if generated by mask B). Events can carry parameters containing form input submitted through a mask or data produced by the business logic to facilitate communication between elements. They are not bound to HTTP requests or responses, but can also link two actions or two masks. The DFN also provides **dialog modules** (symbolized by boxes with rounded corners) which encapsulate dialog graphs and enable the specification of nested dialog structures. When a module receives an event from the exterior dialog graph that it is embedded in, traversal of its interior dialog graph starts with the **initial event**. When the interior dialog graph terminates, it generates a **terminal event** that is propagated to the super-module and continues the traversal of the exterior dialog graph (Fig. 1). For more complex dialog structures, the DFN offers a number of additional event and element types [2] that we will not discuss here in detail.

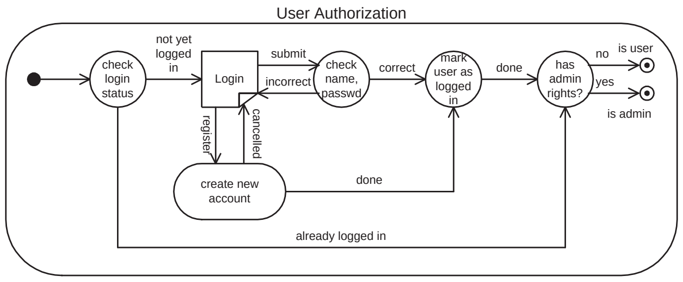

Fig. 1. Example: Dialog graph of *User Authorization* module

To cater to the different interaction patters required for different client devices, the DFN allows the specification of dialog flows for different presentation channels in multiple versions of a module and distinguishing them with **channel labels** (Fig. 2). While the channels employ different dialog masks according to those devices' I/O capabilities, they use the same actions for processing the user's input, as indicated by the shading. This enables developers to reuse the device-independent business logic on multiple channels.

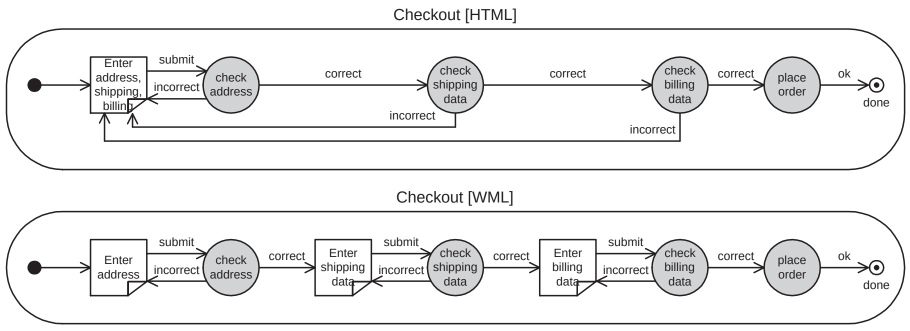

Fig. 2. Example: Dialog graphs of *Checkout* module on HTML and WML channel

3 The Dialog Control Framework

Web applications are usually designed according to the Model-View-Controller (MVC) paradigm [13], which suggests the separation of UI, business logic and control logic. The Dialog Control Framework (DCF) features a very strict implementation of the MVC pattern, completely separating not only the business logic and UI, but also the dialog flow specification and dialog control logic. As the coarse architecture (Fig. 3) shows, the action objects contain only calls to the business logic. The generic dialog control logic is contained in the **dialog controller** that receives events coming in through **channel servlets** on each presentation channel. It looks up the receivers of these events in the **dialog flow model** – a collection of objects representing dialog elements that hold references to each other to mirror the dialog flow, built upon initialization of the framework by parsing an XML-based representation of the graphical dialog flow specification. Depending on the receiver that the controller retrieved from the model for an event, it may call an action, forward the request to a mask, nest or terminate modules. The latter operations are performed on **module stacks**.

Due to the strict separation of tiers, device-independent applications can be built with minimal redundancy: Only the dialog masks and the dialog flow specifications need to be specified for the different presentation channels, while the business logic is implemented device-independently only once and the dialog control logic is provided by the framework. Since the dialog controller is aware of the whole dialog flow specified for each channel, it can manage complex dialog constructs such as nesting modules that would be hard to realize if the dialog control logic was distributed over all action objects.

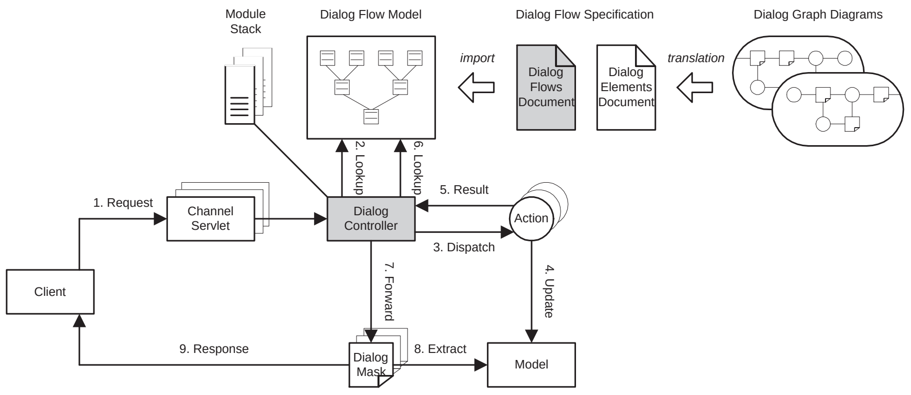

Fig. 3. Coarse architecture of the Dialog Control Framework

4 Related Work

Most tools offering dialog control implementation support for web applications follow the MVC design pattern to facilitate easier dialog control. The Apache Jakarta Project's Struts framework [1] is the most popular solution today, however, it forces developers to combine business logic and dialog control logic in the action objects, which renders the dialog control implementation cumbersome and inflexible. While the concept of an application-independent "screen flow manager" that determines the next view is described in the Java BluePrints [16], no framework seems to exist yet that employs this pattern to implement complex dialog constructs such as the arbitrarily nestable modules and device-specific dialog flows offered by the DCF. The World Wide Web Consortium's XForms initiative [5] is mostly concerned with the specification of widgets on pages and does not support nestable dialog modules.

Notations for the specification of web-based UIs mostly focus on data-intensive information systems, but not interaction-intensive applications [6]: Development processes such as RMM [11] and OOHDM [15], modeling notations and languages such as HDM-lite (used by the Autoweb tool [7]), and WebML [4] support the generation of web pages out of a large, structured data basis or provide dynamic views on database content, but do not allow the specification of highly interactive features with modular, nested dialog structures.

While the concept of modeling dialog systems as state-based systems is not new [9] and generic notations for this already exist (e.g. Statecharts [10]), we chose not to use any generic notation because expressing the particularities of web-based dialog flows (e.g. different dialog elements, modules and events) in those would be cumbersome in practice. Also, we wanted to provide the DFN with constructive instead of mere descriptive power, enabling developers to use complex dialog constructs intuitively without having to spell out their details in a generic notation.

5 Conclusions

The pragmatic approach advocated above notwithstanding, we are currently working on the definition of formal semantics that will enable us to reason about the specifications produced with the DFN (in addition to the operational semantics already defined by the DCF implementation). This can be achieved by showing that all DFN constructs can also be expressed by means of a more generic formalism, even if that would not be suitable for practical use.

While related methodologies tend to derive the web-based UI more or less directly from an established data model, the DFN does not make any assumptions about the underlying data model, but exclusively describes the users' interaction with the system. Thus, it should support a dialog-driven development process that emphasizes the ISO dialog principles of suitability for the task and conformity with user expectations [12] from the start.

References

1. Apache Jakarta Project. Struts. http://jakarta.apache.org/struts/
2. Book, M., Gruhn, V.: A Dialog Control Framework for Hypertext-based Applications. Proc. 3rd Intl Conf. on Quality Software (QSIC 2003), IEEE Press, 170–177
3. Butler, M., Giannetti, F., Gimson, R., et al.: Device Independence and the Web. IEEE Computing **6**, 5 (2002), 81–86
4. Ceri, S., Fraternali, P., Bongio, A.: Web Modeling Language (WebML): A Modeling Language for Designing Web Sites. Computer Networks **33** (2000), 137–157
5. Dubinko, M., Klotz, L.L., Merrick, R., et al.: XForms 1.0, W3C Recommendation (2003). http://www.w3.org/TR/2003/REC-xforms-20031014/
6. Fraternali, P.: Tools and Approaches for Developing Data-Intensive Web Applications: A Survey. ACM Computing Surveys **31**, 3 (1999), 227–263
7. Fraternali, P., Paolini, P.: Model-Driven Development of Web Applications: The Autoweb System. ACM Trans. on Information Systems **28**, 4 (2000), 323–382
8. Gaedke, M., Beigl, M., Gellersen, H.-W., et al.: Web Content Delivery to Heterogeneous Mobile Platforms. Advances in Database Technologies, LNCS 1552 (1998)
9. Green, M.: A Survey of Three Dialogue Models. ACM Trans. on Graphics **5**, 3 (1986), 244–275
10. Harel, D.: Statecharts: A visual formalism for complex systems. Science of Computer Programming **8**, 3 (1987), 231–274
11. Isakowitz, T., Stohr, E. A., Balasubramanian, P.: RMM: a methodology for structured hypermedia design. Comm. ACM **38**, 8 (1995), 34–44
12. International Organization for Standardization: Ergonomic requirements for office work with visual display terminals (VDTs) – Part 10: Dialogue principles. ISO 9241-10 (1996)
13. Krasner, G.E.: A Cookbook for using the Model-View-Controller User Interface Paradigm in Smalltalk. Journ. of Object-Oriented Programming **1**, 3 (1988), 26–49
14. Rice, J., Farquhar, A., Piernot, P., et al.: Using the web instead of a window system. Proc. CHI '96, 103–110. ACM Press (1996)
15. Schwabe, D., Rossi, G.: The object-oriented hypermedia design model. Comm. ACM **38**, 8 (1995), 45–46
16. Singh, I., Stearns, B., Johnson, M., et al.: Designing Enterprise Applications with the J2EE Platform, 2nd Edition. Addison-Wesley (2002)