A DIALOG FLOW NOTATION FOR WEB-BASED APPLICATIONS

Matthias Book, Volker Gruhn
Chair of Applied Telematics/e-Business,* Department of Computer Science, University of Leipzig
Klostergasse 3, 04109 Leipzig, Germany
{matthias.book, volker.gruhn}@informatik.uni-leipzig.de

Abstract

Increasingly, client-server applications are implemented as web-based applications with user interfaces consisting entirely of web pages or equivalent renderings on other presentation channels (e.g. mobile or speech-based devices). However, the page-based medium and the stateless pull communication impose restrictions on the user interface that often manifest themselves in unsatisfactory dialog control, i.e. possibly severely diminished usability. We therefore present a Dialog Flow Notation that allows developers to encapsulate sequences of multiple dialog steps into reusable dialog modules that can be nested arbitrarily, and to specify different interaction patterns for different devices. The notation is complemented with a Dialog Control Framework that manages dialog flows on multiple channels, leaving only the tasks of implementing the device-independent application logic, designing the interface pages, and specifying the dialog flow to the developer.

Key Words

Software Tools and Techniques, Web-Based Software Engineering, Modelling Languages, Framework Techniques

1. Motivation

Since the introduction of the World Wide Web, the services that web sites provide to users have continuously become more sophisticated: From presenting static *information,* sites have evolved to offer simple one-step *interactions* (e.g. search engines), then multi-step *transactions* (e.g. shopping carts), and now complex *applications* (e.g. e-mail readers, customer support systems, portal systems), where the user interface (UI) consists of web pages presented in a browser [1]. Compared to window-based UIs, web-based UIs require only modest client capabilities, making them suitable for a wide variety of client platforms, especially thin clients such as mobile devices with their strict energy, memory, input and output limitations [2]. Furthermore, the simple information elements and interaction techniques of web-based UIs can be rendered on various presentation channels, ranging from desktop to mobile devices (e.g. PDAs or cell phones) and from visual to auditory or haptic interfaces (e.g. screen readers or Braille lines) [3]. With reasonable effort, such a wide variety of clients can only be served by employing the thin client principle, where the application logic is implemented presentation channel-independently on a central server, while the UI is rendered individually on various client devices [4]. However, when developing applications with web-based UIs, software engineers need to be aware that their implementation differs in some important characteristics from applications with window-based UIs ([5], [6]):

Firstly, different presentation channels have different input and output capabilities (e.g. regarding screen size), possibly restricting the amount of information users can work with at a time. Consequently, presentation channel-independent applications must not only implement different UIs, but also be able to handle different interaction patterns. Secondly, web-based UIs present information on pages instead in windows. Consequently, interactions that would be performed without involving the application logic in a window-based UI (e.g. closing a window) require the generation of a new page in a page-based UI and thus involve the application logic for every interaction step. Thirdly, web-based UIs employ a request-response mechanism to pull data from the server. Since the application logic cannot push data to the client, it can only react passively to user actions (e.g. clicking on a link) instead of actively initiating dialog steps (e.g. opening a new window). Finally, HTTP is stateless: The protocol only transports data, but does not maintain any information on the state of the dialog system. Consequently, the application itself has to manage the dialog state for each user session, which requires complicated logic for complex dialog structures.

Regarding the impact of these characteristics on the user experience, one of the most notable effects is the limitation to simple dialog structures in many web-based applications today: Linear and branched dialog sequences can be easily implemented and are therefore commonplace, but already simple nested structures (e.g. an authorization form inserted at the beginning of a sensitive transaction) require a lot of dialog control logic, and no application that the authors are aware of is capable of nesting arbitrary dialogs on multiple levels.

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

Since users have a long-established conceptual model of nested dialogs from window-based applications, they will likely transfer that model to web-based applications. However, because of insufficient dialog control logic, many applications still violate users' expectations today when they send them to other pages than they intended to reach (e.g., in some web applications, login forms return users to the homepage after a successful login instead of sending them to the area that required authorization, forcing them to navigate manually to the desired area). This violation of the ISO dialog principles of controllability and conformity with user expectations [7] imposes a high cognitive and memory load on the user.

Since these challenges are independent of a specific application, a desirable solution would be a notation and a framework that can be used for the specification and implementation of any web-based application. After giving an overview of the related work (section 2), this paper will therefore introduce a Dialog Flow Notation for specifying complex, nested dialog flows (section 3), and present the architecture of a Dialog Control Framework for managing them on different devices (section 4).

2. Related Work

A number of notations for the specification of hypertext systems have been proposed over time. However, they mostly focus on *data-intensive* information systems, but not *interaction-intensive* applications [8]: Development processes such as RMM [9] and OOHDM [10], modeling notations and languages such as HDM-lite (used by the Autoweb tool [11]), WebML [12] and DoDL [13] 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.

Regarding the implementation of web-based applications, several frameworks exist that separate the user interface from the application logic to facilitate easier dialog control, as suggested by the Model-View-Controller (MVC) design pattern [14]. The Apache Jakarta Project's Struts framework [15] is one of the most popular solutions today. However, Struts forces developers to combine dialog control logic and application logic in the Model implementation, since the Controller does not implement any actual dialog control logic, but merely maps action names to class names (a more thorough discussion of the Struts approach vs. the one suggested in this paper will be presented in section 4).

The challenges of device-independent design are addressed in the Sisl (Several Interfaces, Single Logic) approach [16]. It inserts a so-called "service monitor" between the central application logic and the presentation logic for each device type to coordinate the events that the interface can generate with the events that the application logic can currently handle. This allows Sisl to support a wide spectrum of devices, including speech recognition systems, and handle the partial or unordered input that they may produce. Still, Sisl seems more suitable for simple prompt- or menu-based scenarios than for highly interactive applications with complex dialog structures.

We are still missing a solution that controls the dialog structure of a web-based application independently of the implementation of the Model and View tiers, supports different interaction patterns on different devices, and allows developers to work with complex dialog constructs like dialog modules nested on multiple levels. The Dialog Control Framework and Dialog Flow Notation introduced in this paper are designed to address these needs.

3. Dialog Flow Notation

To define the concept of a "dialog flow" and develop the elements of the Dialog Flow Notation (DFN), we first examine the client-server communication taking place when users work with a web-based application.

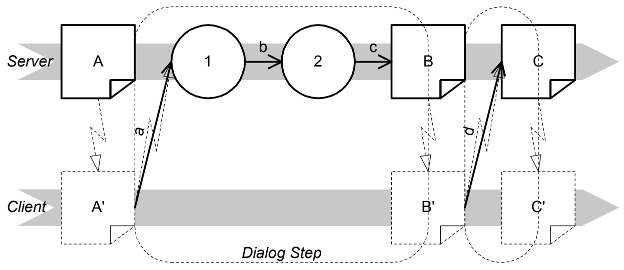

Figure 1. Client-server communication in HTTP

As Figure 1 shows, a page *A'* displayed on the client is rendered from source code (e.g. HTML) that was first generated by an entity *A* (e.g. a JavaServer Page) on the server and then transmitted to the client. When the user follows a link or submits a form on this page, the resulting data *a* is transmitted to the server. The application logic may now process the data in a number of steps (here: *1* and *2*), which each generate data (*b* and *c*) that is processed in the next step. Finally, the source code for the following page is generated (*B*), transmitted to the client and rendered there (*B'*). Alternatively, user-submitted data (such as *d*) may not require any application logic processing, but directly lead to the generation of a new page (*C* and *C'*).

We call the server activity happening between the submission of a request and the receipt of a response by the client a **dialog step** (in an online shop, for example, a dialog step might begin with submission of the user's billing information, comprise the validation of his credit card data by the application logic, and end with the generation of a confirmation page). Multiple consecutive dialog steps form a **dialog sequence** – for example, an online shop's checkout dialog sequence might be composed of several dialog steps for collecting the user's address, shipping options, and billing information. Finally, all possible dialog sequences that can be performed on a certain presentation channel of an application constitute that channel's **dialog flow**. An online shop's dialog flow might for example comprise searching for products, putting products into the cart, checking out, etc.

3.1 Notation Elements

Looking back at Figure 1, we realize that the client-server communication and thus the distinction between generating (*A*) and rendering pages (*A'*) is irrelevant for the purpose of modeling dialog flows: When specifying how the user interacts with the application logic via the UI pages, the dialog flow designer does not need to know about technical details such as pages' source code being generated on the server and transmitted to the client prior to rendering. The DFN therefore only specifies the order of the UI pages and processing steps, and the data exchanged between them. It also does not specify details about any UI widgets or client-side scripting, since those may be presentation channel-specific while the notation is channel-independent.

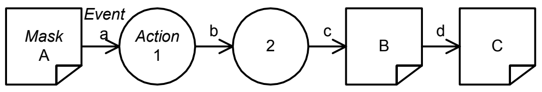

Figure 2. Dialog graph

The DFN models the dialog flow as a transition network, i.e. a directed graph of states connected by transitions called a **dialog graph** (Figure 2).[1] As illustrated in the communication model in Figure 1, dialog graphs do not need to be bipartite. The notation refers to the transitions as **events** and to the states as **dialog elements**, discerning atomic and modular elements. **Atomic dialog elements** are hypertext pages (symbolized by dog-eared sheets and referred to by the more generic term **masks** here) and application logic operations (symbolized by circles and called **actions** from now on). Every dialog element can generate and receive multiple events, enabling the developer to specify complex dialog graphs. 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 *3* if it was generated by mask *D*, but be received by action *4* if generated by mask *E*). Events can carry parameters, e.g. application-specific form input submitted from a mask, and thus facilitate communication between elements.

Theoretically, the complete dialog flow of an application could be described using only atomic elements. However, the resulting specification would be much too complicated to understand, and the "flat" structure does not support reuse of often-needed dialog graphs. The DFN therefore provides **dialog modules** (symbolized by boxes with rounded corners) which encapsulate dialog graphs and realize nested dialog structures: A module's **interior dialog graph** can contain sub-module, and the module itself can be embedded in the **exterior dialog graphs** of super-modules.

Every dialog module has one entry point and one or more exit points: When a module receives an event from an exterior dialog graph, traversal of its interior dialog graph starts with the **initial event**. When the interior dialog graph terminates, it generates an appropriate **terminal event** that is propagated to the super-module and continues the traversal of the exterior dialog graph.

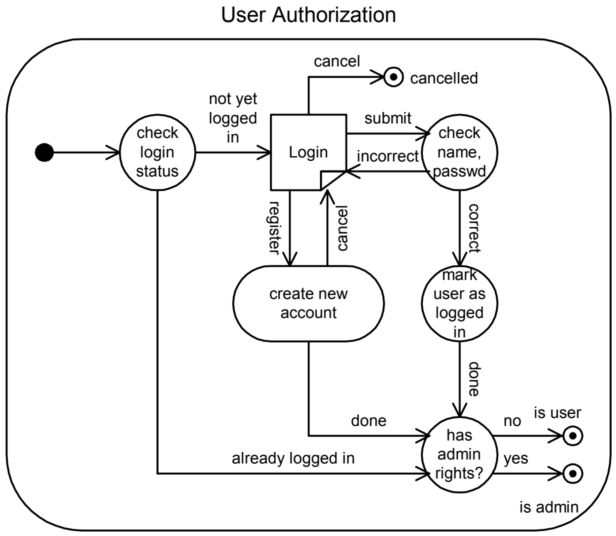

Figure 3. *User Authorization* dialog module

We will introduce the features of dialog modules using the *User Authorization* module in Figure 3 as an example. This module checks if the user is already logged in and shows a *Login* mask to prompt for his user name and password, if necessary. If the user's credentials are correct, the module marks him as logged in, checks his access rights and terminates, notifying the super-module of the user's status. If the user does not yet have an account, he can register using the embedded *create new account* sub-module. By splitting the application logic into fine-grained operations, the module can react flexibly to different situations, like bypassing the credential check when the user is already logged in.

To specify more complex dialog structures, the DFN provides a number of additional constructs that we will not discuss here for the sake of brevity.

3.2 Presentation Channels

The notation elements introduced so far allow developers to specify complex, hierarchical dialog flows. However, we still need a way to specify the presentation channel-dependent dialog flows required for different client devices. In the DFN, this can be achieved by

[1] The basic concepts and symbols of this notation were inspired by Harel's Statecharts [17], but their semantics have been adapted for the context of hypertext dialog flow specification.

specifying the dialog flows for different media in separate dialog modules and adding labels for the respective channels in square brackets after the module's name.

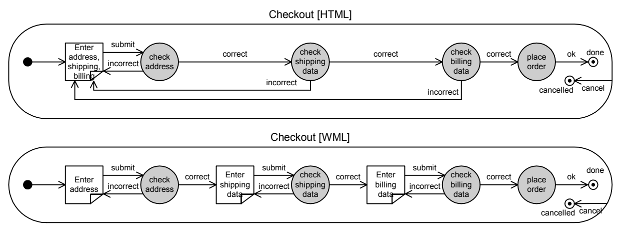

Figure 4. *Checkout* module on HTML and WML presentation channel

For example, Figure 4 specifies the dialog flows for a *Checkout* module on the HTML and WML presentation channel. Note that while the channels employ different dialog masks according to the clients' input/output capabilities, they use the same actions for processing the user's input, as indicated by the shading. This enables developers to implement the device-independent application logic only once and then reuse it for multiple presentation channels. If the actions were designed with sufficient granularity, further presentation channels can be added just by implementing the respective masks and specifying the new channels' dialog flows.

3.3 Dialog Flow Specification Language

After the dialog flows of an application have been specified in dialog graph diagrams, an efficient transition from specification to implementation is desirable: The dialog graph diagrams should not just visualize the dialog flow, still requiring developers to implement the appropriate dialog control manually, but rather serve as direct input for an application-independent dialog control logic, instructing it how to handle events. This can be achieved by transforming the diagrams into documents written in the **Dialog Flow Specification Language (DFSL)**, an XML-based language consisting of elements that mirror the Dialog Flow Notation's dialog elements, events and constructs. For the sake of brevity, we will not discuss DFSL elements and the rules for transforming DFN diagrams into DFSL documents in more detail here.

4. Dialog Control Framework

The dialog control logic that reads the DFSL documents and manages the dialog flow accordingly is application-independent. Therefore, we implemented it in a Dialog Control Framework that can be reused for any web-based application and presentation channel.

Web-based applications are usually designed according to the Model-View-Controller (MVC) paradigm [14], which suggests the separation of user interface, application logic and control logic. While user interface and application logic can be distinguished quite naturally ("what the user sees" vs. "what the system does"), the distinction between application logic and dialog control logic is much more subtle ("what the system does" vs. "what it should do next, based on the user's input"). Therefore, it is easy to mix up the implementation of application and dialog control logic, even if both are separated well from the presentation logic.

4.1 Struts: Decentralized Dialog Control

For example, in the Jakarta Struts framework [15], the dialog flow is controlled by so-called `Action` objects. They implement the application logic and also decide where to forward a request, while the controller just executes that forward command (Figure 5):

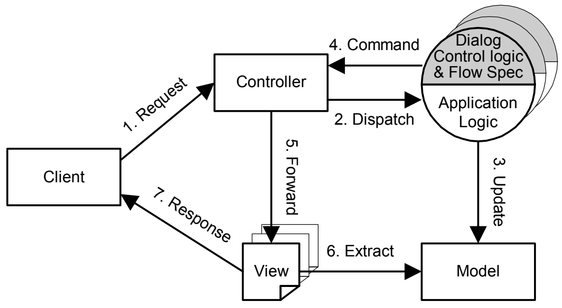

Figure 5. Struts architecture (coarse)

As indicated by the shading in the figure, the dialog control logic is not specified outside the application, but distributed over all actions in the Struts approach. This allows the actions to make only relatively isolated dialog flow decisions, and hampers the implementation of more complex dialog structures with constructs like nested dialog modules. To raise the actions' awareness of the "big picture" and enable them to control more complex constructs, still more control logic would have to be implemented in them, exacerbating the problem. Also, the hard-coded decentralized implementation of the dialog control logic is relatively inflexible, almost unsuitable for reuse and hard to maintain. Finally, achieving presentation channel independence would require additional effort and possibly redundant work: Since the dialog flow depends on the presentation channel, while the application logic does not, their close coupling prevents the reuse of actions on multiple presentation channels. Instead, each presentation channel would require its own set of `Action` objects to implement the individual dialog flow for the respective devices.

4.2 DCF: Centralized Dialog Control

In contrast, the Dialog Control Framework (DCF) presented in this paper features a very strict implementation of the MVC pattern, completely separating not only application logic and user interface, but also dialog flow specification and dialog control logic:

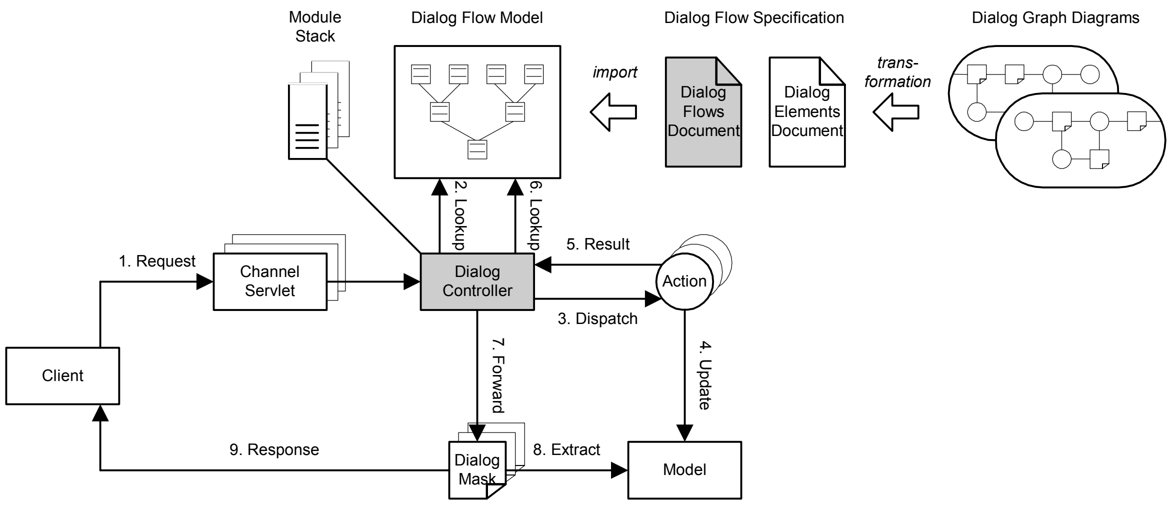

Figure 6. Dialog Control Framework architecture (coarse)

The controller decides where to forward requests by looking up the receivers of events generated by elements in a central dialog flow model [18]. As the coarse architecture in Figure 6 shows, the actions are relatively lightweight here since they contain only application logic, while all dialog control logic has been moved to the **dialog controller**. This controller does not receive requests from the clients directly anymore. Instead, on each presentation channel, it receives events that have been extracted from the requests by **channel servlets**. The dialog controller looks up the receivers for 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. This object structure is built upon initialization of the framework by parsing the DFSL documents containing the dialog flow specification. Depending on the receiver that the controller retrieved from the dialog flow model for an event, it may call an action or forward the request to a mask. If modules shall be activated or terminated, the dialog controller pushes them onto or retrieves them from the user's **module stack** and then looks up the next event in the dialog flow model.

This centralized dialog control solution has three advantages over the previously discussed approach: Firstly, the strict separation between application logic implementation, UI design, dialog flow specification and dialog control logic enables a high degree of flexibility, reusability and maintainability for the components of all four tiers. Secondly, due to this clean separation, presentation channel-independent applications can be built with minimal redundancy: Only the dialog masks and the dialog flow specifications for the different channels have to be adapted, while the application logic is implemented only once and the dialog control logic is provided by the framework. Finally, since the central dialog control logic is aware of the whole dialog flow (it knows the "big picture"), it can provide mechanisms for the realization of complex dialog constructs.

To build an application with this framework, developers do not need to know about the inner structure or implementation of the framework. They only need to provide classes implementing the actions, JavaServer Pages implementing the dialog masks, DFSL documents specifying the dialog flow and mapping elements to their implementing entities, and if required, channel servlets for various presentation channels (the prototype framework implemented by the authors already provides `HTMLChannel` and `WMLChannel` servlets). Since these deliverables are completely application-specific, the framework is suitable for black box reuse.

To validate the suitability of the DFN, DFSL and DCF for practical use, a "Travel Planner" demo application that employs all dialog control features was developed at the Chair of Applied Telematics' Mobile Technology Lab.

5. Conclusions and Further Research

We presented a Dialog Flow Notation and Dialog Control Framework for the specification and management of dialog flows in web-based applications. The framework not only strictly distinguishes application logic, user interface and dialog control, but also separates the dialog control logic from the dialog flow specification, enabling developers to work with nestable dialog modules on different client devices. The dialog flows specified in the graphical notation can be refined incrementally throughout the software development process before being transformed into DFSL documents that serve as input to the framework, allowing for a smooth transition from specification to implementation.

While the framework implementation already defines operational semantics for all notation constructs, further work with the notation obviously requires a more solid foundation. We are therefore currently developing formal semantics for the Dialog Flow Notation.

A weak point of the notation may be the fine granularity of actions that is required to employ them flexibly on different presentation channels (this especially concerns actions responsible for processing user input submitted through forms): The finer the actions are grained, the easier it is to adapt to different interaction patterns – however, finer granularity also results in higher specification, implementation and performance overhead. Research on solutions to this dilemma is in progress.

Another issue that merits further research is the framework's robustness against backtracking: On the Web, clicking the browser's "back" button is the second most frequent user activity after clicking on a link [19]. It must therefore be regarded as a normal interaction pattern that the application should be able to handle as well as regular clicks on links. Backtracking aims to revisit a previous dialog mask without changing the application's data model. This is a challenge since the user events that are recreated through backtracking often lead to actions, which perform application-logic operations before the dialog step finally completes with displaying a mask. Research on a mechanism that allows backtracking even through nested dialog modules is currently in progress.

Finally, we are striving to gain empiric experiences from larger projects, which should yield insights into the applicability and possible limitations of the notation and framework in certain application domains or on certain presentation channels, and their integration into various software development process models.

References

[1] L. Baresi, F. Garzotto, & P. Paolini, From Web Sites to Web Applications: New Issues for Conceptual Modeling, *ER'2000 Workshop on Conceptual Modeling and the Web, Lecture Notes in Computer Science 1921*, 2000, 89-100.

[2] M. Gaedke, M. Beigl, H.W. Gellersen, & C. Segor, Web Content Delivery to Heterogenous Mobile Platforms, *Advances in Database Technologies, Lecture Notes in Computer Science 1552,* 1998.

[3] M. Butler, F. Giannetti, R. Gimson, & T. Wiley, Device Independence and the Web, *IEEE Computing 6*(5), 2002, 81-86.

[4] A. Sinha, Client-Server Computing, *Communications of the ACM 35*(7), 1992, 77-98.

[5] J. Rice, A. Farquhar, P. Piernot, & T. Gruber, Using the web instead of a window system, *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '96),* Vancouver, BC, Canada, 1996.

[6] W. Zhao, D. Kearney, & G. Gioiosa, Architectures for Web Based Applications, *4th Australasian Workshop on Software and Systems Architectures (AWSA 2002),* Sydney, Australia, 2002, www.dstc.monash.edu.au/awsa2002/papers/Zhao.pdf

[7] International Organization for Standardization, Ergonomic requirements for office work with visual display terminals (VDTs) – Part 10: Dialogue principles, *ISO 9241-10,* 1996.

[8] P. Fraternali, Tools and Approaches for Developing Data-Intensive Web Applications: A Survey, *ACM Computing Surveys 31*(3), 1999, 227-263.

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

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

[11] P. Fraternali & P. Paolini, Model-Driven Development of Web Applications: The Autoweb System, *ACM Transactions on Information Systems 28*(4), 2000, 323-382.

[12] S. Ceri, P. Fraternali, & A. Bongio, Web Modeling Language (WebML): A Modeling Language for Designing Web Sites, *Computer Networks 33,* 2000, 137-157.

[13] E.E. Doberkat, A Language for Specifying Hyperdocuments, *Software – Concepts and Tools 17,* 1996, 163-172.

[14] G.E. Krasner, A Cookbook for using the Model-View-Controller User Interface Paradigm in Smalltalk, *Journal of Object-Oriented Programming 1*(3), 1988, 26-49.

[15] Apache Jakarta Project, Struts, http://jakarta.apache.org/struts/

[16] T. Ball, C. Colby, P. Danielsen, L.J. Jagadeesan, R. Jagadeesan, K. Läufer, P. Mataga, & K. Rehor, Sisl: Several Interfaces, Single Logic, *International Journal of Speech Technology 3*(2), 2000, 91-106.

[17] D. Harel, Statecharts: A visual formalism for complex systems, *Scientific Computer Programming 8*(3), 1987, 231-274.

[18] I. Singh, B. Stearns, M. Johnson, et al., *Designing Enterprise Applications with the J2EE Platform, 2nd Edition* (Addison-Wesley, 2002).

[19] L.D. Catledge & J.E. Pitkow, Characterizing Browsing Strategies in the World Wide Web, *Computer Networks and ISDN Systems 27,* 1995, 1065-1073.