

Modelling Safe Interface Interactions in Web Applications

Marco Brambilla¹, Jordi Cabot², and Michael Grossniklaus¹

¹ Dipartimento di Elettronica e Informazione, Politecnico di Milano
Piazza Leonardo da Vinci 32, I-20133 Milano, Italy
{mbrambill|grossniklaus}@elet.polimi.it

² Department of Computer Science, University of Toronto
St. George Street 140, M5S 3G4 Toronto, Canada
jcabot@cs.toronto.edu

Abstract. Current Web applications embed sophisticated user interfaces and business logic. The original interaction paradigm of the Web based on static content pages that are browsed by hyperlinks is, therefore, not valid anymore. In this paper, we advocate a paradigm shift for browsers and Web applications, that improves the management of user interaction and browsing history. *Pages* are replaced by *States* as basic navigation nodes, and *Back/Forward* navigation along the browsing history is replaced by a full-fledged interactive application paradigm, supporting transactions at the interface level and featuring *Undo/Redo* capabilities. This new paradigm offers a safer and more precise interaction model, protecting the user from unexpected behaviours of the applications and the browser.

1 Introduction

The Web has evolved from a platform for navigating hypertext documents to a platform for implementing complex business applications. Pages nowadays contain *complex business logics* both at the client and at the server side. User interaction is able to deal with *several kinds of events*, generated both by users and systems. In this context, the original interaction paradigm of the Web is not valid anymore.

Browsers themselves, that still provide the traditional features of *Back* and *Forward* page navigation along the browsing history, are inadequate for dealing with the complexity of current applications [1]. Several user events are missed by this paradigm, and the Web application behaviour is indeterministic with respect to the actions allowed by the different browsers. Results for the same application vary depending on the browser and on the settings defined for it. These buttons are also problematic when navigating links that trigger side effects.

With RIA applications, this problem is even more pronounced as the user is often sent back to the initial state of the whole Web application when using the Back button instead of just moving back one interaction step in the current page. For example, the initial release of Gmail suffered from this problem as the

whole application was made available using one single URL¹. In the meantime, the problem has been recognised by both Web application providers as well as by browser developers. More recent releases of GMail, for instance, append unique message identifiers to the URL to improve the user experience, while browsers such as Microsoft Internet Explorer 8 provide embedded functionality to handle AJAX navigation². These efforts document that the problem of navigating complex Web applications is relevant, but there is not yet a generic and systematic way to address this problem based on standard Web engineering methods. Thus, several major applications do not address this issue at all and suffer from all the above-mentioned problems.

Therefore, we propose to evolve the interaction paradigm by moving the Web (and the supporting browsers) from the browsing paradigm based on *Pages*, with related *Back* and *Forward* actions, to a full-fledged interactive application paradigm, based on the concept of *State*, that features *Undo* and *Redo* capabilities, and transactional properties. This results in a safer interaction environment where users can navigate freely through Web applications and undo/redo their actions without experiencing unexpected application behaviour, including the side effects generated by the user interactions. Safety of the interfaces is therefore increased with respect to possible wrong or unexpected behaviours of the user, that is granted to always navigate among correct application states.

This paper is organised as follows. In Sect. 2, we continue to motivate the relevance of our work. Section 3 presents our modeling approach based on state machines and Sect. 4 presents the application programming interface designed to support the interface models and ensure its consistent behaviour. Section 5 discusses some methodological guidelines for adopting the approach and Sect. 6 describes our implementation experiences. Finally, we compare our approach to related works in Sect. 7 and conclude in Section 8.

2 Motivation

As the Web matures into an application platform, the concept of Web pages becomes less important, while the state of these applications gains in relevance.

In the page-base approach of traditional Web applications parameters are passed from one page to another using HTTP requests. Naturally, this can lead to problems when pages are visited in a different order or out of context. The state-based approach, on the other hand, represents the Web application as a finite state machine. The state of the page is then read from a global state repository when a page is accessed and stored back when the state is left (either to go to a different page or to move to a different state within the same page).

Currently, browsers are still designed to deal with page-based Web sites which causes the problems mentioned in the introduction. To illustrate these problems and motivate our approach we will use the GMail Web application as a running

¹ <http://mail.google.com/mail/?ui=1>

² <http://msdn.microsoft.com/en-us/library/cc288472.aspx>

example. GMail, as well as many other known Web sites, suffers from several interaction problems, due to the widespread adoption of AJAX.

For instance, GMail “fold” and “unfold” functionality in the message view to hide or expand conversations does not work together with the Back and Forward buttons, being implemented purely using JavaScript and CSS. Another chain of interactions that sometimes demonstrates the unexpected behaviour of the Back button is related to the login action: after logging in, users are presented with an overview of their inbox and can then select a message, which is then displayed. The expected outcome of using the Back button on this page would be to bring the user back to the inbox page. However, the use of the back button redirects the user back to the login page.

Additionally, some parts of the GMail interface should exhibit transactional properties in the sense that there is an “all or nothing” semantics over all states involved in the interaction. In particular, it should not be possible to step backwards through the states of a transaction once it has been completed. For instance, currently, if users display and then delete messages, they are redirected to the inbox and notified that the message has been moved to the trash. If, then, however, the user clicks the Back button, the deleted message reappears again, but the effect of deleting the message is not undone, i.e. the message remains deleted which confuses the user. We believe that in a state-based Web application, clicking the back button should rollback the whole delete transaction, redirecting the user to the state of the inbox previous to deleting the message and undoing the removal operation.

Many other famous Web sites suffer from the same problems: iGoogle.com, linkedin.com, maps.google.com, www.surveymonkey.com, www.hostelworld.com, www.lastminute.com, www.amazon.co.uk, www.gap.com, among many others, even including the official examples of AJAX and FLEX³. In general, almost no site (including the ones developed with pure server-side technology) is able to preserve the complete history of the navigation, that includes the input submitted by the user in the forms, which should be retrieved once the Back button is clicked. Other critical issues are not addressed at all by Web applications, including: rollback (or compensation) of side effects in correspondence to backward navigation and fine grained back and forward managements, e.g., at the level of fields in a form.

Existing Web design methods such as WebML [2], Hera [3], OOHDm [4], etc. already provide models intended for the specification of hypertext navigation. Even though some of them have been extended to support basic RIA features, they lack of mechanisms for exactly specifying fine-grained user interactions, since they operate at the granularity of pages and components. In this paper, we propose a modelling approach and run-time support system designed to allow the specification and implementation of safe user interactions in Web applications. The methodology is complementary and orthogonal to existing approaches in the sense that it can be used in addition to an existing language. Our approach is based on the shift of paradigm from page-based navigation to

³ e.g., <http://examples.adobe.com/flex3/devnet/dashboard/main.html>

state-based interaction. We apply the principle of separation of concerns, thus defining a specific model for interface navigation behaviour.

3 Modeling Safe Interfaces

This section introduces our modelling approach for the definition of the interface interaction aspects of Web applications. Our proposal is based on the state machines sublanguage of the UML [5] which we have adapted to the Web applications domain by adding concepts like *Page*, *GraphicalElement*, *Transaction* and so forth.

The abstract syntax of the language is partly described by the MOF-compliant metamodel depicted in Fig. 1. The language permits to define Web pages (*Page* metaclass) and decompose them in a set of interaction substates (*State* metaclass), where exceptional states (*ExceptionState* subclass) are useful to specify the default application behaviour in case of unexpected errors. State transitions (*Transition* metaclass) are triggered by events (class *TriggerEvent*) on the graphical elements of the interface (*GraphicalElement* metaclass). The possible triggering events are predefined in the enumeration *EventType* to facilitate their definition and a more homogeneous treatment. Transitions may involve the execution of a sequence of action instances (*Action* metaclass). Actions may alter the state of the graphical elements in the page or change the population/value of the Web application data. Links between pages can be inferred by detecting transitions between states in different pages.

As an example, Fig. 2(a) shows a subset of the GMail Web application interface definition, as commented before. The first page (shown using a *package* symbol) is the *Login page* and the second page is the *Mail page* that provides all functionality of the GMail client. The Login page has only one state, *Show Login* (drawn using a circle shape; a bold border denotes that this is the initial state for the page), that displays the input fields for user name and password (not shown in the figure). After the user submits this form, the Web application progresses to the *Show Inbox* state on the Mail page that displays a list of all mails. On the same page, selecting a message moves the application to the *Show Message* state where the selected message is displayed. In this state, the *next* and *previous* transitions allow the user to step through all messages currently in the inbox. This model makes it clear that clicking the Back button when in the state *ShowMessage*, users have to be moved back one state (to the *Show Inbox* or the same *ShowMessage* state, depending if they arrived to the current state via the *next*, the *previous* or the *show* transition) but never back to the previous page (as it happens when the modeling language does not allow designers to specify state-based interfaces).

Transitions may be part of a transaction. As usual, rolling back a transaction implies undoing all changes done since its beginning. That is why each action needs to provide not only the definition of the changes performed when executing the action (the *do* behaviour specification) but also the *undo* specification (if possible) that will be used if the transaction must be rolled back. For pre-

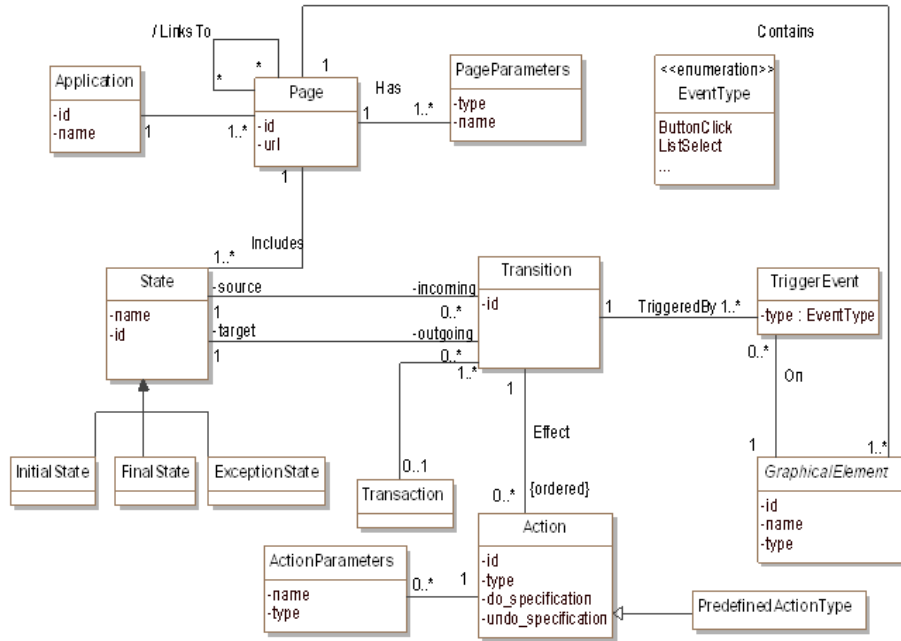


Fig. 1: Interaction metamodel

defined actions (e.g., create new object, update an attribute, enable a button, ...), the behavior of the *do* and *undo* operations may be skipped and a default behaviour depending on the action type can be predefined and used instead. For other actions this behaviour must be provided by the designer as part of the interface modelling process. Transactions can span over different pages and sets of side effects. An example for this notion of a transaction is the chain of states shown in Fig. 2(b) that deletes the currently displayed message. The transaction is denoted by the dashed box enclosing states *Show Message* and *Delete Message*. Defining this transaction avoids the unexpected behaviour described in the previous section.

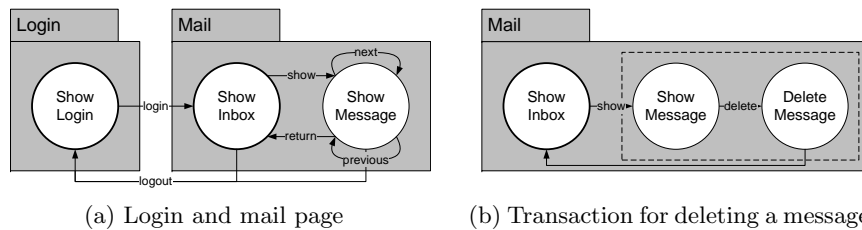


Fig. 2: State models for GMail

GraphicalElement is an abstract class that could be decomposed into a set of *Button*, *ComboBox*, *EditField*, ... subclasses (not shown in the figure). Depending on how we plan to use this interaction model (see Sect. 5) these classes can be directly taken from the metamodel of the Web modeling language we integrate our approach with.

Additionally, we have defined several well-formedness rules (WFRs) that enforce a consistent relationship between the different components of the interface when specified using our metamodel. Examples of well-formedness rules are: all pages must have at least one initial state, the type of trigger event must be compatible with the kind of graphical element associated to the event or that transitions in a transaction must represent a consecutive sequence of steps in the interaction. These WFRs can be formally expressed using a textual language like OCL.

4 Tracking User Interaction at Run-time

Once the interface has been modeled, we need to ensure that its implementation provides the expected behaviour at run-time. To achieve this, we need two types of knowledge: (1) the static information defined by the designer in the previous interface model; and (2) the execution trace of all events, state visits, page access and actions executed so far (e.g., to be able to retrieve the correct state of the interface and application data state when clicking the back button).

The data structures required for the latter aspect are shown in the class diagram of Fig. 3. Each different execution of the Web application is recorded in the *ApplicationExecution* class. In each execution, we record all visits to the states defined in the interface model. For each visit, we record the transitions that lead/exited to/from the visit and the event that triggered those transitions. Even more important, we record all actions executed during the process including all the arguments used when executing those actions and the current values of all graphical elements at that point (e.g. items selected in each combobox, state of the checkboxes, ...). Every time we visit a state in a different page, we also record the page access plus the parameters used when loading the page.

By recording all these pieces of information, we are able to recreate the complete state of the application at any previous point of time, and thus, ensure a safe interaction behaviour. This information is managed through an API we have predefined to ease the development of Web applications following our proposal. A subset of the API is described in Tab. 1. For each method we describe the class where the method is attached⁴, its input and output parameters and a short description of its semantics. Methods `getNext` and `getPrevious` can be used by the application developer to query the next or previous visit in the history, respectively. Instead, the `do` and `undo` method are then used to actually perform a move to the next or previous visit and, thus, they manipulate the history records during the process. The method *redo* re-visits a state that has already

⁴ We could also add all methods to a single class, following the *Facade* pattern

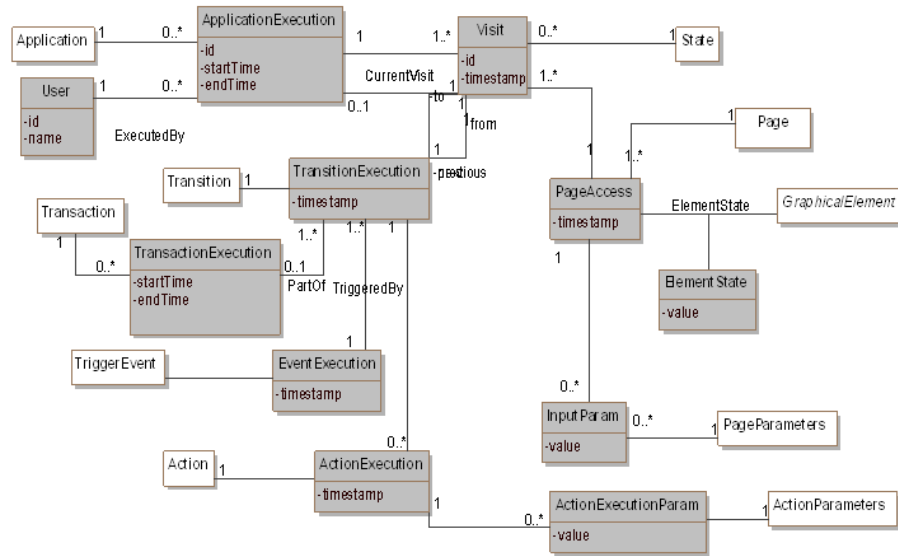


Fig. 3: Class diagram to track the user interaction

been visited. As the event and parameters to move to the new state have already been recorded by our framework in this case, they do not have to be provided once more. Notice that the navigation trace recording is not expected to reduce significantly the application performances, since other application parts (e.g., AJAX interfaces) are more likely to become bottlenecks. The complexity of the stored data is in line with the standard logging information of Web applications.

To show how the API can be used to enforce safe interaction semantics, we show a sequence diagram that sketches the operation sequence executed in response to a user click on the Back button in Fig. 4. Once the *ApplicationExecution* instance receives this event (intercepted by means of JavaScript code in the client browser), it accesses the visit previous to the current one, which is where the user must be redirected. The redirection is performed by cloning the previous visit (i.e., creating a new one, with identical properties) instead of just pointing directly to it. This allows the correct behaviour of the undo/redo mechanism also in the case of partially rolled back history, where the user goes back and forth several times over the same interaction sequence. This clone visit is inserted in the history of the navigation, for tracking purposes, with the following values: (1) its next visit is the current visit at the beginning of the execution of the back operation and (2) the previous visit is the same as the one of the visit we have cloned. That is, a redo operation would move the user back to the initial visit state. An additional Back operation would move another step back in the sequence of visits. At the end of the operation, this cloned visit is returned as the new current one and can be used by the Web server to retrieve all the needed

Table 1: API Methods (Excerpt)

Method	Remarks
<code>State::getNextState(Event e): State</code>	informs about the next state to go based on the current one and the given event
<code>Visit::getNext(): Visit</code>	queries next visit
<code>Visit::getPrevious(): Visit</code>	queries previous visit
<code>ApplicationExecution::do(EventExec e, Parameter[] p): Visit</code>	moves to next visit
<code>ApplicationExecution::redo(): Visit</code>	moves to the (previously visited) next visit
<code>ApplicationExecution::undo(): Visit</code>	moves to previous visit reversing all executed actions
<code>Visit::clone(): Visit</code>	creates a clone of the visit
<code>ActionExecution::do(ActionExecutionParam[] params)</code>	executes the action
<code>ActionExecution::undo(ActionExecutionParam[] params)</code>	undoes the effect of the action
<code>TransitionExecution::undo()</code>	undoes all actions associated to the transition
<code>TransactionExecution::rollback()</code>	rollbacks the transaction

information (including the related page access and parameters) to compute the Web page.

The procedure for rolling back a transaction would be similar, with an initial iteration to undo all visits until the last visit before starting the transaction.

5 Methodology

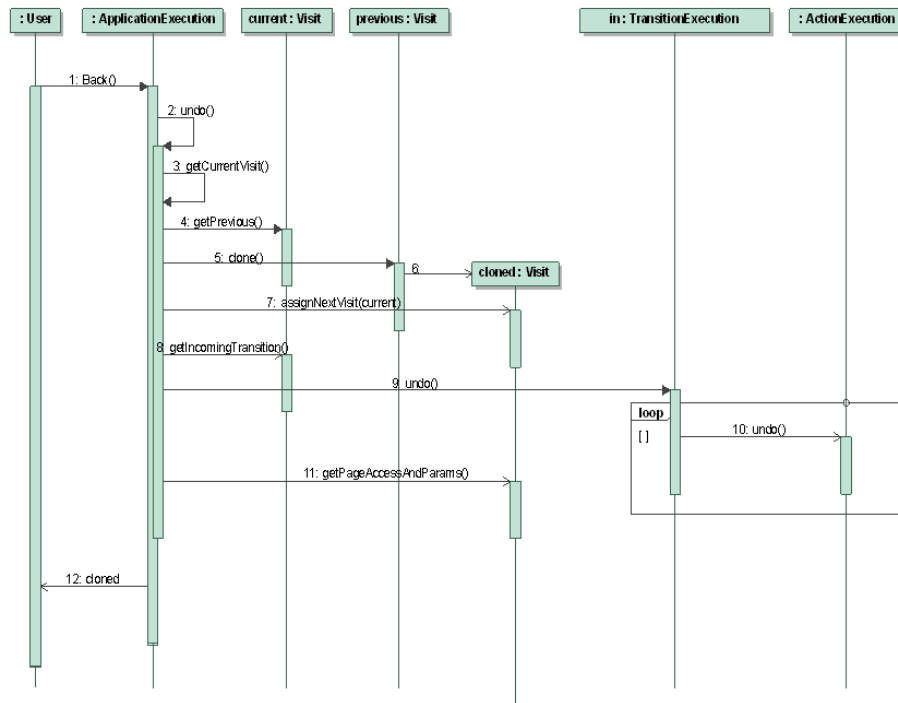
This section proposes methodology guidelines for adopting our approach, possibly combined with other design methods. Indeed, our method can be used both (1) together with an existing Web engineering method and (2) as a stand-alone approach that directly exploits the API just focusing on the design of the state behaviour of the application. In the following, we provide an indication of how to use our method in the two aforementioned scenarios.

5.1 Integration with WebML

Due to the orthogonality of the issues addressed by our approach with respect to existing Web engineering methods, it is straightforward to blend it within an existing design methodology for Web applications.

The joint use of our method with existing approaches requires to identify:

- the set of primitives that overlap with the existing modelling language (and therefore that can be directly mapped one onto the other);
- the set of primitives that are not defined nor covered in the existing modelling language (and therefore need to be introduced);

Fig. 4: Sequence diagram for the *Back* operation

- and the set of primitives that conflict or partially intersect the semantics of some existing primitives (and therefore require to resolve the conflict).

To demonstrate the feasibility and neutrality of the approach with respect to the modelling methodology of choice, we show how the approach can be used together with WebML (Web Modeling Language) [2], a methodology for designing Web applications. The choice of WebML is due to its widespread adoption, the availability of a MDE tool-suite (WebRatio⁵), and our knowledge of its basic usage and semantics.

The combination of our approach with WebML and WebRatio has two main advantages: the possibility of extending a well established code generation framework for covering the new features (i.e., through invocations to our API); and the availability of a huge set of real industrial application models that can be exploited for validating our approach.

Given a multi-model design approach such as WebML, two ways can be followed for merging the two methods: (1) defining a new modeling view of the application, that orthogonally describes the state modeling as a separate aspect of the application; (2) blending the state modeling concepts within one of the existing types of models provided by the methodology. The latter is convenient

⁵ <http://www.webratio.com>

when there is an overlap some modelling primitives of the methodology already include concepts that overlap with our method.

WebML is a good example of integration according to the approach (2), since the hypertext model includes examples for all the three categories of primitives mentioned above:

- the *page* concept already exists in WebML and perfectly maps to the new *page* concept;
- the *state* and *transaction* concepts do not exist in WebML⁶, and therefore need to be explicitly introduced in the notation;
- finally, some concepts actually create some conflict with the existing models. Such conflicts could be due to the semantics of the concept or to its granularity. For instance, the *transition* concept partially overlaps with the WebML link concept, and therefore needs to be reconciled. On the other side, some granularity conflicts arise because some WebML features are more coarse-grained than we expect in our model: e.g., primitives such as *Entry units* encapsulate the whole behaviour of a form; primitives such as *Landmarks* represent sets of links coming into a page.

To clarify how these issues are solved in concrete cases, we exemplify in Fig. 5 a simple visual integration of a WebML hypertext model with the concepts of our proposal. The picture shows in black thin lines the native WebML concepts (pages, units, operations, and links) of a hypertext model describing a simplified email management interface: a *Home page* contains a form for the *Login*, which triggers the login server-side action and then redirects to the *Email page*. There, the user is shown a hierarchy of email *Folders*, that can be browsed. Once a folder is selected, the list of contained *Emails* is shown, and a specific *Message* can be chosen for looking at its details. Then the user can either delete the current message or reply to it. For replying, they can click on a link that leads to the *Send Message* page, which includes the *New Msg* form. Once the message is submitted, its data is recorded and the message is sent.

This basic WebML model, however, does not cover safe state management at all. Therefore, we extend it with our approach, by adding State, Transition, and Transaction primitives on top the hypertext model. States (represented by gray rounded boxes) are added as an orthogonal dimension over sets of WebML units. Notice that the distribution of units within the states is arbitrary, according to the logic that the designer wants to convey. Transitions (represented as thick arrows) basically map to WebML links, when they connect one state to another. Transactions (shown as dot-dashed boxes) surround set of states. WebML server-side operations are regarded as *Actions* in our approach, and thus are associated to the related transitions.

In the example, state *S1* is associated to the completion of the login form and to the click on the submit button. The exiting transition *T1* leads to state *S2*

⁶ A transaction concept actually exist, but it represents an atomic set of server-side actions to be performed altogether within a single server request; therefore, it does not collide with our concept. For clarity, we will refer to that concept as “server-side transaction”.

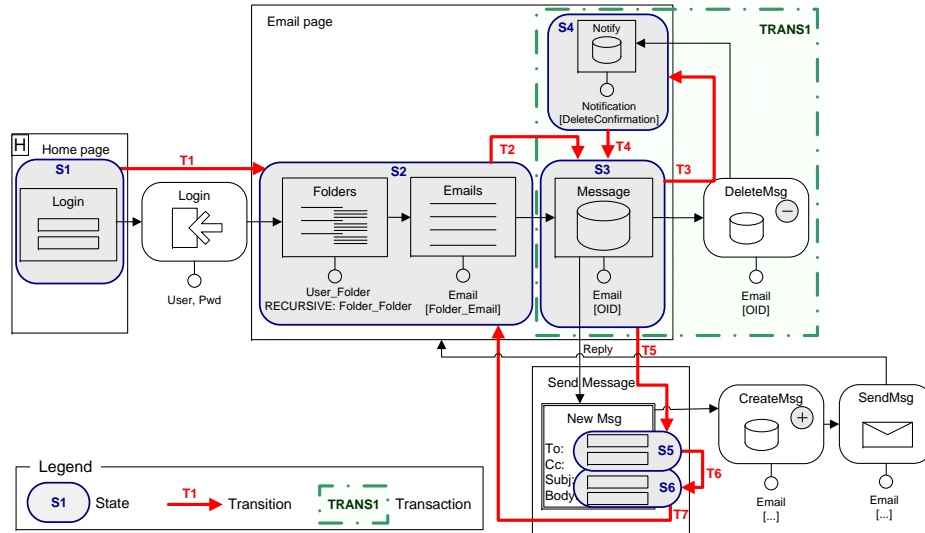


Fig. 5: Example of WebML hypertext model integrated with state awareness.

and comprises the action *Login*. Analogous definitions are assumed for the other states. Notice that states can be defined over set of units (e.g., $S2$), but also on sub-units (e.g., groups of fields in a form, such as $S5$ and $S6$). This allows undo and redo actions to be performed at different granularity levels. Due to transactions (e.g., $TRANS1$), safe interaction is granted also upon deletion or updates of contents.

Notice that one aspect that cannot be graphically addressed in this representation is the definition of the reverse actions for allowing rollbacks. This must be defined separately for non-predefined actions. For predefined ones, the reverse procedure is already implicit in the semantics of the action definition and thus does not need to be explicitly defined by the designer again.

5.2 Standalone approach

Another possible usage of our approach consists of directly exploiting our API for the design and implementation of Web applications. This approach is suitable for traditional developers that are not familiar with model-driven design. Therefore, they are familiar with using existing APIs for getting immediate benefits within the developed application. The most they are willing to do is adopting a very simple visual notation to summarize the general application structure, which can be achieved through the simple notation we have proposed so far for modeling pages, states, transitions, and transactions.

Through that notation, it is possible to describe the aspects covered by the approach and to use it to guide the implementation phase. This can be done through a set of implementation guidelines that rely on the adoption of the API described in Tab. 1, as roughly summarized here.

- *basic events* (i.e., user clicks) can be captured by the default behaviour of the browser when managing hypertext links.
- *advanced events* (e.g., clicks on the back-forward button, right clicks, drag and drop, and so on) can be captured by JavaScript embedded within the pages, e.g., by adopting an AJAX library.
- *change of state* can be implemented by embedding the appropriate calls to the methods that register the change of state, in the server-side actions. Such actions must be invoked by *any* state-relevant hyperlink (in case of simple events) or by any asynchronous invocation (in case of AJAX events).
- *undo and redo* actions can be implemented through appropriate links/buttons on the page, that invoke the proper actions at server side; or through capturing the back-forward events as mentioned above and redirecting them to the same actions.

Therefore, once the API plus the set of guidelines—possibly more precisely specified and illustrated with simple coding examples—are provided to the developers, they are able to complete the development by quickly implementing the state-specific aspects of the application.

6 Implementation Experience

This section aims at briefly reporting our experience in implementing the proposed approach. The setting where our approach has been tested is within the J2EE framework. The (partially) presented API in Sect. 4 has been implemented as a set of Java classes and respective methods. The API has been tested in the context of a few Web applications, that have been implemented through a set of JSP pages and Servlets that coherently invoke the API. Management of special events has been delegated to the AJAX Prototype (and Scriptaculous)⁷ libraries. The server-side architecture is based on Struts. However, thanks to the independence from the technology, our approach can obviously be implemented in any other environment.

7 Related Work

Most traditional methods for user interface design use state machines (in different flavours) as the underlying formalism. An early approach proposing state-based definition of general user interfaces is Jacob [6]. Later, similar techniques were adopted by the Hypermedia community to specify navigation [7] and by the Web Engineering community to express the structure and behaviour of Web sites. For example, Leung et al. [8] were among the first to propose the use of statecharts to address the growing complexity of dynamic Web sites. Later, StateWebCharts [9] were proposed, a refinement of previous approaches that extend statecharts with more concepts targeted towards the modelling of Web

⁷ <http://www.prototypejs.org/>, <http://script.aculo.us/>

applications. Finally, Draheim and Weber [10] propose the use of bipartite state machines to model both the pages and the server actions on them.

In the domain of model-driven Web engineering, the necessity to support the specification of RIAs was recognised early on [11]. Proposals in this direction include an extension to WebML [12] and the RUX-Model [13], ADV-Charts [14], and an orchestration model for widgets [15]. Nevertheless, most existing Web design methodologies such as WebML [2], Hera [3], OOHDM [4], etc. do not provide concepts to precisely specify complex application states. While these approaches support the specification of the desired interface behaviour, they do not consider the problems caused by the user-interaction with the Web browser (such as the *Amazon bug*) nor do they provide any kind of transactional support for interface interactions.

Currently, the only possibility that designers have is to identify the potential interaction problems using, for example, an existing verification/validation technique [16] and, then, correct these issues during the implementation phase. The particular issues arising from using the browser's Back button in modern Web applications, however, have been addressed by a number of approaches. For example, Milic et al. [1] study the different problems that can occur in the context of back navigation and have proposed a specific solution called Smartback. Alternative approaches have been proposed [17–19]. To address the problems identified by these authors in a generic and model-based design method is precisely the goal of our method.

8 Conclusions

This paper presented a new method for modelling and implementing safe interface interactions in Web applications that includes transactional properties and full-fledged undo and redo capabilities. The method is based on decomposing pages in a set of states linked by transitions executed in response to user events. An API allows to record the complete trace of the interactions, thus allowing consistent back and forward navigation, including the possibility of rolling back a sequence of interaction steps defined as a transaction at the model-level. Our approach improves the browsing user experience within complex business applications and makes the behaviour deterministic regardless of the browser and the implementation technologies.

As future work we plan to extend the code generator of WebML/WebRatio for supporting the new features of our approach (i.e., the API invocations) and validate the scalability and completeness of such generator and of our new primitives upon several existing WebML models of real industrial applications. Finally, we aim at developing a simple code generation prototype that transforms our standalone models to skeletons of Java and JSP code that invoke our API.

Acknowledgements: Work supported by the project TIN2008-00444 from the Spanish Ministry of Education and Science, by the 2007 BP-A 00128 grant from the Catalan Government, and by grant PBEZ2-121230 from the Swiss Science Foundation (SNF).

References

1. Milic-Frayling, N., Jones, R., Rodden, K., Smyth, G., Blackwell, A., Sommerer, R.: Smartback: Supporting Users in Back Navigation. In: Proc. WWW'04. (2004) 63–71
2. Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., Matera, M.: Designing Data-Intensive Web Applications. The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann Publishers Inc. (2002)
3. Vdovják, R., Fräsincár, F., Houben, G.J., Barna, P.: Engineering Semantic Web Information Systems in Hera. *Journal of Web Engineering* **1**(1-2) (2003) 3–26
4. Schwabe, D., Rossi, G., Barbosa, S.D.J.: Systematic Hypermedia Application Design with OOHD. In: Proc. Hypertext'96. (1996) 116–128
5. Object Management Group: UML 2.0 Superstructure Specification. (2004)
6. Jacob, R.J.K.: A Specification Language for Direct-Manipulation User Interfaces. *ACM Trans. Graph.* **5**(4) (1986) 283–317
7. de Oliveira, M.C.F., Turine, M.A.S., Masiero, P.C.: A Statechart-based Model for Hypermedia Applications. *ACM Trans. Inf. Syst.* **19**(1) (2001) 28–52
8. Leung, K.R., Hui, L.C., Hui, S., Tang, R.W.: Modeling Navigation by Statechart. In: Proc. COMPSAC'00. (2000) 41–47
9. Winckler, M., Palanque, P.: StateWebCharts: A Formal Description Technique Dedicated to Navigation Modelling of Web Applications. In: Proc. Intl. Workshop on Design, Specification and Verification of Interactive Systems. (2003) 279–288
10. Draheim, D., Weber, G.: Modelling Form-based Interfaces with Bipartite State Machines. *Interacting with Computers* **17**(2) (2005) 207–228
11. Preciado, J.C., Linaje, M., Sánchez, F., Comai, S.: Necessity of Methodologies to Model Rich Internet Applications. In: Proceedings of International Symposium on Web Site Evolution, September 26, 2005, Budapest, Hungary. (2005) 7–13
12. Bozzon, A., Comai, S., Fraternali, P., Toffetti Carughi, G.: Conceptual Modeling and Code Generation for Rich Internet Applications. In: Proceedings of International Conference on Web Engineering, July 10-14, 2006, Menlo Park, CA, USA. (2006) 353–360
13. Linaje, M., Preciado, J.C., Sánchez-Figueroa, F.: A Method for Model Based Design of Rich Internet Application Interactive User Interfaces. In: Proceedings of International Conference on Web Engineering, July 16-20, 2007, Como, Italy. (2007) 226–241
14. Urbieto, M., Rossi, G., Ginzburg, J., Schwabe, D.: Designing the Interface of Rich Internet Applications. In: Proc. LA-WEB'07. (2007) 144–153
15. Pérez, S., Díaz, O., Meliá, S., Gómez, J.: Facing Interaction-Rich RIAs: The Orchestration Model. In: Proc. ICWE'08. (2008) 24–37
16. Alalfi, M.H., Cordy, J.R., Dean, T.R.: A Survey of Analysis Models and Methods in Website Verification and Testing. In: Proc. ICWE'07. (2007) 306–311
17. Biel, B., Book, M., Gruhn, V., Peters, D., Schäfer, C.: Handling Backtracking in Web Applications. In: Proc. EUROMICRO'04. (2004) 388–395
18. Ceri, S., Daniel, F., Matera, M., Rizzo, F.: Extended Memory (xMem) of Web Interactions. In: Proc. ICWE'06. (2006) 177–184
19. Baresi, L., Denaro, G., Mainetti, L., Paolini, P.: Assertions to Better Specify the Amazon Bug. In: Proc. SEKE'02. (2002) 585–592