Tools for Modeling and Generating Safe Interface Interactions in Web Applications

Marco Brambilla[1], Jordi Cabot[2], and Michael Grossniklaus[1]

[1] Politecnico di Milano, Dipartimento di Elettronica e Informazione
P.za L. Da Vinci, 32. I-20133 Milano - Italy
{mbrambil,grossniklaus}@polimi.it
[2] INRIA - École des Mines de Nantes
Rue Alfred Kastler, 4 B.P. 20722 - F-44307 NANTES Cedex 3 - France
jordi.cabot@inria.fr

Abstract. Modern Web applications that embed sophisticated user interfaces and business logic have rendered the original interaction paradigm of the Web obsolete. In previous work, we have advocated a paradigm shift from static content pages that are browsed by hyperlinks to a state-based model where back and forward navigation is replaced by a full-fledged interactive application paradigm, featuring undo and redo capabilities, with support for exception management policies and transactional properties. In this demonstration, we present an editor and code generator designed to build applications based on our approach.

1 Introduction

The Web has evolved from a platform for navigating hypertext documents to a platform for implementing complex business applications, where user interaction relies on richer interaction paradigms (RIA, AJAX). In this context, the original interaction paradigm of the Web, based on a simple navigation approach of moving from one page to another is too simplistic. 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]. Depending on the bowser and the application, problems with the use of back and forward buttons include loss of data in pages with form fields, resetting the state of AJAX applications or repeatedly triggering a side effect of a link, e.g., the *Amazon bug*. The behaviour after exceptions and errors (e.g., session timeout) is also indeterministic.

These issues complicate the modelling of complex Web applications and hamper the user experience. State-based models are well suited for the specification of user interfaces and applications [2]. In previous work [3], we have therefore proposed a state-based modelling language to specify safe user interactions for Web applications, that is complementary to existing Web design methodologies, e.g., [4,5,6]. Our approach evolves the interaction paradigm by moving the Web from the browsing paradigm based on *Pages*, with related *Back* and *Forward* actions, to a full-fledged interactive application paradigm, based on the concept

B. Benatallah et al. (Eds.): ICWE 2010, LNCS 6189, pp. 482–485, 2010.

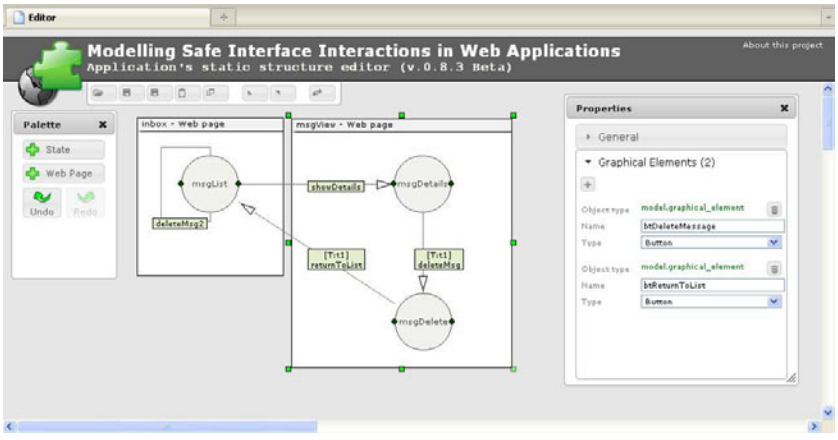

Fig. 1. Example Web interface model drawn with the online editor

of *State*, that features controlled *Undo* and *Redo* capabilities, exception management, and transactional properties.

This paper presents a toolset consisting in (1) a model editor to specify Web application interfaces, (2) an API that grants access to the model concepts, and (3) a code generator that automatically produces prototypical applications from the models that exploit our API at runtime for granting safe navigation.

2 Modelling Safe Interfaces for Web Applications

The first step of the development process is the specification of the interface and behaviour of the Web application. In our proposal Web applications are represented as state machines consisting of *states* (i.e., possible situations the application can be in) and *transitions* (i.e., changes from a state to another, triggered by an event). A single Web page can comprise several states, depending on the granularity chosen by the designer. Additional modeling primitives allow the definition of exception events and states (that model the response to unexpected situations) and the definition of *transaction* regions, i.e., a set of states that must be accomplished with all-or-nothing semantics. We also offer a set of predefined kinds of transitions between states (e.g., click button, list selection, ...) to facilitate the definition of the state machine.

As an example, Fig. 1 shows our model editor[1] at work, depicting a model for a Web email application. Page *inbox* shows an index of all available messages, that can be deleted or selected for visualization. The *msgView* page shows the details of the selected message, which can be deleted. The *deleteMsg* and *returnToList* transitions belong to the same *transaction T1*. If users undo the deletion, they are actually sent back to the previously deleted message, which is also restored (through a rollback operation) in the application state.

[1] Available at `http://home.dei.polimi.it/mbrambil/safeinterfaces.html` (beta version)

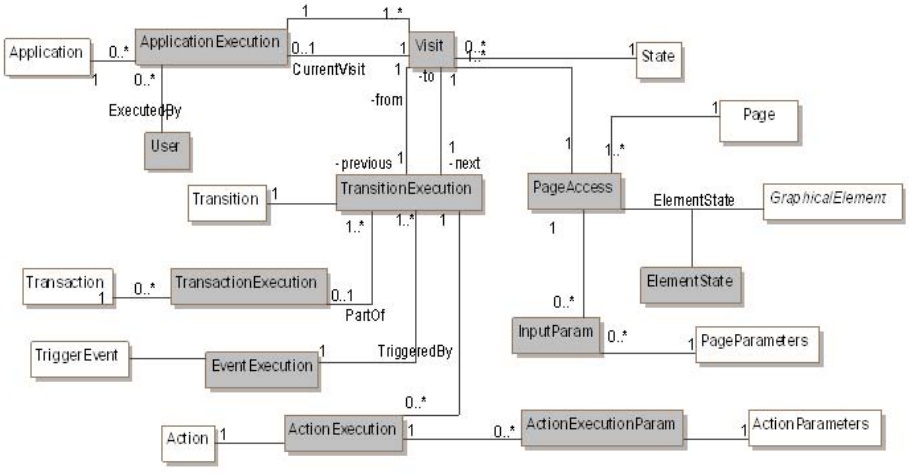

Fig. 2. Interaction metamodel

The online editor has been implemented as a Rich Internet Application, exploiting the OpenJacob Draw2D and Yahoo! User Interface libraries. The editor allows to save, load, edit, and validate models, and provides automatic generation of running prototypes from the models.

The modeling language used to describe the state machines is defined by the internal metamodel shown in Fig. 2 (white classes). Our metamodel is based on the the state machines sublanguage of the UML, adapted to the Web applications domain by adding concepts like Page, GraphicalElement, Transaction and so forth (as described above).

3 Run-Time Support for Safe Web Interactions

Our tool also helps to implement the modeled Web application by (1) automatically parsing the model information and passing it on to a predefined server component that acts as a controller for the application (MVC architecture) and (2) providing a run-time API that programmers can use to interact with the controller and easily manage all events involving state changes in the application and implement correct state behaviours (including undo and redo features) with little effort.

The data structures used internally by the controller to manage run-time dynamic information (current state the user is in, input parameters, user events) are shown in Fig. 2 (grey classes). For instance, every move of the application user to a state is recorded as a new *Visit*. Obviously, the same user can visit the same state several times. The visits trace is permanently stored to allow undo/redo computation.

Both the static and run-time information of the application can be accessed/updated using our API. For reasons of space, we only present the main API functions. For instance, methods `getNext` and `getPrevious` can be used by

Table 1. API Methods (Excerpt)

Method	Remarks
`Visit::getNext(): Visit`	queries next visit
`Visit::getPrevious(): Visit`	queries previous visit
`ApplicationExecution::do (EventExec e, Parameter[] p): Visit`	moves to next visit and performs the corresponding actions
`ApplicationExecution::redo():Visit`	moves to the (previously visited) next visit
`ApplicationExecution::undo():Visit`	undoes the last transition and actions
`TransactionExecution::rollback()`	rollbacks the transaction

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. Note that the `do` method uses the parsed information from the state machine to know the state to go to according to the current user state and the event triggered by the user. The method `redo` re-visits a state that has already been visited.

4 Conclusion

We have sketched an approach for modeling Web application interfaces using extended state machines. A run-time API supports the implementation of applications and ensures safe and deterministic application behaviour even in the case of exceptions. As future work, we plan to validate the solution in industrial case studies and provide full coverage of transactionality of side effects.

References

1. Baresi, L., Denaro, G., Mainetti, L., Paolini, P.: Assertions to Better Specify the Amazon Bug. In: Proc. SEKE '02, pp. 585–592 (2002)
2. Draheim, D., Weber, G.: Modelling Form-based Interfaces with Bipartite State Machines. Interacting with Computers 17(2), 207–228 (2005)
3. Brambilla, M., Cabot, J., Grossniklaus, M.: Modelling Safe Interface Interactions in Web Applications. In: Proc. ER 2009, pp. 387–400 (2009)
4. Schwabe, D., Rossi, G., Barbosa, S.D.J.: Systematic Hypermedia Application Design with OOHDM. In: Proc. Hypertext '96, pp. 116–128 (1996)
5. Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., Matera, M.: Designing Data-Intensive Web Applications. Morgan Kaufmann, San Francisco (2002)
6. Vdovják, R., Frăsincar, F., Houben, G.J., Barna, P.: Engineering Semantic Web Information Systems in Hera. Journal of Web Engineering 1(1-2), 3–26 (2003)