Modelling and Verification of Web Navigation *

Zuohua Ding[1], Mingyue Jiang[1], Geguang Pu[2], and Jeff W. Sanders[3]

[1] Center of Math Computing and Software Engineering, Zhejiang Sci-Tech University
Hangzhou, Zhejiang, 310018, P.R. China
zouhuading@hotmail.com, jiang_my@126.com
[2] Software Engineering Institute, East China Normal University
Shanghai, 200062, P.R. China
ggpu@sei.ecnu.edu.cn
[3] International Institute for Software Technology
Unite Nations University, P.O. Box 3058, Macao
jeff@iist.unu.edu

Abstract. Web navigation model provides a dynamic view for web modelling. It is useful for clarifying requirements and specifying implementation behaviors of systems from design intensions. In this paper, we propose a formal model to describe web navigation of user behaviors, where link activities play an important role. Several issues have been considered in our model, such as web browser effects, adaptive navigation, frame communication etc. After the link activity model is established, we use model checker SPIN to check whether there exist problems such as such as broken links, dead ends, missed reply pages, reachability of pages etc. This method can help us to analyze user behaviors, meanwhile it provides us a way to expose design faults in web systems.

Keywords: Web Modelling, Link Analysis, Requirement Engineering, Model Checking.

1 Introduction

Recently, various web modelling approaches are proposed to model web applications. There are two main different concerns for conceptual modelling of web applications: information modelling and navigation modelling. Information modelling describes the contents of web applications and it is considered as from static point of view such as [1]. Navigation modelling pictures the navigation capabilities, i.e., the paths on which users can traverse to explore the information required, and it is considered as from dynamic point of view. Examples can be found in [9]. We regard these two approaches are complement to each other. This paper is concerned with navigation modelling of web applications.

As the work [6] indicated, navigation models are useful for clarifying requirements and specifying implementation behaviors of systems. There are a few web navigation models proposed [1,9,11,2,5,3], such as the models based on extended

* Zuohua Ding is supported by NNSFC (No.90818013). Geguang Pu is partially supported by NNSFC (No.60603033) and Qimingxing Project (No. 07QA14020).

M. Gaedke, M. Grossniklaus, and O. Díaz (Eds.): ICWE 2009, LNCS 5648, pp. 181–188, 2009.

UML notation [8], Statecharts [6], navigation map [4] [10], etc. While taking the advantages of these models, we still need to consider some issues which may not be considered by the existing navigation modelling or in the information modelling in the web design phase:

- Web browser effects. In addition to the navigation provided by hyperlinks, web browsers can provide additional navigation functions that is out of control of the web pages, such as scrolling, back button, forward buttons and a history list. Static design model cannot include these navigation information.
- Adaptive navigation. In this situation, the next page also depends on the user's mode, for example whether he is a customer or an administrator, or depends on what pages the user has visited previously.
- Frame communication. For example, synchronization between continuous and static documents (i.e., scenes and pages). Since in principle pages are not ordered, their ordering comes as a result of scene ordering and scene-page relationships.

Generally speaking, the navigation of a web application is the possible sequences of web pages a user can visit. To handle the above issues, we present a formal dynamic behavior model for specifying the behavior of the navigation by *link point* activities. Through our approach, the navigation model is defined rigorously. The developer of the web application can better understand the application requirements and the implementation behavior with this proposed model and thus can develop web applications in a rigorous way.

2 A Formal Navigation Model

In the navigation, users navigate from a web page to the other by clicking on a hyperlink. A hyperlink can be in the form of text strings, graphcs or video, activated explicitly by users, using a mouse click for example. Hyperlinks can also be included in client side program/scripts (such as Javascripts, VBScript, Java and ActiveX), to be invoked automatically by the browser on some predefined events. Examples of these events include timeout, mouse movements and window focus.

The place where the user clicks the hyperlink is called link point. Here we give a formal definition for link point.

Definition 1 (Link Point). *A link point lp is a tuple defined as follows:*

$$(currentLink,\ nextLink,\ linkState,\\ variable\_set,\ action\_name,\ duration\_time)$$

where

- *currentLink(cl) is a string indicating the URL that the current page is located.*
- *nextLink(nl) is a string indicating the URL that the next page is located.*
- *linkState(ls) indicates that the link is statically or dynamically defined, enabled or disabled.*

- *variable_set(vs) indicates that after clicking the hyperlink, these variables information will be brought to the next page, such as login user data.*
- *action_name(an) denotes the action name with this link, which is mostly specified by web designers.*
- *duration_time(dt) denotes the interaction time between web servers and users after the link is clicked, which is actually a performance property decided by the deployment environment.*

Based on the states of link point, we can classify the link points as Type I, Type II and Type III. Type I link point must have new user input data which will be passed to the next page. For example, when a user clicks login button, he needs to input login name and password. Login button is the link point and some user id will be generated (maybe login name) for the next page. Another example is the search button. Key words are the values for the search button. Key words will be passed to the next page. Type II link point does not need any user input, but it will bring user information to the next page implicitly. For example, after a user logins to an online shopping web site, user id will be passed from one page to another page when the current user surfs for products. Type III link point only breaks down the user session. For instance, if a user clicks a logout link, the session built up earlier is closed and the user data is removed. Type I link point is denoted by $lp\bullet$, Type II link point is denoted by $lp\circ$, and Type III link point is denoted as $lp\times$.

2.1 Syntax

To specify the behaviors of users, we attempt to design an activity calculus to describe the web navigation behaviors performed by users. Because link point clicking reflects the intended behaviors of users, we design this activity calculus based on link points. It is easy to see that the sequence of link points is the same as the navigation path. The syntax of the activity calculus is defined as follows.

$$
\begin{array}{rll}
NE ::= & p \bullet & \text{(Type I)} \\
| & p \circ & \text{(Type II)} \\
| & p \times & \text{(Type III)} \\
| & \mathsf{skip} & \text{(Skip)} \\
| & \mathcal{B}p & \text{(back action)} \\
| & \mathcal{F}p & \text{(forward action)} \\
| & \mathcal{R}p & \text{(refresh action)} \\
| & NE \ ; \ NE & \text{(Sequence)} \\
| & NE \triangleleft \triangleright NE & \text{(nondeterminism)} \\
| & NE \triangleleft b \triangleright NE & \text{(Conditional)} \\
| & b * NE & \text{(Loop)} \\
| & NE \parallel NE & \text{(Parallel)}
\end{array}
$$

Without confusion, we use the same notations $p \bullet$, $p \circ$ and $p \times$ for both link points and link activities here. Activity skip does nothing but terminates. $\mathcal{B}p$ is

back operation which will clear all the user inputs on the page p. $\mathcal{F}p$ is forward operation which will not generate any user data, or bring any data to page p. $\mathcal{R}p$ is refresh operation which may change some link state from disable one to enabled one on the page p.

Action $NE \triangleleft\triangleright NE$ means that the user can randomly choose the link points. Action $NE \parallel NE$ denotes two link points may be performed in parallel. Note that $b * NE$ denotes loop activity, where b represents the boolean expressions, which depends on the variable values of link points or the history of variable values. Loop activity can specify the repeated actions performed by users.

2.2 Operational Semantics

The operational semantics of the link activities is presented in this subsection. We use the classical Labelled Transition System (LTS) to define an operational semantics, and the small-step operational semantics is adopted as well. The transition label a can be the link activity or the internal action τ.

The configuration of the transition system is designed as $\langle NE, \mathcal{E}, ST, LR, CL\rangle$, where NE is the link activity; $\mathcal{E}$ is the global activity trace in which each element is a link activity, and for instance, trace $< a, b, c >$ denotes that the user first performs link a and link b sequently, and then performs links c afterwards. The trace can help the user keep the visiting history list; ST represents the global state of link activities, which is a function of link variables; LR records the set of relations among link activities, which will be increased gradually based on the link activities performed by users; and CL stands for the current link activities.

Trace $\mathcal{E}$ has two types: one is for security and is bounded, denoted as $\mathcal{E}_s$; the other is for doing things, and is unbounded, denoted as $\mathcal{E}_t$. We use $\mathcal{E}^\frown a$ to represent the resulted trace after adding activity a to the end of $\mathcal{E}$, $\mathcal{E} \setminus a$ to represent the resulted trace after removing the top sequence composed by activity a in $\mathcal{E}$, and $a \in \mathcal{E}$ to represent that the activity a is contained in the trace $\mathcal{E}$.

The transitions of Type I and Type II activities are the following. The activating of Type II link will add the activity $\circ$ to the queue $\mathcal{E}$, while the activating of Type I link will add the activity $\bullet$ to the global queue $\mathcal{E}$.

$$\frac{}{\langle p\circ, \mathcal{E}, ST, LR, CL\rangle \xrightarrow{\circ} \langle skip,\ \mathcal{E}^\frown\circ, ST, LR \cup (CL, p\circ), p\circ\rangle}$$

$$\frac{}{\langle p\bullet, \mathcal{E}, ST, LR, CL\rangle \xrightarrow{\bullet} \langle skip,\ \mathcal{E}^\frown\bullet, \mathsf{ud}(ST), LR \cup (CL, p\bullet), p\bullet\rangle}$$

where predicate ud updates the global state based on link variables.

If Type III activity is performed, then the top sequence of activities $\bullet$ will be removed from the global trace. For example, in the queue, from the top we have $\bullet\bullet\circ\circ\circ\ldots$, then $\bullet\bullet$ will be removed.

$$\frac{}{\langle p\times, \mathcal{E}, ST, LR, CL\rangle \xrightarrow{\tau} \langle skip,\ \mathcal{E} \setminus \bullet, \mathsf{ze}(ST), LR \cup (CL, p\times), p\times\rangle}$$

where symbol $\mathcal{E} \setminus a$ represents the resulted trace after removing the top sequence of activity a in $\mathcal{E}$; and predicate ze sets the link variables to null.

The parallel expression has two cases. In the first case, if one part of the parallel structure moves to a new state, then the whole parallel structure will move to a corresponding state.

$$\frac{\langle NE_1, \mathcal{E}, ST, LR, CL\rangle \xrightarrow{a} \langle NE_1', \mathcal{E}', ST', LR', CL'\rangle}{\langle NE_1 \parallel NE_2, \mathcal{E}, ST, LR, CL\rangle \xrightarrow{a} \langle NE_1' \parallel NE_2, \mathcal{E}', ST', LR', CL'\rangle}$$

$$\frac{\langle NE_2, \mathcal{E}, ST, LR, CL\rangle \xrightarrow{a} \langle NE_2', \mathcal{E}', ST', LR', CL'\rangle}{\langle NE_1 \parallel NE_2, \mathcal{E}, ST, LR, CL\rangle \xrightarrow{a} \langle NE_1 \parallel NE_2', \mathcal{E}', ST', LR', CL'\rangle}$$

The second case is that if both parts in parallel structure terminate, then the whole parallel structure terminates as well.

$$\frac{}{\langle skip \parallel skip, \mathcal{E}, ST, LR, CL\rangle \xrightarrow{\tau} \langle skip, \mathcal{E}, ST, LR, CL\rangle}$$

The following rules are for operations $\mathcal{B}, \mathcal{F}$ and $\mathcal{R}$:

$$\frac{}{\langle \mathcal{B}p, \mathcal{E}, ST, LR, CL\rangle \xrightarrow{\mathcal{B}} \langle skip, \mathcal{E}^\frown \mathsf{bpair}, ST', LR, \mathsf{bpair}\rangle}$$

where element bpair is the most recent link point which, as the first element, forms a pair with CL in relation set LR. $ST' = ST \oplus (p.\{variable = null\})$.

$$\frac{}{\langle \mathcal{F}p, \mathcal{E}, ST, LR, CL\rangle \xrightarrow{\mathcal{F}} \langle skip, \mathcal{E}^\frown \mathsf{fpair}, ST', LR, \mathsf{fpair}\rangle}$$

where element fpair is the most recent point link which, as the second element, forms a pair with CL in relation set LR. $ST' = ST \oplus (p.\{variable = null\})$.

$$\frac{}{\langle \mathcal{R}p, \mathcal{E}, ST, LR, CL\rangle \xrightarrow{\mathcal{R}} \langle skip, \mathcal{E}^\frown CL, ST', LR, CL\rangle}$$

where $ST' = ST \oplus (p.linkState = enable)$.

The first expression means that if clicking back button, then (only) the user input on the last page will be removed. The second expression means that if clicking the forward button, then the user input (including null) will be forwarded to the next page. The third expression means to reset the link to the enable state, meanwhile automatically to set the duration time to the default.

3 SPIN Checking Navigation Model

In this section we discuss the use of the SPIN model checker [7] to check the navigation behaviors specified by our formal model. The input language of SPIN is called Promela, a modelling language for finite-state concurrent processes. SPIN model checker verifies (or falsifies, by generating counter- examples) LTL properties of Promela specifications using an exhaustive state space search.

We have defined some rules to translate our navigation model to Promela description. Due to the limited space, we only give some examples to illustrate our method.

- Conditional Choice $NE1 \triangleleft b \triangleright NE2$. Let $NE1$ and $NE2$ be two links, then

```
if
    :: b->atomic{NE1.nextlink='x';...};
    :: else->atomic{NE2.nextlink='y';...};
fi
```

- Nondeterministic Choice $NE1 \triangleleft \triangleright NE2$. Let $NE1$ and $NE2$ be two links, then

```
if
    //atomic1
    :: exp1->atomic{NE1.nextlink='x';...};
    //atomic2
    :: exp2->atomic{NE2.nextlink='y';...};
fi
```

Based on the transformation rules defined above, we have developed a prototype to support the automatic verification of web navigation. Some verification properties have been defined and can be automatically verified by SPIN. The properties that pass (are never violated) will return true, and for those that fail will give counterexample(s).

1) Dead end. For each page, we check the number of links:

```
assert(linkNumber!=0 || backState==true);
```

Once the linknumber is 0 and the backState is false, then we reach a dead end.

2) Broken link. We will define a global array to record all the pages being visited. After the navigation ends, if there still exist some pages not being visited, then we have broken links in this model.

3) Navigation not complete. We define the following sentence to check if $\mathcal{E}_s$ and $\mathcal{E}_{t1}$ still have some $\bullet$:

```
assert(empty(Es) && empty(Et1));
```

4) Reachability. We define a global variable with initial value 0. Once a link is clicked, the variable value increases 1. Thus the link number of a page can be obtained from this global variable.

5) Not removed data. We define the following sentence to check the variable values on the page to see if the values are null:

```
assert(var!='0');
```

We have used SPIN to check two navigation models from Amazon and Elsevier Web systems. For the first example, we check if there exist not removed user data for the Back operation and not complete navigation for the one time navigation. For the first one, we add the following statements:

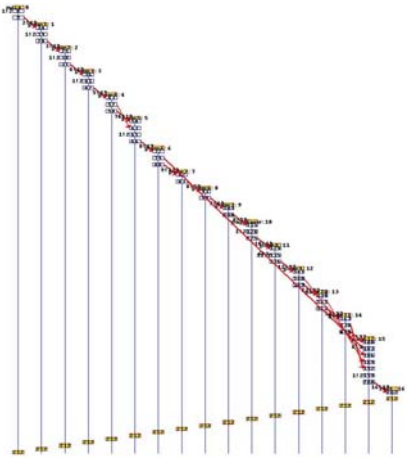

Fig. 1. SPIN checking Amazon

```
Back(var); Assert(var=='0');
```

For the second one, we check wether the following property is obeyed:

```
assert(empty(Es) && empty(Et1));
```

The simulation result is shown in Figure 1. The execution stops at the statement $assert(empty(Es)\&\&empty(Et1))$, which indicates that this assert statement is violated, and thus we have not complete navigation. We do not have not removed user data. The verification result is as the following:

```
pan: assertion violated ((q_len(Es)==0)
     &&(q_len(Et1)==0)) (at depth 161)
pan: wrote pan_in.trail
```

The result shows that at least one of the two channels Es and $Et1$ is not empty since either $q\_len(Es)! = 0)$ or $q\_len(Et1)! = 0)$ or both. Thus the navigation is not complete.

For the example of Elsevier submission system, we check the not complete navigation property by adding the following statements:

```
assert(empty(Es) && empty(Et1));
```

The execution will stop at the statement $assert(empty(Es)\&\&empty(Et1));$. This implies that we have not complete navigation. The verification result is as the following:

```
pan: assertion violated ((q_len(Es)==0)
     &&(q_len(Et1)==0)) (at depth 72)
pan: wrote pan_in.trail
```

The result shows that at least one of the two channels Es and $Et1$ is not empty. Thus the navigation is not complete.

4 Conclusion

We have proposed a formal model to specify web navigation precisely. The implementation semantics of the formal model is also presented. Based on the semantics of this model, we may simulate the user behaviors, and more importantly, we can find whether the user behaviors conform to the intentions of the system designers. To support the automatic checking of navigation model, we employ SPIN to check our navigation model and several properties can be checked. As the future work, we may study the click-action flow and use data mining skill to retrieve the natural language descriptions for these actions from log file, so that we can check the application-specific properties of the web navigation model. We will also continue to make our checking tool more scalable for web designers.

References

1. Alfaro, L.: Model Checking the World Wide Web. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, pp. 337–349. Springer, Heidelberg (2001)
2. Baumeister, H., Knapp, A., Koch, N., Zang, G.: Modelling Adaptivity with Aspects. In: Lowe, D.G., Gaedke, M. (eds.) ICWE 2005. LNCS, vol. 3579, pp. 406–416. Springer, Heidelberg (2005)
3. Ceri, S., Daniel, F., Demaldé, V., Facca, F.M.: An Approach to User-Behavior-Aware Web Applications. In: Lowe, D.G., Gaedke, M. (eds.) ICWE 2005. LNCS, vol. 3579, pp. 417–428. Springer, Heidelberg (2005)
4. Conallen, J.: Building Web Applications with UML. Addison-Wesley, Reading (2002)
5. Deutsch, A., Sui, L., Vianu, V.: Specification and verification of data-driven Web applications. Journal of Computer and System Sciences 73, 442–474 (2007)
6. Han, M., Hofmeister, C.: Modeling and Verification of Adaptive Navigation in Web Applications. In: ICWE 2006, Palo Alto, California, USA, July 11-14, pp. 329–336 (2006)
7. Holzmann, G.J.: Basic Spin Manual (1980), `http://cm.bell-labs.com/netlib/spin/whatispin.html`
8. Koch, N., Baumeister, H., Hennicker, R., Mandel, L.: Extending UML to Model Navigation and Presentation in Web Applications. In: Workshop on Modelling Web Applications in UML, UML 2000, New York, UK (October 2000)
9. Ricca, F., Tonella, P.: Analysis and Testing of Web Applications. In: Proc. of 23rd Int. Conference on Software Engineering, Toronto, Ontario, Canada, May 2001, pp. 25–34 (2001)
10. Rational Software, Pearl Circle Online Auction Reference Application Software Architecture Document, Issue 0.2, Rational Software (2001)
11. Winckler, M., Palanque, P.: Statewebcharts: A Formal Description Technique Dedicated To Navigation Modelling of Web Applications. In: Jorge, J.A., Jardim Nunes, N., Falcão e Cunha, J. (eds.) DSV-IS 2003. LNCS, vol. 2844, pp. 61–76. Springer, Heidelberg (2003)