

Modeling and Verification of Adaptive Navigation in Web Applications

Minmin Han
Lehigh University
19 Memorial Dr. W.
Bethlehem, PA 18015
1-610-758-6972
mih9@lehigh.edu

Christine Hofmeister
Lehigh University
19 Memorial Dr. W.
Bethlehem, PA 18015
1-610-758-4103
hofmeister@cse.lehigh.edu

ABSTRACT

The navigation of a web application is the possible sequences of web pages a user can visit. In the simplest case the next page is determined by the current page and the action (e.g. link, button) selected by the user. However, many web applications now incorporate *adaptive navigation*, where the next page also depends on the user's mode, for example whether they are a customer or an administrator, or depends on what pages the user has visited previously.

Navigation models are useful for clarifying requirements and specifying implementation behavior. When a model is formal, it can also be used to generate design or implementation artifacts, and can be verified for properties such as broken links or length of navigation path. These uses are all important for the case of simple navigation, but even more important for adaptive navigation because of the added complexity. However, none of the current formal approaches can support adaptive navigation.

In this paper we present an approach that uses Statecharts to formally model adaptive navigation, and show how important properties of a navigation model are verified using existing model-checking tools. We summarize the kinds of properties that can be checked with such a model, and describe how to use the Symbolic Model Verifier (SMV) tool to perform the verification. Finally, we use the Blockbuster® web site as a case study to demonstrate how our approach can uncover navigation problems that arise when new requirements are imposed.

Categories and Subject Descriptors

D.2.2 [Software Engineering]: Design Tools and Techniques – state diagrams.

D.2.4 [Software Engineering]: Software/Program Verification – model checking.

General Terms

Documentation, Design, Verification

Keywords

Navigation Model, Web Application, Adaptive Navigation

1. INTRODUCTION

The navigation of a web application is the possible sequences of web page a user can visit. In the simplest case the next page is determined by the current page and the action (e.g. link, button) selected by the user. However, many web applications now

incorporate *adaptive navigation*, where the next page also depends on the user's mode, for example whether they are a customer or an administrator, or depends on what pages the user has visited previously. We use part of the Blockbuster® web application (www.blockbuster.com) as an example of these two types of adaptive navigation.

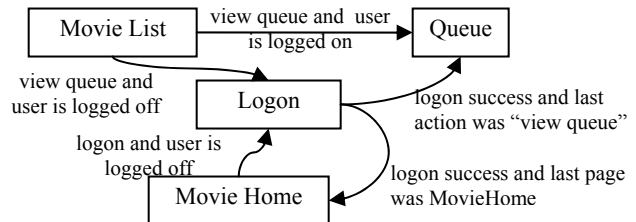


Figure 1 Navigation Map of part of Blockbuster®

Figure 1 shows a navigation model, a navigation map [10][22], which represents part of the navigation in Blockbuster®. In a navigation map, the boxes are web pages, the arrows that connect web pages are navigation links between them, and the text annotating the arrows describes what triggers the navigation link.

From the Movie List page a user can click a link “view queue”. If the user is logged on, they next see the Queue page. But if the user is logged off, they next see the Logon page. Thus whether a user can navigate directly from the Movie List page to the Queue page depends on the user's mode: if the user is currently logged on then the navigation link is allowed. This is an example of *user-mode adaptive navigation*, where the navigation depends on the user's mode.

Another type of adaptive navigation is *history-sensitive navigation*, where the navigation depends on previous actions or previously visited pages. At the Logon page, the user can choose to logon. If the previous action was “view queue”, the next page will be the Queue page. But if instead the Logon page was requested from Movie Home, then the page following Logon will again be Movie Home.

Navigation models are useful for clarifying requirements and specifying implementation behavior. When a model is formal, it can also be used to generate design or implementation artifacts, and can be verified for properties such as broken links or length of navigation path. These uses are all important for the case of simple navigation, but even more important for adaptive navigation because of the added complexity.

When no adaptive navigation is involved, it is easy to see whether a page in the navigation map is unreachable: this is simply any

page with no incoming navigation links. Similarly, a page that is a dead end has no outgoing arrows in the navigation map.

But when adaptive navigation is present it is much harder to check even these simple properties. For example, a page could have incoming navigation links, but the conditions that allow the link to be taken will never be true. In Figure 2, to navigate from page Y to page Z the user must be logged on. However, although user can be logged on at other pages, further examination of the navigation map shows that it is not possible for a user to be logged on while visiting page Y. Thus page Z is not reachable.

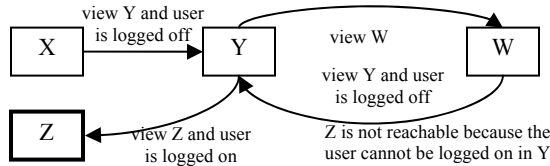


Figure 2 Unreachable Web Page

Another example is a page that has outgoing navigation links, but because of mode constraints on the links, the page is actually a dead end. In Figure 3, Y not only has an outgoing link, its condition is the same as for the incoming link, so it seems not to be a dead end. But the mode can change while at page Y, for example with a logon timeout, which leaves page Y a dead end.

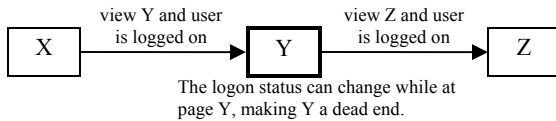


Figure 3 Dead End Web Page

The presence of modes in the navigation means that in some cases a page should not be displayed unless the user's mode has a particular value (e.g. logged on). If all incoming navigation links have logged on as a condition, then clearly the user must be logged on before viewing the page. But other cases are more difficult to assess. In Figure 4, we wish to check that page Y cannot be reached unless the user is logged on, but this is not a condition of all the links that reach Y. However, page Z requires the user to be logged on. We must follow all possible navigation paths to Y and make sure the mode is checked on each of these. We must also check that pages on these paths cannot change the mode if these pages are downstream from the last mode check.

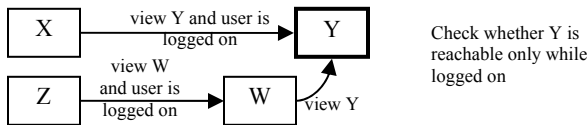


Figure 4 Security of Web Page

In this paper we present FARNav (a Formal Approach for Rich Navigation modeling) that solves these problems by providing a formal approach for describing adaptive navigation, and uses the Symbolic Model Verifier (SMV) tool to check the correctness of a navigation model. Section 2 describes the navigation modeling approach and how a navigation model is translated into CTL, the input language for SMV. Then it describes how to check desirable properties of a navigation model, and how to express these in CTL. Section 3 presents a case study we did on the Blockbuster® web application, in order to demonstrate how FARNav can

uncover navigation problems that can arise when new requirements are imposed. Section 4 discusses related work, and Section 5 concludes the paper.

2. A FORMAL NAVIGATION MODEL

Our formal navigation model uses the Statechart notation [14] to specify the navigation of an application. Statecharts have a graphical representation (e.g. in UML) that is easy for most software engineers to read and understand. But the Statechart notation also allows us to use model-checking methods and tools in order to verify properties of a navigation model.

2.1 Modeling Adaptive Navigation

To model adaptive navigation, we use parallel (ANDed) substates. The main substate contains a state machine having one state per web page, and transitions between pages for the navigation links. When a web application has only simple (no adaptive) navigation, this substate comprises the entire navigation model.

When adaptive navigation is present, each mode is represented by a separate parallel substate containing its own state machine. These are parallel substates because a user is in multiple states at the same time, one for each state machine. For example, the user can be visiting the MovieHome page in the Page Navigation substate machine and is LoggedOn in the Logon Status state machine.

Figure 5 shows an overview of the navigation model for an application with two different modes, so the navigation model has three parallel states. Each substate contains a separate state machine, and they interact via events generated in one substate and received in another. There can be no transitions from one of these parallel substates to another.

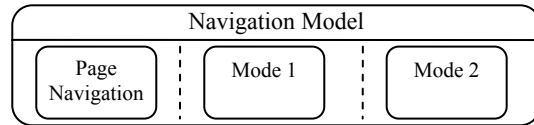


Figure 5 Top View of Navigation Model

The state machine for the Page Navigation substate is created as follows:

- state: Each web page is a separate state. It is up to the developer to determine what constitutes a distinct web page in an application.
- transition: When the user can reach one page from another, there is a transition between the corresponding states. The transition has a label with the form:

`<event> [<guard>] / <action>`

Only the event is mandatory. It describes a user action that causes a request to be sent to the server, and the subsequent response from the server. It can also describe a request generated by the browser, e.g. a timeout.

The guard allows a transition to fire only when a mode has a particular value. For example, it can say that the user must be LoggedOn to fire this transition. It has the form `[in <state of mode x> & in <state of mode y> & ...]`.

The transition may fire an action that appears as an event within a mode state machine. This triggers a transition that causes the mode to change its value.

We model only the navigation between pages: we do not model the navigation inside a page nor explicitly model the frames.

The state machine for a Mode substate is created as follows:

- states: In the case of a user mode, there is one state for each possible value of the mode (e.g. LoggedOn, LoggedOff). History-sensitive navigation is also modeled with a mode. In this case there is one state for when previously visited pages are not relevant (e.g. NotInLogon), and one state for each case where a previously visited page must be remembered (e.g. Logon-Queue when Queue should follow Logon, and Logon-MovieHome when MovieHome should follow Logon).
- transitions: the transitions are usually triggered with events fired in the Page Navigation state machine, or with timing events such as timeout. There is no guard or action for these transitions.

Figure 6 present a formal model for what is shown in Figure 1. Four pages are described here: MovieList page, MovieHome page, Logon page and Queue page.

A user can click “view queue” from the MovieList page whether logged on or not. A user who is not logged in will next see the Logon page. In Page Navigation state machine, this results in a transition from MovieList to Logon, triggered by the event ‘queue’ (the user clicked “view queue”), guarded by the condition [in LoggedOff] (the user is logged off), and generating the action ‘logon-Queue’. The latter instantaneously triggers a transition in the Logon Next-Page state machine, from state NotInLogon to state Logon-Queue.

Next the user fills out the logon information and clicks “logon” on the Logon page. When the logon is successful, the Queue page is displayed. In Page Navigation state machine, this is described by the transition from Logon to Queue: it is triggered by logonSuccess, provided the Logon Next-Page state machine is in state Logon-Queue, which it is. The two actions ‘loggedOn’ and ‘notInLogon’ are fired, and they cause a transition from LoggedOff to LoggedOn, and from Logon-Queue to NotInLogon.

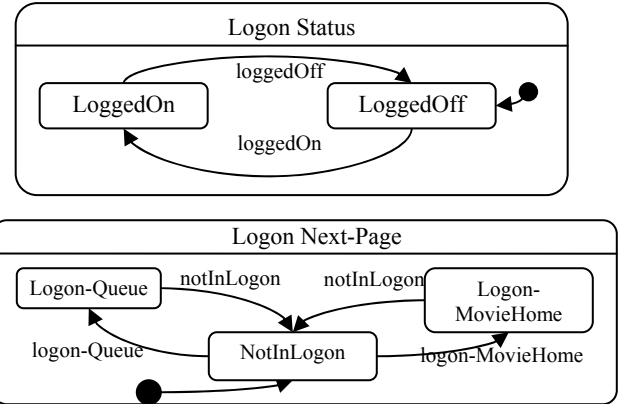
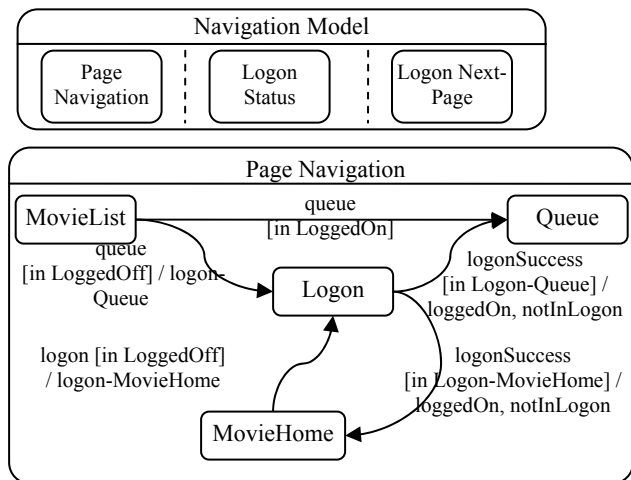


Figure 6 Logon Navigation Model for Blockbuster®

Only the standard Statechart notation is used for our navigation models, so it is also possible to represent these models with a textual notation such as STATEMATE[15]. For the Page Navigation state machine, which is highly variable, we believe a diagram is easier to understand. But the Mode state machines are very regular, and for history-sensitive navigation they can get quite large. Thus for these state machines a textual or tabular notation may be easier to create and understand.

2.2 Translating the Navigation Model to CTL

We use the approach in [1] to translate our navigation model into the Computational Tree Logic (CTL) language [9], then use the Symbolic Model Verifier (SMV) tool [5] to verify the model. The translation rules in this approach provide a translation of each element of a Statechart model.

The navigation model in Figure 6 translates into CTL as follows. The first part of the CTL model is shown in Figure 7.

- For each state machine (Page Navigation, Logon Status, and Logon Next-Page), a variable is declared (lines 3-5). The domain of each variable is the set of states in its state machine.
- The events recognized by the Page Navigation state machine are boolean variables (lines 6-8). Then a variable is created that will be used when executing the state machine, in order to randomly generate the three events (line 9).
- The initial state for each state machine is set. (lines 11-13).
- Events are initialized to 0 (false), which means at startup no event is true (lines 14-16). The test variable is also initialized to 0 (line 17), meaning that no event is selected.
- Lines 19-33 define the triggers for the transitions. First those in the Page Navigation state machine are described directly, by stating the current page then the event and condition that trigger the transition.
- Next the events recognized by the state machines in the mode substates are defined, by stating the transitions in Page Navigation that generate these events (lines 24-27). The loggedOff event is not included because the portion of the navigation shown in our example does not generate this event.
- Lines 28-33 define the transition triggers for Logon Status (28-29) and Logon Next-Page (30-33). Since their events are now defined, the transition triggers can be defined in the same way

as for those of Page Navigation. Note that T_{O_F} (transition from LoggedOn to LoggedOff) is always false, since our example will never make this transition.

```

1 MODULE main
2 VAR
3   Page : {Signin, MovieHome, Queue, MovieList};
4   LogonStatus : {LoggedOn, LoggedOff};
5   LogonNextPage: { Original, ToQ, ToMH };
6   logon: boolean;
7   logonSuccess : boolean;
8   queue: boolean;
9   test: {0, 1, 2, 3};
10 ASSIGN
11   init(Page) := Logon;
12   init(LogonStatus) := LoggedOff;
13   init(LogonNextPage) := NotInLogon;
14   init(logon) := 0;
15   init(logonSuccess) := 0;
16   init(queue) := 0;
17   init(test) := 0;
18 DEFINE
19   T_MH_L := (Page = MovieHome) & logon & (LogonStatus = LoggedOff)
20   T_L_MH := (Page = Logon) & logonSuccess
                & (LogonNextPage = LogonMovieHome);
21   T_ML_L := (Page = MovieList) & queue & (LogonStatus = LoggedOff);
22   T_ML_Q := (Page = MovieList) & queue & (LogonStatus = LoggedOn);
23   T_L_Q := (Page = Logon) & logonSuccess
                & (LogonNextPage = LogonQueue);
24   loggedOn := T_L_MH | T_L_Q;
25   logonQueue := T_ML_L;
26   logonMovieHome := T_MH_L;
27   notInLogon := T_L_Q | T_L_MH;
28   T_O_F := 0;
29   T_F_O := (LogonStatus = LoggedOff) & loggedOn;
30   T_NOT_LQ := (LogonNextPage = NotInLogon) & logonQueue;
31   T_LQ_NOT := (LogonNextPage = LogonQueue) & notInLogon;
32   T_NOT_LMH := (LogonNextPage = NotInLogon) & logonMovieHome;
33   T_LMH_NOT := (LogonNextPage = LogonMovieHome) & notInLogon;

```

Figure 7 First Part of CTL file

The second part of the CTL model decides the value of the events and the states at the next clock tick (Figure 8):

- The test variable is randomly assigned a value from 1 to 3 (line 35). This value will be used to select the next event to be generated (logon, queue, or logonSuccess).
- The event at the next clock tick is true only if the test variable is equal to its index number. So at any time one and only one event is true. (line 36-50)
- The end state for each transition is defined in lines 51 to the end of the file. For each state machine, if one of its transition triggers is true, the state at the next clock tick is set. For example, line 53 says that if T_{L_MH} holds, the next state for the Page Navigation state machine is MovieHome.

```

34 ASSIGN
35   next(test) := {1, 2, 3};
36   next(logon) :=
37     case
38       (test = 1) : 1;
39       1 : 0;
40     esac;
41   next(queue) :=
42     ...
46   next(logonSuccess) :=
47     ...
51   next(Page) :=
52     case
53       T_L_MH : MovieHome;

```

```

54   T_MH_L, T_ML_L : Logon;
55   T_ML_Q, T_L_Q : Queue;
56   1 : Page;
57   esac;
58   next(LogonStatus) :=
59     ...
64   next(LogonNextPage) :=
65     ...

```

Figure 8 Second Part of CTL File

Because the rules for translating one of our navigation models into CTL are straightforward, we expect it to be similarly straightforward to completely automate the translation process.

2.3 Analysis of the Navigation Model

Using the CTL version of our formal navigation model, verification rules can be placed at the end of the CTL file and the SMV tool can be used to automatically verify the rules. The rules that pass (are never violated) will return true, and for those that fail the tool will give a counter example.

Table 1 shows the CTL formulas we found to be useful for verifying a navigation model.

Table 1 Some CTL Formulas

CTL code	Description
AG a	a holds on all paths from initial state.
EF a	a holds on at least one of the path from initial state.
AG EF a	a holds on some places on all paths from initial state.
AG a -> b	In all paths if a holds then b holds
AG a -> AX b	In all paths if at this state a holds, then b must hold at next state.
AG a -> EX b	In all paths if at this state a holds, then b must hold in some branch at next state.
AG a -> EF EX EX b	In all paths if at this state a holds, then b must hold in some branch at next next state.
AG a -> EF b	In all paths if at this state a holds then b may hold in some branch later.
!E (!a U (b & !a))	b cannot be true if a has not been true.

We have split the possible verification rules into four categories: page reachability, mode reachability, page/mode coupling and page or mode sequences.

The first group of verification rules is about the reachability of web pages. In this group, we can check the general reachability of a web page, meaning whether the page can be reached through some navigation path. An unreachable page is most likely an error because it is obsolete or due to design errors. Also we can check for a dead end, meaning a page a user can visit but cannot leave. While some dead ends are desirable, such as a page that acknowledges a logoff, other dead ends may be design errors. Finally, we can check the reachability of web pages within certain number of links. Sometimes usability requirements dictate that a user can reach certain web pages in one or two steps. A usability requirement could be that the logoff page must be reachable from any page by just one link.

Similarly, the second group of verification rules that we can apply on this navigation model is about the reachability of modes. Usually all modes should be reachable. Some types of the dead end of mode change may be acceptable, such as once a user has signed on, he or she remains signed on indefinitely.

The third group is about the allowed modes when accessing pages. Designers may want some web pages to be viewable only in certain modes. If such a rule is broken, it could cause a serious breach of security (e.g. an unauthorized user visits a page that should be accessible only by administrators). In addition designers may want to ensure that some mode can change only at certain web pages (e.g. logon).

The fourth category is for checking the sequence of pages in a navigation path. For example, a confirmation page must have been displayed at some time before the user see the “order done” page.

Table 2 lists sample rules we expect to be useful for verifying a navigation model. Although designers are typically not familiar with CTL, they should be able to follow these patterns in order to write their own rules. Designers can refer to CTL property specification patterns [11] to write CTL code for verification rules we have not thought of. However, it would clearly be preferable for the designer not to use CTL at all. To accomplish this, in addition to providing a translator for producing the CTL description of a model, we envision a simple language for writing the kinds of rules needed for navigation models, and a tool to automatically translate these into CTL rules.

3. CASE STUDY AND DISCUSSION

A FARNav model is useful in both forward engineering and reverse engineering. For forward engineering, the model is created in accordance with the client’s requirements, and design

rules are created to describe what the navigation should or should not do. Then it can be verified to ensure that the navigation conforms to the user requirements and that it obeys the design rules. The latter can be verified automatically. Finally, the model provides a basis for further design and implementation.

For reverse engineering, this model can be created by observing the behavior of the application. Then it provides documentation in order to improve understanding. In addition, after specifying appropriate design rules, it can be used to verify the correctness of navigation changes made during maintenance. Our case study uses FARNav in the second way.

3.1 Creating the Navigation Model

We created a navigation model of a portion of the Blockbuster® web application. This portion of the navigation covers user logon, browsing for movies, accessing and updating the queue (which holds the list of movies the user wishes to rent), and several related pages.

To create the model we first observed the behavior of the web application. Screens providing a similar type of content are treated as one web page; for example two screens presenting movie details for different movies treated as one web page. For this part of Blockbuster®, there are 21 web pages and 79 navigation links among these pages. We observed one user mode related to the logon status, and three places where history-sensitive navigation is used: for the Logon page, the Locate Stores page, and the Add to Queue page.

Table 2 Categories of Verification Rules and CTL Patterns

Page Reachability		
Category	CTL Formula	Sample Verification Rule
Reachability	EF (PageVar = PageName)	There is at least one navigation path from the start page to Queue page. EF (Page = Queue)
Dead End	AG EF (PageVar = pageName)	Along any path from start point it is always possible to reach Queue page. (Queue page is reachable from any other page.) AG EF (Page = Queue)
	AG ((PageVar = Page1Name) -> EF (PageVar = Page2Name))	Along any path from start point, it is possible to reach Queue page after visiting Logoff. AG ((Page = LogOff) -> EF (Page = Queue))
Reach certain page in certain number of steps	AG ((PageVar = Page1Name) -> AX (PageVar = Page2Name))	After Order Step 1 page the only possible next page is Order Step 2 page. AG ((Page = OrderStep1) -> AX (Page = OrderStep2))
	AG ((PageVar = Page1Name) -> EX (PageVar = Page2Name))	It is possible to visit sign in page from sign off page. AG ((Page = LogOff) -> EX (Page = LogOn))
	AG ((PageVar = Page1Name) -> EF EX EX (PageVar = Page2Name))	It is possible to visit Queue page within 2 links from Movie Home page. AG ((Page = MovieHome) -> EF EX EX (Page = Queue))
Mode Reachability		
Reachability	EF (ModeVar = ModeName)	It is possible for a user to be in “logged-on” mode sometime. EF (LogonStatus = LoggedOn)
Dead End	AG EF (ModeVar = ModeName)	It is always possible for a user to be in “logged-off” mode on all navigation paths. AG EF (LogonStatus = LoggedOff)
	AG ((ModeVar = Mode1Name) -> EF (ModeVar = Mode2Name))	It is possible for a user to switch to “logged-off” mode after the user is in “logged-on”. AG ((LogonStatus = LoggedOn) -> EF (LogonStatus = LoggedOff))
Page/Mode Coupling		
Allowed page and mode pairs	AG ((PageVar = PageName) -> (ModeVar = ModeName))	A user must be logged-on to visit account page. AG ((Page = Account) -> (LogonStatus = LoggedOn))
Mode change only within certain pages	!E(!(PageVar = PageName) U ((ModeVar = ModeName) & !(PageVar = PageName)))	A user must visit Logon page to change to logged-on mode. !E (!(Page = Logon) U ((LogonStatus = LoggedOn) & !(Page = Logon)))
Page or Mode Sequences		
One page must be visited before another	!E (!(PageVar = Page1Name) U ((PageVar = Page2Name) & !(PageVar = Page1Name)))	A user must visit Signin page before visiting Queue page. !E (!(Page = Logon) U ((Page = Queue) & !(Page = Logon)))

Thus we created five parallel states. The first, for Page Navigation, contains a state machine with a state for each of the 21 web pages, and 79 transitions representing the 79 navigation links we found. The state machine for Logon Status has just two states, and two transitions between them. The state machines for the remaining three history-sensitive modes all have one state for when the user is not at the page where history affects the navigation (e.g. NotInLogon), and a state for each page that can link to that page. Transitions to the NotInLogon are all labeled with the same event notInLogon, indicating that history-sensitive navigation is no longer in effect. Transitions from NotInLogon are labeled with an event indicating what page should follow the Logon page (e.g. logon-Queue, logon-MovieHome).

3.2 Verifying the Navigation Model

Next we created a number of design rules for the navigation model. The first set checks the general reachability of all web pages. In CTL, these are stated one per line in the form: EF (page = pageName). The second set checks for dead ends, and these have the form: AGEF (page = pageName).

The next set of checks is related to the page/mode coupling, in order to check the security of the application. The Queue, Logoff, and Account page all ask the user to first logon. Thus we created rules (Rule1-3) stating that the user must be logged on in order to visit these pages:

Rule 1: AG ((page = Queue) -> (logonStatus = LoggedOn))

Rule 2: AG ((page = Logoff) -> (logonStatus = LoggedOn))

Rule 3: AG ((page = Account) -> (logonStatus = LoggedOn))

Not surprisingly, the navigation passes all of these tests. These kinds of navigation errors should never be present in a deployed application.

The fourth check asks a different question about the security of the application. Users typically want the ability to explicitly log off before they leave a web site, and we observed that the Blockbuster® application provides this feature. Rule 4 states that if a user is logged on, he or she can log off from any page:

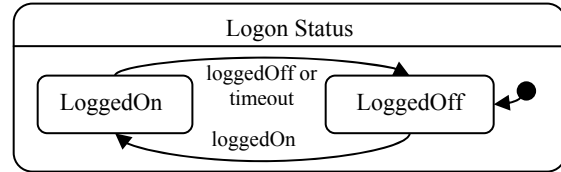
Rule 4: AG ((logonStatus = LoggedOn) -> EX (page = SignOff))

The current design does not pass this test. The counter example given by the SMV tool is that the Customer Service page does not allow a user to log off. To correct this, in the navigation model we added a transition from the Customer Service page to the Logoff page, and then the rule passes. Of course, the implementation needs to be modified accordingly.

Thus we found a situation where the application behavior probably violates the original intent. To show an example of how the model can help when performing maintenance on an application, we add a new feature to the application.

In current version of Blockbuster®, once a user is logged on, they will stay logged on until explicitly choosing to log off. This is not very secure because a user may forget to log off before leaving, then another person can access the user's personal information such as credit card number. So we wish to add a timeout feature: if the user has taken no action for at least 15 minutes, a timeout event will occur and the user will be logged off.

To model this navigation change, we add a new event (timeout) as a trigger from the LoggedOn state to the LoggedOff state in the Logon Status mode.



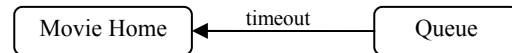
Upon re-checking the design rules, we find that Rule 1 does not hold. However, there are two main ways an application can behave in response to something like a logon timeout. One is to display a different page (e.g. Movie Home) when a logon times out. The other is to allow the logon to expire without changing the page currently displayed, but prevent navigation to any other secure pages.

We know that the former does not hold for our current navigation, since the above rule is violated. To see whether the latter behavior holds currently, we need to ensure that when the user is logged off, they may not navigate to any secure page (such as Queue page). Rule 5 checks this:

Rule 5: AG (AX(page = Queue) -> (logonStatus = LoggedOn))

Our current model does pass this rule (along with similar ones for the Logon and Account pages).

If we desire the navigation behavior where a different page is displayed when a logon times out, we add the following transition from a secure page to the Movie Home page:



Now the model passes the check Rule 1, and we add similar transitions for each secure page.

3.3 Scalability of the Notation

In our diagrams so far we have shown a small portion of the Page Navigation state machine, which has 21 states and 79 transitions. With a tabular notation the specification of the state machine has no problem scaling, but this format is not as easy for humans to understand. However a graphical notation such as we use in this paper does not scale well. In this section we suggest some techniques to handle scalability in a graphical notation.

Scalability is also an issue for web sites, so typically a larger web site has hierarchical groupings of its web pages. Since Statecharts support hierarchy, the Page Navigation model should have the same hierarchical structure that the web site exhibits.

In Blockbuster® there is a group of web pages for movie rentals, one for game rentals and another for the remaining pages, so we mirror this structure in the Page Navigation state machine. Although Page Navigation is a substate in the complete navigation model, it also acts as a superstate to a set of nested substates. For Blockbuster®, we model the general web pages and movie rental pages as substates of Page Navigation. Each of these contains approximately ten states.

In CTL, we use two macros to identify which superstate the current page is in: Home, About, Affiliates, Careers, Rights,

IRelations, Signin, SignOff, CustomerService and Account pages are in the General Pages substates while the other pages are in the Movie Pages state machine.

```

DEFINE
inGeneralPages := (page = Home) | (page = About) | (page = Affiliates) |
(page = Careers) | (page = Rights) | (page = IRelations) | (page = Signin) |
page = SignOff) | (page = CustomerService) | (page = Account);
inMoviePages := (page = Queue) | (page = MovieHome) | (page =
MovieList) | (page = MovieDetail) | (page = SearchResult) | (page =
AdvancedSearch) | (page = FindStore) | (page = ActorList) | (page =
MovieInStores) | (page = MovieAddedToQueue);

```

We can also represent the transitions more compactly, again by using superstates. Since the user can link to the Movie Home page from every other page in the Movie Pages state machine, we can replace the ten transitions from another state to Movie Home with a single transition from the Movie Pages superstate to the Movie Home state. This is easily translated into CTL by adjusting the transition conditions.

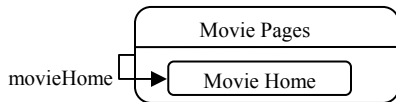


Figure 9 Transition Shorthand

```

DEFINE
T_MOVIEPAGES_MH := inMoviePages & gotoMovieHome;

```

The use of nested states and superstate transitions does not affect the substance of the model, but can help to reduce the complexity of understanding the navigation models.

4. RELATED WORK

Many of the modeling techniques for web applications include a navigation model. The most basic navigation model is a descriptive model that is used to represent the navigation, such as the “Architecturally Significant Navigation Map” in Pearl Circle[22]. A navigation map contains request pages, response pages, arrows to connect them, and a label on the arrow to explain why the user would arrive at the response page. The figures in Section 1 are examples of this kind of navigation model. Another modeling technique in this category is the navigation model of Koch et. al., which uses an extended UML notation [16].

At the next level are models that support forward engineering by generating code skeletons. This group includes the ADM-d site scheme for data-intensive web sites in the Araneus approach[19], the navigation model in WebML[8], and “navigational contexts schema” in OOHDM [24]. Book et. al. take this one step further, with a model that describes a mix of navigation information and related server processing [4]. This web-based dialog flow model can be translated into tables for runtime lookup in the implementation so designers can write only business logic while the navigation is automatically generated from the model.

However, none of the above models can be verified for correctness.

Navigation models such as HMBS (Hypermedia Model Based on Statecharts) [20] and the navigation model by Leung et. al. [17] use Statechart notation. The states in those models are web pages and frames while the transitions are the navigation links. Both of

models are used mainly for representing navigation and providing user different levels of views by using hierarchical states.

Finally, there are graph-based navigation models such as the web application structure model by Alfaro [2] and the web site organization diagram by Ricca et. al [23]. In these models, web pages and frames are nodes and navigation links are edges. These models can be used to verify page reachability, dominators of the navigation path, navigation path length, strongly connected components, broken links, and frame errors. Also it is possible to do pattern matching to find out if the navigation model contains a diamond structure, tree structure or index structure. Either a tool (ReWeb) or formal method (constructive μ -calculus) is used to perform automatic verification.

While the graph-based models provide some automatic model checking, none of the above models support adaptive navigation, except those in the first group that are purely descriptive. Thus none support the automatic verification of adaptive navigation. However, since FARNav uses Statecharts, the limitations of state machines modeling capabilities make it difficult to verify certain properties that are easy to verify with a graph-based model. For example, it is difficult to count the length of the navigation path with our approach.

Adaptive web applications are web applications that can “semi-automatically improve their organization and presentation by learning from visitor access patterns”.[21] The navigation in an adaptive web application can be dynamically changed according to the user’s status, the user’s visiting path, and other contextual information. Casteleyn et. al. present a navigation model to support adaptive behavior such as navigation promotion and demotion during runtime [6]. Baumeister et. al. use aspect-oriented modeling techniques to model adaptive web applications. Links are associated with aspects that read the user’s status and change the navigation accordingly [3]. Ceri et. al. use ECA rules to support navigation in context-aware and adaptive web applications [7].

For navigation in adaptive web applications, we can split the problem into two parts. One is defining the conditions that cause the user mode to change, and the other is defining how user modes affect the navigation. We model the latter, but except in some simple cases we do not model the former. In the Blockbuster® example, the model shows that a timeout event causes the Logon Status mode to change, so for this case the model does describe what causes the mode change. But when the mode changes via the Logon or Logoff page, the model simply captures the fact that the mode can change, not what constitutes a successful login.

The other work on adaptive web applications described above typically looks at both of these parts. Although different approaches have different degrees of support for the two parts, none support the verification of the resulting models.

5. CONCLUSION

In this paper we presented FARNav, a formal approach for navigation modeling that uses Statecharts to describe adaptive navigation. No previous approaches can formally model adaptive navigation.

While a model need not be formal in order to be useful for clarifying requirements or for guiding development, when a model is formal it can be used in several additional ways. First, it can be automatically verified to check important properties of the navigation. FARNav currently supports this using an existing tool (SMV). Second, it can be used to automatically generate design and implementation artifacts. Third, it can be used during testing to ensure that the implementation conforms to the navigation model. This could be done by instrumenting the web browser or server to produce traces of request and responses, then automatically checking them against the model. These last two uses are ones envisioned for FARNav in the future.

Because FARNav makes the adaptive aspects of the navigation explicit in the model, the uses listed above are possible for adaptive navigation, which is significantly more complex than simple, non-adaptive navigation. FARNav can verify rules that constrain how user modes and pages may be combined, for example rules stating that a user must be logged in or must be an administrator in order to visit a particular set of pages.

We applied the current version of FARNav to part of the Blockbuster® web application. For this case study we created the navigation model, then inferred a set of design rules based on the observed behavior of the application. For example, we inferred rules stating that the user must be logged in order to view the Queue, Account, and Logoff pages. Another rule we inferred is that the user may logoff from any page. After checking this rule with SMV, we found that it is not universally true: there is one page, the Customer Service page, from which the user cannot logoff. Since there is no obvious reason for this exception, the exception may well be the result of an oversight.

We also showed the utility of the navigation model during maintenance by adding a new feature, a logon timeout. We showed how the new feature is added to the navigation model, then the design rules are automatically checked. Thus we discover whether additional changes to the navigation are needed in order to support the new feature, and where they are needed.

In addition to further use of a navigation model throughout the development process, we see several other areas for future work. One is to improve our ability to scale the model for very large applications, both by making use of hierarchical substates and by tools that support a kind of slicing on statecharts. While the hierarchical substates change the model itself, the slicing would simply provide different visualizations of the model, by displaying only part of the model at a time.

Another is to expand the modeling concepts to cover the “back button” problem (see for example [18]). This phenomenon occurs when instead of using actions in the page to navigate, the user clicks the back button of the web browser. This potentially gives rise to a situation where the content seen by the user is inconsistent with what is stored on the server. We plan to expand the navigation model to describe in which pages the inconsistency may occur, and how the user will be shielded from such inconsistencies.

Finally, we earlier developed a model for describing “navigation routing” [12][13], by which we mean how requests are routed through components on the server. Thus we plan to put these two kinds of navigation models together, to cover both the navigation

visible to the user and the navigation of requests through server components.

6. REFERENCES

- [1] Aderson, R. J., Beame, P., Burns, S., Chan, W., Modugno, F., Notkin, D., and Reese, J. D. Model Checking Large Software Specifications. *IEEE Trans. Softw. Eng.*, 24, 7 (Jul. 1998), 498-520.
- [2] Alfaro, L. de. Model checking the world wide web. *Computer Aided Verification*, Lecture Notes in Computer Science, vol 2102, Springer-Verlag, 2001, 337-349.
- [3] Baumeister, H., Knapp, A., Koch, Nora, and Zhang, G. Modelling Adaptivity with Aspects. *Web Engineering, Proceedings of ICWE*, Springer-Verlag, 2005, 406 – 416.
- [4] Book, M., Gruhn, V. Modeling Web-Based Dialog Flows for Automatic Dialog Control. In *Proceedings of 19th IEEE International Conference on Automated Software Engineering (ASE 2004)*, (Sep 2004), Linz, Austria. IEEE Computer Society 2004, 100-109.
- [5] Cadence Berkeley Labs. SMV model checking system.
- [6] Casteleyn, S., De Troyer, O., Brockmans, S. Design Time Support for Adaptive Behavior in Web Sites. In *Proceedings of the 18th ACM Symposium on Applied Computing*, Melbourne, USA (2003).
- [7] Ceri, S., Daniel, F., Demaldé, V., and Facca, F. M. An Approach to User-Behavior-Aware Web Applications. *Web Engineering, Proceedings of ICWE*, Springer-Verlag, 2005, 417 – 428.
- [8] Ceri, S., Fraternali, P., Bongio, A. Web Modeling Language (WebML): a Modeling Language for Designing Web Sites. *Proceedings of the 9th International World Wide Web Conference*, Elsevier, Amsterdam, Netherlands, (May, 2000).
- [9] Computational Tree Logic (CTL) http://en.wikipedia.org/wiki/Computational_tree_logic
- [10] Conallen, J. Building Web Applications with UML. Addison-Wesley. 2002.
- [11] Dwyer, M. B., Avrunin, G. S., Corbett, J. C. Patterns in Property Specifications for Finite-State Verification. In *Proceedings of the 21st International Conference on Software Engineering*, (May, 1999), 411-420.
- [12] Han, M., and Hofmeister, C. Modeling Navigation Routing in J2EE Web Applications. Lehigh University Technical Report, 2004.
- [13] Han, M., and Hofmeister, C. Separation of Navigation Routing Code in J2EE Web Applications. ICWE 2005. pp 221- 231.
- [14] Harel, D. Statecharts: A visual Formalism for complex systems. *The Science of Computer Programming*, 8, (1987), 231-274.
- [15] Harel, D. and Naamad, A. The STATEMATE semantics of statecharts. *ACM Trans. Softw. Eng. Methodol.* 5, 4 (Oct. 1996), 293-333.
- [16] Koch, N., Baumeister, H., Hennicker, R. and Mandel, L. Extending UML to Model Navigation and Presentation in

- Web Applications. In Procs. *Modelling Web Applications in the UML Workshop, UML 2000*, York, England, (Oct 2000).
- [17] Leung, K. R. P. H., Hui, L. C. K., Yiu, S. M., and Tang R. W. M. Modelling Web Navigation by Statechart. In *Proceedings of the 24th Annual International Computer Software and Applications Conference (COMPSAC 2000)*, (Oct 2000), 41-47.
- [18] Licata, D. R., and Krishnamurthi, S. Verifying Interactive Web Programs. In IEEE International Symposium on Automated Software Engineering (ASE 2004), 164-173.
- [19] Merialdo, P., and Atzeni, P. Design and Development of Data-intensive Web-Sites: The Araneus Approach. *ACM Transactions on Internet Technology*, 3, 1, (Feb 2003), 49-92.
- [20] Oliverira, M. C. F. de, and Turine, M. A. S., and Masiero, P. C. A Statechart-based Model for Hypermedia Applications. *ACM Transactions on Information Systems*, 19, 1, (Jan 2001), 28-52.
- [21] Perkowitz, M., and Etzioni, O. Towards Adaptive Web Sites: Conceptual Framework and Case Study. *Artificial Intelligence*, 118, 1-2, (Apr 2000), 245-275.
- [22] Rational Software. Pearl Circle Online Auction Reference Application Software Architecture Document, Issue 0.2. Rational Software 2001.
- [23] F. Ricca and P. Tonella. Analysis and testing of web applications. In Proc. of ICSE 2001, International Conference on Software Engineering, Toronto, Ontario, Canada, (May 2001), 25-34.
- [24] Schwabe, D., and Rossi, G. An Object Oriented Approach to Web-Based Application Design. *Theory and Practice of Object Systems*, 4, 4, (1998), Wiley and Sons, New York.