

Relating Navigation and Request Routing Models in Web Applications

Minmin Han and Christine Hofmeister
 Lehigh University
mih9@lehigh.edu, crh@cse.lehigh.edu

Abstract

A navigation model describes the possible sequences of web pages a user can visit, and a request routing model describes how server side components handle each request. Earlier we developed formal models and analysis operations for such models. While each is useful independently, their utility is greatly improved by relating the models, which is the contribution described in this paper. We describe mappings between the models, and show that the mappings preserve navigation behavior and are bijective, thus supporting traceability and allowing the models to be used in round-trip engineering. With these mappings built into our Model Helper tool, it is now possible to automatically determine whether a Request Routing model conforms to the navigation design, and to automatically generate a Request Routing model from a navigation model. Finally, we describe one of a number of case studies where we used Model Helper in a round-trip engineering scenario.

1. Introduction

The navigation of a web application is the possible sequences of web pages a user can visit. For simple cases, the next page displayed depends only on which button or link the user selects in the current page. But web applications often use *adaptive navigation*, where the next page may also depend on: the user's mode, for example whether they are a customer or an administrator (mode-adaptive navigation); what pages the user has visited previously (history-sensitive navigation).

Navigation is a very important part of a web application. It is central to the usability of the application, since it defines the sequence of pages a user sees. It is critical for security, since it controls which users have access to pages, and under what circumstances. Finally, it is a fundamental part of the

application's correctness; for example, a broken link is an application error.

However, navigation is usually described informally in a diagram such as Figure 1. Boxes are web pages, and arrows are navigation links between pages, with text describing the circumstances under which the navigation link is taken. From a diagram like this we can see that when a user is at the Home page, is logged on, and wishes to view their account, they next see the Account page. But it is problematic to determine whether the user must always be logged on in order to view the Account page. The link from the Home page explicitly states this, but the link from the Order page does not, so we must examine all links that reach the Order page. Clearly, as the number of pages and links grows, this becomes problematic with an informal diagram.

On the other hand, there are some web engineering approaches that include formal navigation models, but none of these are capable of describing adaptive navigation. ([1], [12])

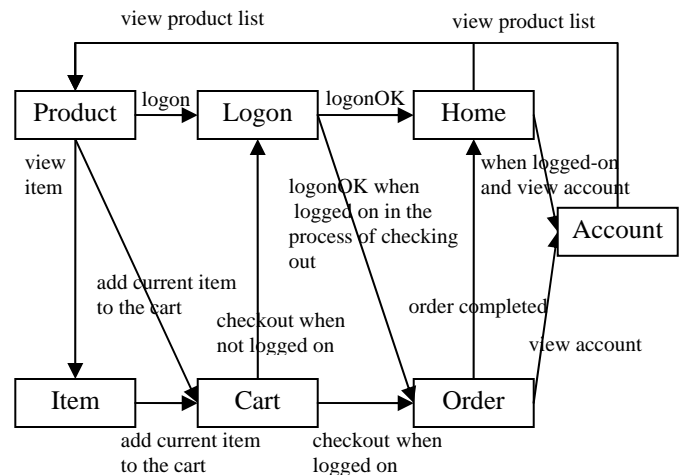


Figure 1 Typical Description of Navigation

What is needed is a formal navigation model that supports adaptive navigation, and supports automated analysis to answer questions like the one posed above.

We earlier provided this with FARNav (a Formal Approach for Rich Navigation) [7]. A FARNav model uses statecharts to precisely describe navigation, including adaptive navigation. The model is then converted to CTL [4], questions about the model are formulated as CTL rules, and the SMV model checker [3] determines which rules hold and which are violated.

A navigation model describes how the application should behave, so it is the developer’s responsibility to provide an implementation that conforms to the navigation model. With a platform such as J2EE, each navigation link first results in a request received by the server. The web application processes the request using a set of server components, then prepares a response page that is sent back to the user, which completes the navigation link. Thus for each navigation link the developer must determine which server components process it, in what order, and which response page will be returned. We call this a *request route*, the sequence of components that handle a request originating from a web page. The *request routing* is the union of all possible request routes from all web pages, so it provides implementation information for an application’s navigation.

Unfortunately, identifying request routes with a platform such as J2EE is difficult. As shown in Figure 2, each component along the request route participates in the request routing by checking and/or setting the associated URI, so this code is scattered across a number of components. In addition, although the request handling is in effect a pipeline, communication between server components is indirect, with web.xml mapping a request between components using its URI. This makes the tracing of request routes a complicated and error-prone task.

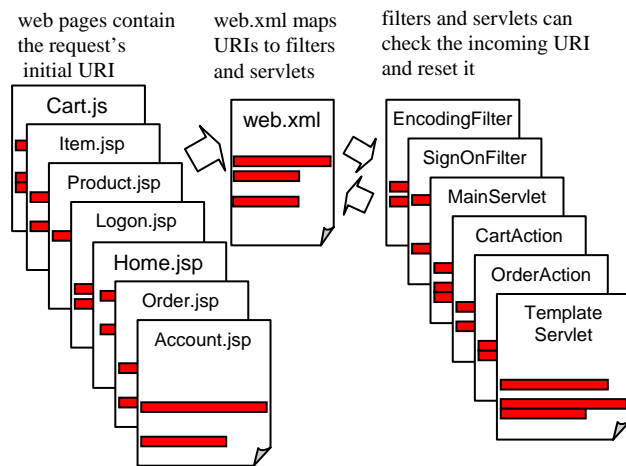


Figure 2 Tracing a Request through Server Components

To support the developer in understanding request routing, we earlier developed a formal Request Routing model to explicitly represent the request routing of a web application, and we provide operations for analyzing the request routing. [6] The model and operations are specified with Z, and Jaza is used to read in a model and perform operations on it. No prior approach explicitly represents and/or supports the tracing of request routes.

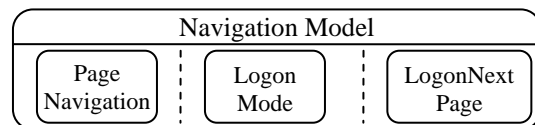
The ability to see and analyze a request routing model can help the developer determine whether the implementation conforms to the desired navigation behavior. It also helps during maintenance, where most tasks involve some tracing of request routes. Changes in the navigation design will affect the request routing and vice versa.

Thus traceability between a navigation model and a request routing is critical for both initial development and maintenance. An even bigger benefit to the developer would be to provide support for round-trip engineering from navigation model to request routing model and vice versa.

In this paper we describe the FARNav and Request Routing models (Sections 2 and 3) and present mappings between them (Section 4). Because the mappings preserve navigation semantics and are bijective, we can use them in round-trip engineering. We provide a Model Helper tool to perform the mappings on Request Routing models (Section 5). With this tool support, it is now possible to automatically determine whether a Request Routing model conforms to the navigation design, and to automatically generate a Request Routing model from a navigation model.

2. The FARNav Navigation Model

FARNav uses Statecharts with parallel state machines to model navigation. The main state machine is the Page Navigation state machine, which contains one state per web page, with navigation links represented by transitions between these pages. The other state machines are mode state machines, one for each mode, and one for each case of history-sensitive navigation. For the example in Figure 1, there are three state machines, as shown below.



Page Navigation transitions may use information about the state of the modes as a guard, and may send events to zero or more mode state machines in order to

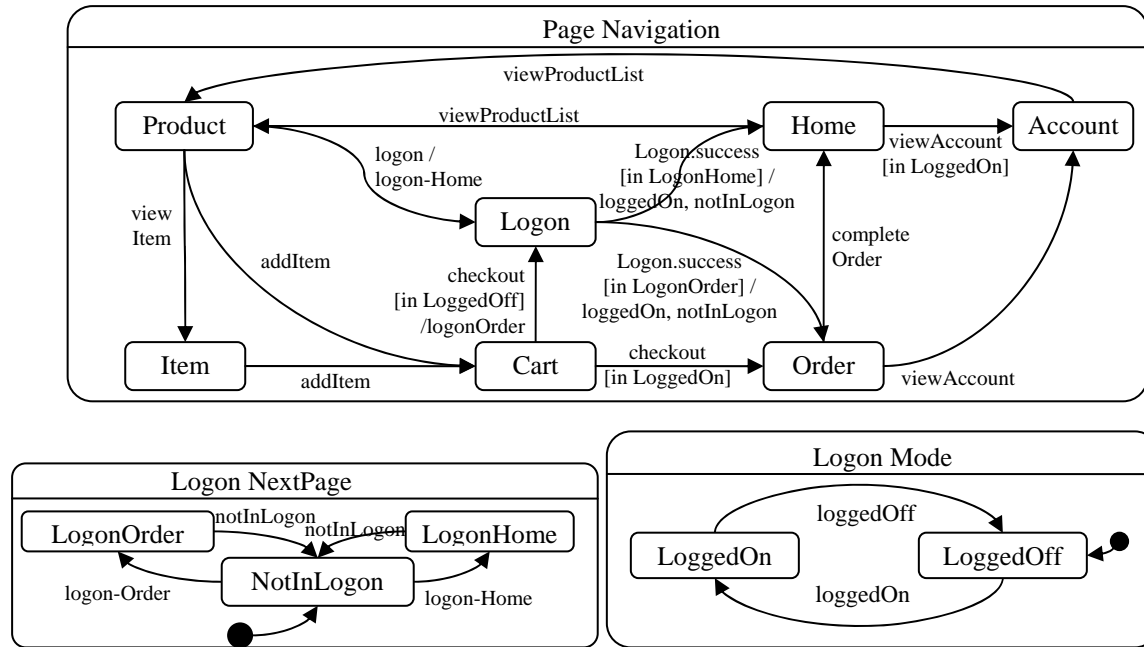


Figure 3 FARNav model for the example application

change their states. Thus the mode state machines control which Page Navigation transition fires, and these transitions in turn cause a mode to change state.

The FARNav model for the example in Figure 1 is shown in Figure 3. In general, the state machine for Page Navigation contains the following:

- **state:** a web page
- **transition:** a navigation link; format is 'event [guard] / action'
- **event:** a user action; format is 'eTag.eRes', where eTag is the user action and eRes is the result of processing this action (e.g. logon.success)
- **guard:** navigation link applies only for these mode values; format is 'in gs₀, in gs₁, ..., in gs_i' where gs is a state in a mode state machine.
- **action:** navigation link causes the mode in a parallel substate to change its value; format is 'a₀, a₁, ..., a_i' where a is a transition in a mode state machine.

For a mode state machine representing mode-adaptive navigation, there is one state for each possible value of the mode. (Logon Mode in Figure 3.) For history-sensitive navigation, there is one state for when previously visited pages are not relevant, and one state for each case where a previously visited page must be remembered. (LogonNextPage in Figure 3.)

Transitions in mode state machines are usually triggered with events fired in the Page Navigation state machine, or with timing events such as logon timeout. With the exception of timing events, a transition label

is the same as the name of the state it goes to, but starting with a lower-case letter.

All state names must be unique, across all state machines in the model. Currently we use a tabular format to represent a navigation model.

Although a navigation model such as that in Figure 3 precisely describes the navigation, it can still be difficult to check properties of the navigation. For example, it is still not obvious that the Account page cannot be reached unless the user is logged on. To check this, we can write the following rule in CTL (using a template we provide):

```
AG ( (page = Account) -> (loginStatus = LoggedOn))
```

The result is true, so this confirms that the Account page cannot be reached unless the user is logged on.

3. The Request Routing Model

The Request Routing model describes a set of nodes that represent server components, which can be web pages, filters, or servlets. Each node has associated entry and exit ports. A port has a URI, so an entry port on a node indicates that the node receives requests with the URI of the entry port, and an exit port on a node indicates that the node forwards requests with the URI of the exit port.

An exit port of one node is connected to an entry port of another node when their URIs are the same (a between-node connection). Within a node, an entry port can be connected to an exit port (an in-node connection), and the in-node connection may have an

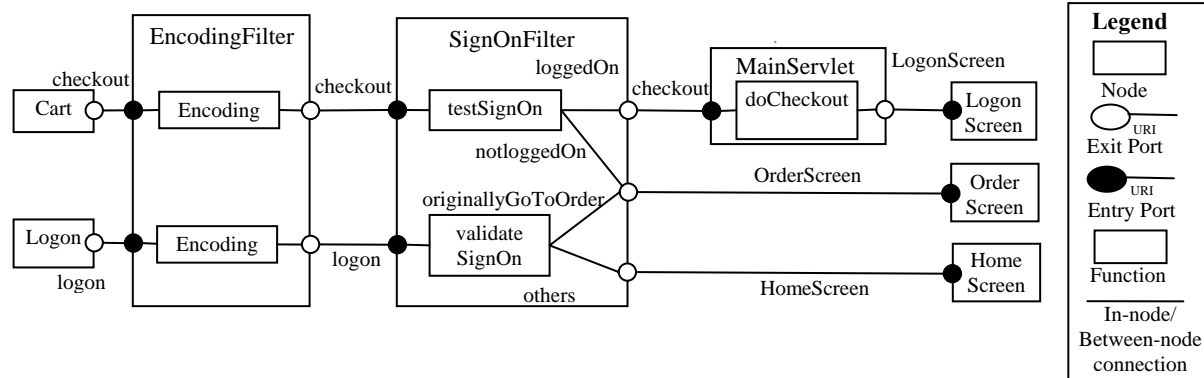


Figure 4 The Request Routing Model for Example Application

associated function that processes requests arriving with the URI of the entry port. A function can be associated with multiple in-node connections of a node, and in this case each in-node connection has a different return value and exit port, or has a different entry port.

The model is represented by a file containing a list of model elements and their values; these are input into a Z specification to populate the model. Then the specified Z operations can be executed to analyze the model.

In this paper we use diagrams to show request routing models rather than showing their underlying representation. The Table 1 summarizes the notation for these diagrams, and Figure 4 shows the Request Routing model for the example from Section 1.

Now we can more precisely define a request route to be a path through a sequence of ports (ExitP₀, EntryP₁, ExitP₁, EntryP₂, ExitP₂, ..., EntryP_i, ExitP_i, ... EntryP_{n-1}, ExitP_{n-1}, EntryP_n) where:

- EntryP_i (0 < i ≤ n) are entry ports;
- ExitP_i (0 ≤ i < n) are exit ports;
- ExitP₀ is attached to a request web page node;
- EntryP_n is attached to a response web page node;
- EntryP_i and ExitP_i (0 < i < n) are attached to the same node (Node_i) and are in-node connected;
- The in-node connection between EntryP_i and ExitP_i (0 < i < n) may have an associated function (Func_i) and return value (FuncReturn_i).
- ExitP_i and EntryP_{i+1} (0 ≤ i < n) are attached to different nodes, and are between-node connected, thus the URI of both is URI_i.

If two request routes go through exactly the same set of ports, they must be the same request route, so the sequence of ports uniquely identifies a request route.

4. Relating the Models

The FARNav and Request Routing models both describe navigation behavior, so they are related. A transition between two states in the Page Navigation state machine of the FARNav model corresponds to a request route in the request routing model.

We wish not just to establish traceability between the models but to support round-trip engineering. With round-trip engineering the task of keeping the models synchronized is greatly simplified: if a developer changes the FARNav model, then the Request Routing model is generated or adjusted accordingly. Similarly, if the developer changes the Request Routing model, the FARNav model can be extracted from the Request Routing model.

We provide mappings to build a Request Routing model from a FARNav model (RRtoFAR) and vice versa (FARtoRR). We then show that the mappings are bijective and that they preserve the navigation-related behavior. Both of these properties are necessary for the mappings to be used in round-trip engineering. We do this first for the core part of the mappings, then extend the mappings to handle transformation operations on a Request Routing model.

Property 1: Correctness. The mapping result (a FARNav model or a Request Routing model) is semantically equivalent to the mapping source (a Request Routing model or a FARNav model) for the navigation-related features.

Property 2: Bijection. Let N be a FARNav model and R be a Request Routing model,

$$(\forall N, \text{RRtoFAR}(\text{FARtoRR}(N)) = N) \\ \wedge (\forall R, \text{FARtoRR}(\text{RRtoFAR}(R)) = R).$$

4.1 The Core Mappings

The general idea of the core mapping from navigation to request routing is as follows. After leaving the request web page node, each request route will go through a ProcessRequest node. This is where

the developer can provide code to evaluate the request, so a function is assigned to each request route. The function may have return values that split requests along two or more routes. An example is a login function that returns either ‘success’ or ‘failure’.

At the other end of the request routes, the last node before the response page node is a PrepareResponse node. Again a function is assigned so that a developer can provide code to prepare the response page.

The nodes between ProcessRequest and PrepareResponse are created to handle the adaptive navigation. They check mode values and route a request accordingly, then set new mode values as required.

To describe the core mappings we represent the models in a tabular format. For a FARNav model, each row represents a Page Navigation transition and describes its transit-from state, event, guard, action and transit-to state (Tables 2 and 5). For a Request Routing model, we use one request route per table. Each row in the table describes how the request route travels through a node. So between-node connections are between adjacent rows (Tables 3 and 4).

Next we describe how the transition in Table 2 is mapped to the rows in Table 3.

Row 1: The transit-from state pageA is mapped to a node named pageA, with sequence number “1”. This node has an exit port ExitP₀ with URI pageA_eTag.

Row 2: This row uses the ProcessRequest node, which is numbered 10. The entry URI is the previous row’s exit URI, and the exit URI uses the same value with ‘_ER_eRes’ concatenated on the end. The function is named ‘checkeTag’, and the function value uses the event response: ‘_ER_eRes’.

Rows 3 through 2+i, where i is the number of guard value pairs: For guard value pair GM=gs, a row uses the node named CheckGM, and node number computed using that mode state machine’s sequence number. The entry URI is the previous row’s exit URI, and the exit URI concatenates on _C_gs. The function is named ‘checkGM’ and its value is ‘_C_gs’.

Rows 3+i through 2+i+j, where j is the number of action value pairs: For action value pair AM=ts, a row uses the node named SetAM, and node number computed using that mode state machine’s sequence number. The entry URI is the previous row’s exit URI, and the exit URI concatenates on _S_ts. The function is named ‘setAM’ and its value is ‘_S_ts’.

Row 3+i+j: This row uses the PrepareResponse node, which is numbered 9990. The entry URI is the previous row’s exit URI, and the exit URI is the transit-to state pageB plus “Screen”.

Row 4+i+j: The transit-to state pageB is mapped to a node named pageBScreen, with sequence number

“10000”. This node has an entry port ExitP_{3+i+j} with URI pageBScreen.

There is problem with assigning an arbitrary ordering for the mode state machines. This arises when a state in the Page Navigation state machine has two or more outgoing transitions. If each of the outgoing transition guards checks a different mode state machine, then the ordering of these checks does not matter. If each guard checks exactly the same set of mode state machines, again the ordering does not matter. But for the third case, where the guards check different but overlapping sets of mode state machines, the ordering is critical: the first mode checked must be the shared one. Because different sets of transitions could impose conflicting restrictions on the ordering of mode state machines, we instead “normalize” transitions for the third case, by making each transition from the same state check exactly the same set of mode state machines. Thus a transition that originally did not check a mode must be split into several transitions, one for each value of the added mode.

The Function Name column of Table 2 contains modes that this function checks or sets, and we define a set of check-tags and a set of set-tags to specify these mode state machine and states. The tags were not originally part of the request routing model since they are needed only for maintaining traceability with the navigation model.

The tags are kept in a table format. A function may contain multiple check-tags and set-tags. Each tag contains an argument and a set of possible values.

The reverse mapping (from Table 3 to Table 4) is simpler. The first row of Table 4 maps to the transit-from state and the event tag. Then the check values of the middle rows are combined and mapped to the event response and guard of the transition. Similarly the set values of these rows are combined and mapped to the action. The last row maps to the transit-to state.

The core mapping from a FARNav model to a Request Routing model satisfies Property 1 because according to the mapping steps provided earlier, each page navigation transition is mapped to a request route. It starts from an exit port of a node representing the transit-from state and the event tag; goes through functions checking a mode with a state as return value as all guard value pairs; goes through functions setting a mode with a state as return value as all action value pairs; ends at an entry port of a node representing the transit-to state. Thus it is not possible for a page navigation transition to have no corresponding request route.

Table 1 Page Navigation Transitions

Transit-From	Event	Guard	Action	Transit-to
pageA	eTag.eRes	$GM_0 = gs_0, GM_1 = gs_1, \dots, GM_i = gs_i$	$a_0(AM_0=ts_0), a_1(AM_1=ts_1), \dots, a_i(AM_i=ts_i)$	pageB

Table 2 Mapped Request Routes

Entry Port	Exit Port	Node Name (seq. num.)	Entry URI	Exit URI	Function Name	Func Ret. Val.	
-	ExitP ₀	pageA (1)	-	pageA_eTag	-	-	
EntryP ₁	ExitP ₁	ProcessRequest (10)	pageA_eTag	pageA_eTag_ER_eRes	check eTag	_ER_eRes	
$0 < k < i$	EntryP _{1+k}	ExitP _{1+k}	CheckGM _k (20 + i*10, when M _i = GM _k)	pageA_eTag_ER_eRes_C_gs ₀ _C_gs ₁ ..._C_gs _{i-1}	pageA_eTag_ER_eRes_C_gs ₀ _C_gs ₁ ..._C_gs _{i-1} _C_gs _i	check GM _k	_C_gs _k
$0 < k < j$	EntryP _{1+i+k}	ExitP _{1+i+k}	SetAM _k (30 + m*10 + i*10, when S _i = AM _k)	pageA_eTag_ER_eRes_C_gs ₀ _C_gs ₁ ..._C_gs _{i-1} _C_gs _i _S_ts ₀ _S_ts ₁ ..._S_ts _{k-1}	pageA_eTag_ER_eRes_C_gs ₀ _C_gs ₁ ..._C_gs _{i-1} _C_gs _i _S_ts ₀ _S_ts ₁ ..._S_ts _{k-1} _S_ts _k	setAM _k	_S_ts _k
EntryP _{2+i+j}	ExitP _{2+i+j}	Prepare Response (9990)	pageA_eTag_ER_eRes_C_gs ₀ _C_gs ₁ ..._C_gs _{i-1} _C_gs _i _S_ts ₀ _S_ts ₁ ..._S_ts _{k-1} _S_ts _k	pageBScreen	pageA_eTag_ER_eRes_C_gs ₀ _C_gs ₁ ..._C_gs _{i-1} _C_gs _i _S_ts ₀ _S_ts ₁ ..._S_ts _{k-1} _S_ts _k _proRes	-	
EntryP _{3+i+j}	-	pageB Screen (10000)	pageBScreen	-	-	-	

Table 3 A General Request Route

Entry Port	Exit Port	Node Name	Entry URI	Exit URI	Function Name	Func. Ret.Val.	
/	ExitP ₀	Node ₀	/	Node ₀ _Event	/	/	
$0 < i < n$	EntryP _i	ExitP _i	Node _i	URI _{i-1}	URI _i	Func _i [checkTag arg.:event"] ⁺ [checkTag arg.: GM _{i0} , GM _{i1} , ..., GM _{ik}] ⁺ [setTag arg.: AM _{i0} , AM _{i1} , ..., AM _{ij}] ⁺	[_ER_erV] ⁺ [_C_cV _{i0} _C_cV _{i1} ... _C_cV _{ik}] ⁺ [_S_sV _{i0} _S_sV _{i1} ... _S_sV _{ij}] ⁺
EntryP _n	/	Node _n	URI _{n-1}	/	/	/	

Table 4 Mapping Result Page Navigation Transition

Transit-From	Event	Guard	Action	Transit-to
Node ₀	Event. erV	$\prod_{i=1}^{n-1} (\prod_{j=0}^k GM_{ij} = cV_{ij}) cV_{ij} \neq \text{"ANY"}$	$\prod_{i=1}^{n-1} (\prod_{j=0}^k tosV_{ij} (AM_{ij} = sV_{ij})) sV_{ij} \neq \text{"NOT"}$	Node _n

Also it is not possible to have a request route with no corresponding page navigation transition. The reason is the request routing model is the combination of all “correct” request routes that are mapped from the page navigation transitions. Because no two request routes can join before the exit port of PrepareResponse, it is not possible to have extra request routes that partially follow the route of a

“correct” request route. Thus each request route maps to a page navigation transition in the FARNav model.

A similar argument applies when showing that the core mapping from a Request Routing model to a FARNav model satisfies Property 1.

Since both core mappings satisfy Property 1, it is obvious that Property 2 holds: or both core mappings, the number of request routes and the number of page navigation transitions is the same and there is 1 to 1

mapping. So to applying the inverse core mapping gives the original source model.

4.2 FARtoRR and RRtoFAR

The Request Routing model generated using the core mappings can be directly used to guide implementation. However, a developer may want to refine the generated Request Routing model before implementation. We provide two model transformation algorithms: node reordering and node combining. Although we do not have space to explain it here, after applying any sequence of these operations, the resulting Request Routing model still maps to the same FARNav model as the original.

Node reordering changes a node's sequence number and updates the request routing model. Node reordering may be used to reduce the complexity of the request routing model or may be used before combining two nodes since only adjacent nodes can be combined. A developer may want to combine adjacent nodes together, for example when using the front controller design pattern, which allows just one servlet.

The mappings CoreFARtoRR and CoreRRtoFAR are the core mappings. However, if transformation operations have been applied to the request routing model, there are now two request routing models that map to the same navigation model, and we lose the bijective property. Thus the transformation operations must be carried along by both models. The complete mappings FARtoRR and RRtoFAR are:

$$\text{FARtoRR}(N, \text{ops}) = \text{executeOps}(\text{ops}, \text{CoreFARtoRR}(N))$$

$$\text{RRtoFAR}(R, \text{ops}) = \text{attachOps}(\text{ops}, \text{CoreRRtoFAR}(R))$$

Although the operations are included in the FARNav model, they are treated as a comment. Thus FARtoRR and RRtoFAR satisfies Property 1 because the core mappings satisfy Property 1 and neither attaching nor applying the operations affects the "correctness" of the result model.

The intuition behind our proof that Property 2 holds for both mappings is as follows. If R is the result of executing a sequence of operations on the core request routing model coreR, then applying RRtoFAR followed by FARtoRR regenerates the original request routing model and reapplies the sequence of operations, so the result is the same R. Similarly, if we start with the FARNav model N, convert it to coreR then to R, we have applied only the operations in N, and these are carried along when converting back to the FARNav model.

5. Tool Support

We developed a Model Helper tool that implements the mappings and transformation operations described in Section 4. For the forward engineering task of the round-trip engineering, Model Helper generates a request routing model including the function tags from a FARNav model. If there are existing transformation operations (OPS), these are applied on the generated request routing model, in the order specified.

Then a developer can use the tool to transform the request routing model by reordering the nodes or combining the nodes. A record of the operations applied is added to OPS. Then the request routing model can be used in Jaza to trace and analyze request routes.

For the reverse engineering task, Model Helper uses the request routing model with embedded function tags and saved OPS in order to generate a FARNav model. The function tags are used to recreate the mode state machines, but the OPS are simply stored as a comment in the FARNav model. Then the developer can check the navigation design using the SMV tool.

We applied the Model Helper in a number of case studies using existing web applications. The goal was to test the Model Helper tool and the round-trip engineering process. Next we give an overview of one of these case studies. A portion of the FARNav model for the PetStore [15] application appears in Figure 3. This was created by observing the application behavior. Figure 4 shows the corresponding Request Routing model, which was created by examining the source code.

Starting from the FARNav model in Figure 3, we applied Model Helper to generate the Request Routing model shown in Figure 8. The port URIs generated are quite long, because they record the event tag, event response, and all check and set values used in the adaptive navigation.

Next we applied some transformation operations, in order to reach a request routing model like the one in Figure 4, which reflects the current implementation. We applied the node combining operation four times to combine nodes ProcessRequest, CheckLogonNextPage, CheckLogonStatus, SetLogonNextPage and SetLogonStatus. This combines the processing of requests and the handling of adaptive navigation into one node. Model Helper also records the operations in OPS. The result is shown in Figure 9.

The structure of the model is now close to our target, and we used Model Helper again to confirm that the new model conforms to the FARNav model, and that it can be regenerated from the FARNav model. However, node names, function names, and URIs are meaningful but quite lengthy, and not all functions are needed, so the next step is to edit the model manually:

- rename the RequestProcess... node to SignOnFilter;
- rename the functions in the SignOnFilter node to “testSignOn” and “validateSignOn” (also in the function tag table);
- rename the Cart_checkout_ER_LoggedOn_proRes function to “doCheckout”;
- remove all other functions in the PrepareResponse node, and their related entry and exit ports (since no processing is needed for these response pages);
- rename the PrepareResponse node to MainServlet;
- rename the events and URIs with simpler names.

The last change is to add a new Encoding node and encoding function right after the request web page nodes.

The resulting request routing model looks exactly like Figure 4, except for the function return values, which still have their generated long form. Changing these is currently not allowed since that would affect the regeneration of the FARNav model.

These changes are not currently supported as transformation operations. We can still use the Model Helper to generate the corresponding FARNav model, but the manual changes are not saved as OPS, so they are lost if we then regenerate the request routing model.

After applying the manual changes to the request routing model, we used it to regenerate a FARNav model, then compared that to the original and saw that it does match. So the developer is assured that the manual changes do not affect the navigation design.

6. Related Work

The mappings we provide in Section 4 are model transformations (using terminology from model-driven software development). Model transformation is at the core of model-driven software development [13] and it is central to round-trip engineering.

As Henriksson described in [7], for round-trip engineering it is necessary for model transformations to have the two properties we state in Section 4: correctness and bijection. However, examining whether these properties hold is not commonly done for model transformations (for example, [11]).

While there are many navigation models among the existing web engineering models, many are simply descriptive models. There are however several navigation models that can be used to generate an implementation. Fraternali et. al use HDM-lite notation to specify structure, navigation and presentation of web application semantics, then generate database schema and web pages from the model [5]. Merialdo

et. al generate code from design artifacts [9]. Navarro et. al. use the Pipe approach for characterizing navigational maps then map to UML-Conallen class diagrams with the web pages [10]. Book et. al use the Dialog Flow Notation to represent the dialog flow within a web application, then translate this into an object-oriented dialog flow model for run-time lookups [1]. Vilain et. al created the User Interaction Diagram (UID) as a basis for generating a preliminary class diagram using heuristic guidelines. [15]

However, most of these approaches simply map to web pages (except for [1] and [15]). None support adaptive navigation nor explicitly show how the navigation impacts the request routing. None support traceability between navigation and implementation, nor do they provide an inverse mapping back to the navigation model.

7. Discussion and Conclusion

A FARNav model and a Request Routing model describe different features of a web application and should be used at different stages of development and maintenance. The FARNav model represents the navigation design so it is related with user requirements and high-level design. The request routing model represents the request routes in a web application, so it is closer to the implementation: the elements in a request routing model directly map to the source code elements.

Earlier we developed these models along with operations that automatically analyze properties of the models. While each is useful independently, their utility is greatly improved by relating the models. This is the contribution described in this paper. We provide mappings (model transformations) between a FARNav model and a Request Routing model and vice versa. We show that the mappings are correct, meaning that they preserve navigation behavior, and that they are bijective.

Because the mappings have these properties, they support traceability between the models, and they allow the models to be used in round-trip engineering. We also gained practical benefit by examining whether the mappings had these properties. We started by writing algorithms for the model transformations in pseudo-code, then discovered errors after explicitly addressing these properties for the mappings.

These mappings are implemented in a new Model Helper tool. Starting from a FARNav model, the developer can generate an initial Request Routing model. We also provide operations for reordering and combining nodes in a Request Routing model, which

can be used to simplify the generated model while preserving the mapping properties. Thus the Model Helper saves effort for the developer by generating the Request Routing model, and by ensuring that the two models are consistent.

We describe one of a number of case studies we have done, using the Model Helper in a round-trip engineering scenario.

Although we had a general idea of how we would relate the two models as we developed each, the models have changed as a result of defining the mappings between them. One was refining the format of the events on transitions in the Page Navigation state machine. Another was a change in the way adaptive navigation is exhibited in the Request Routing model. An earlier version had a special notation for adaptive navigation, but now we generate an assembly of nodes containing check and set functions to handle the adaptive navigation. We had to add function tag tables to the Request Routing model in order to be able to recreate the details of the mode state machines. Finally, we had to carry the Request Routing model's transformation operations along in the FARNav model, in order to reapply those operations to a generated Request Routing model.

The next step for the mappings and Model Helper tool is to define additional transformation operations for a Request Routing model. Once we have a richer set of operations, we can investigate the use of heuristics to generate more customized Request Routing models, for example when the developer specifies that the front controller pattern should be used.

8. References

- [1] Alfaro, L. de. Model Checking the World Wide Web. *Computer Aided Verification*, Lecture Notes in Computer Science, vol 2102, Springer-Verlag, 2001, 337-349.
- [2] 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.
- [3] Cadence Berkeley Labs. SMV model checking system.
- [4] Computational Tree Logic (CTL)
http://en.wikipedia.org/wiki/Computational_tree_logic
- [5] Fraternali, P. and Paolini, P. Model-driven development of Web Applications: The AutoWeb system. *ACT Trans. Inf. Syst.* 18, 4 (Oct 2000), 323-382.
- [6] M. Han, and C. Hofmeister. Modeling Request Routing in Web Applications. To Appear, *Proceedings of the 8th IEEE International Symposium on Web Site Evolution (WSE 2006)*.
- [7] M. Han, and C. Hofmeister. Modeling and Verification of Adaptive Navigation in Web Applications. In *Proceedings of the 6th International Conference on Web Engineering (California, USA, July 11 - 14, 2006)*, 329-336.
- [8] Henriksson, A., Larsson, H. A Definition of Round-trip Engineering. Technical report, Linkopings Universitet, Sweden, 2003.
- [9] 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.
- [10] Navarro, J.L. Sierra, A. Fernandez-Valmayor and B. Fernández-Manjón. Conceptualization of Navigational Maps for Web Applications. In *Proceedings of Workshop on Model-driven Web Engineering (MDWE2005)*. July 26, 2005
- [11] Nickel, U. A., Niere, J., Wadasck, J. P., Zündorf, A. Roundtrip Engineering with FUJABA. In *Proceedings of 2nd Workshop on Software-Reengineering (WSR)*, Fachberichte Informatik, Universität Koblenz-Landau, August 2000.
- [12] Ricca, F. and Tonella, P. Analysis and testing of web applications. In *Proc. of ICSE 2001*, International Conference on Software Engineering, Toronto, Ontario, Canada, (May 2001), 25-34.
- [13] Sendall, S., Kozaczynski, W. Model Transformation: The Heart and Soul of Model-Driven Software Development. *IEEE Software*, vol. 20, no. 5, September/October 2003, 42-45.
- [14] Sendall, S., Küster, J. Taming Model Round-Trip Engineering. In *Proceedings of Workshop on Best Practices for Model-Driven Software Development*, (2004).
- [15] Sun Java Center. Java Petstore 1.3.01
http://java.sun.com/developer/releases/petstore/petstore1_3_01.html
- [16] Vilain, P.; Schwabe, D. Improving the Web Application Design Process with UIDs. In *Proceedings of the 2nd International Workshop on Web-Oriented Software Technology* (2002).

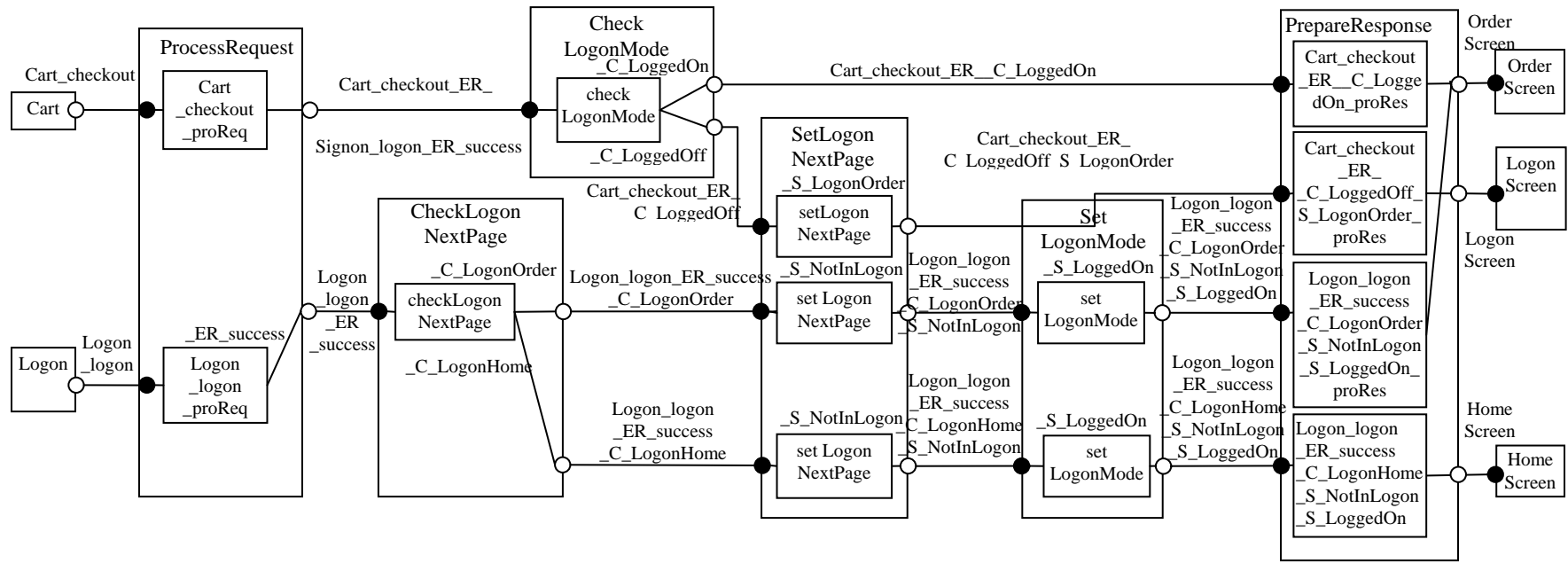


Figure 5 Generated Request Routing Model for Example Application

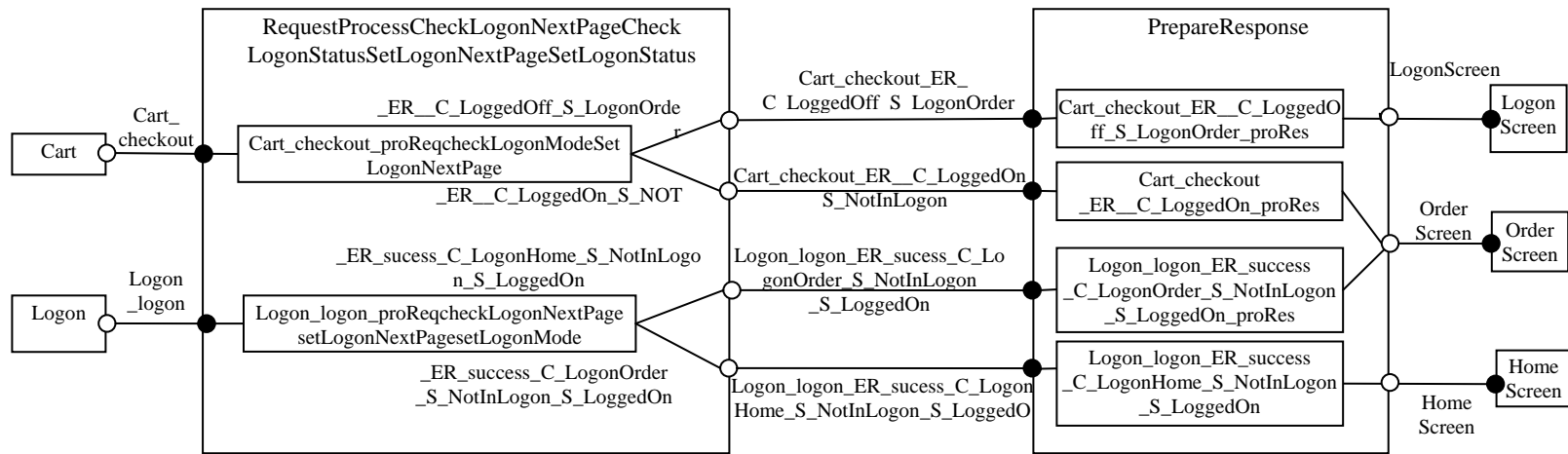


Figure 6 The Request Routing Model after Node Combining

DRAFT