DESIGNING CLIENT VIEW NAVIGATIONS USING REST STYLE SERVICE PATTERNS

Eunjung Lee and Kyung-Jin Seo
Computer Science Department, Kyonggi University, San 94, Suwon, South Korea

Keywords: View Navigations, REST Service Pattern, Model based Development, XForms.

Abstract: This paper considers an approach to the development of view navigations in a REST client page. When a page interfaces multiple service methods, it needs to maintain multiple views, along with local data. For this reason, it is necessary to develop navigational codes between views and service requests. The contributions of this paper are as follows: First, we discussed a formal approach for using REST service method patterns in order to design client page views and navigations. Second, we presented type conditions for possible method calls and view moves. In addition, we introduced a design model to help developers to describe the relations between views and resources on an abstract level. Finally, we presented a prototype implementation for navigational code generation using XForms pages, applying the proposed approach and standard patterns.

1 INTRODUCTION

As RESTful web services have become more popular, the number of client pages for RESTful service grows rapidly. Currently, a web page supports remote services by allowing users to send requests and to handle the returned results. Therefore, web pages are able to provide client-side service compositions, often called “mashup”, by connecting the result of a request to an input of another method call.

REST (Representational State Transfer) is the name of a client-server architecture proposed by R. Fielding (Fielding, 2000), which refers application state transitions by requests. On the client side, the application states transfer by events, such as moving to another view, calling a method, or receiving a response. Therefore, view navigations are an important part of designing client side mashups.

There have been studies on view navigations as model-driven web application development (Gordillo et al., 2006, Guell et al. 2000). Many approaches have been proposed to determine an appropriate design for a set of navigations in terms of user concerns, contexts and/or application requirements.

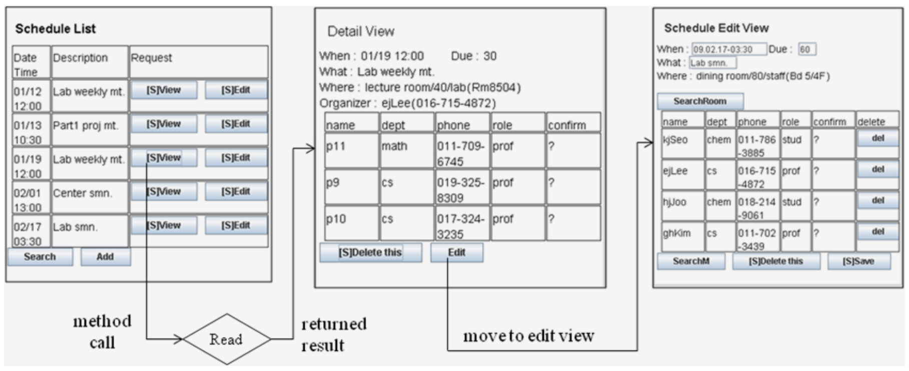
*This work was partly supported by the Gyeonggi Regional Research Center.

However, there has been less research on designing navigations inside a page.

In this paper, our goal is to present a model-driven development technique for generating view navigational codes inside a client page. The main idea is to apply REST service standards and patterns to designing client pages and view navigations. REST is a conceptual architecture, and is a rather informal terminology. However, since it is practically useful and accepted almost as a standard, we use the term ‘standard’ when describing the related conventions and style suggestions (Richardson, 2007, Vosloo, 2008). Moreover, it is very useful for the design of web applications that REST services have well-defined methods and standard data types. For a given set of service methods, we present rules and conditions to identify possible and necessary navigations.

When a client page includes a number of services for more than one resource, the number of potential navigations gets quite large. To aid the navigation design in such pages, we propose a design model called a resource interface diagram (RID). RID accepts REST service patterns as a main consideration. This diagram is useful for abstracting the methods and views, and allows designers to concentrate on navigational needs.

Copyright © 2010 by SCITEPRESS – Science and Technology Publications, Lda. All rights reserved



Schedule List

Date Time	Description	Request
01/12 12:00	Lab weekly mt.	
01/13 10:30	Part1 proj mt.	
01/19 12:00	Lab weekly mt.	
02/01 13:00	Center smn.	
02/17 03:30	Lab smn.	

Detail View

When : 01/19 12:00 Due : 30
What : Lab weekly mt.
Where : lecture room/40/lab(Rm8504)
Organizer : ejLee(016-715-4872)

name	dept	phone	role	confirm
p11	math	011-709-6745	prof	?
p9	cs	019-325-8309	prof	?
p10	cs	017-324-3235	prof	?

Schedule Edit View

When : 09.02.17-03:30 Due : 60
What : Lab smn.
Where : dining room/80/staff(Bd 5/4F)

name	dept	phone	role	confirm	delete
kjSeo	chem	011-786-3885	stud	?	del
ejLee	cs	016-715-4872	prof	?	del
hjJoo	chem	018-214-9061	stud	?	del
ghKim	cs	011-702-3439	prof	?	del

(a) Search result view (b) read output view (c) Update input view.

Figure 1: Example of input and output views with navigations.

To generate navigational codes, we apply policies based on REST patterns and conventions, represented in RID. We have implemented the proposed approach to generate XForms pages, adopting WADL as a specification language for the services provided. The generated pages and implemented scenario services validate the proposed approach by showing a reasonable level of interface quality.

This paper is organized as follows. In section 2, we introduced the client page model, including REST services and service specifications. In section 3, we presented the navigation design method. Section 4 briefly introduces the implementation of the approach, and section 5 includes related work of the result. Section 6 concludes the paper.

2 CLIENT SYSTEM MODEL

In this section, we introduce the client system model. We are interested in views and navigations on a mashup client page. Also, a set of service methods connected from a page is defined with a formal model. Also, scenario services and use cases are introduced.

2.1 Active Client Page

A web page may send and receive messages, often connecting to more than one server. RESTful services and client script technologies have become popular for the development of such pages. Views of the page allow users to input parameter data and to check the returned results of service requests (May, 2005 and Lee, 2008). Such pages include local data and action codes. In this paper, we called a client.

In the first step of developing an active client page, a set of service methods should be defined as a specification. A set of views is then determined for preparing input parameters, and another set of views is determined for checking the returned results. The set of necessary views is the starting point of designing the user interface. Figure 1 shows examples of input and result views.

When a page has more than one view, we need to consider the layout of views. Usually, layout is dependent on the platform; on a desktop computer, multiple views are visible at the same time by sharing the screen area, while on a mobile platform, only one view is shown at a time because of the small screen size. We do not consider the layout problem for the simplicity, assuming the small screen.

When a page includes more than one view, a navigation scheme is necessary for users to be able to access all operations and service methods. As shown in Figure 1, navigation control is either a method call (a to b) or a view move (b to c). We call those operations *view navigation operations*. For navigation operations, some of the view data is used as input parameters of a method call, or is transferred to the target view. For example, in Figure 1(a), the selected schedule id is used as an input parameter of the read method request. In Figure 1(b), schedule data is transferred to the update view of Figure 1(c) as initial data.

In today's computing environment, location-based or timely services are determined dynamically. In order to assemble client pages that can connect to a dynamic set of services, we need to generate client pages at the moment of service access. Although there are well-established code-generation techniques available for the presentation and communication of web pages, view navigations in a page are still addressed less in terms of model-with navigations driven development.

It is challenging to add view navigation operations on top of presentation code when the page interfaces multiple service methods. When the number of interface methods grows, the number of possible navigation relations grows rapidly. On the other hand, users should be able to access every method of the page. Therefore, selecting the necessary navigation controls to be added is a complex task and an important aspect of the quality of the user experience.

In the next section, we present a method of choosing a proper set of navigation operations.

2.2 Typed REST Service Specification

While most web service specifications are for publishing services by providers, a set of service methods for a client page serves the needs for client page designs. For example, when a gas station provides gas and drink ordering, auto services, and a timely service for a customer survey, a user may select the gas ordering, drink ordering, and survey services to use. A page then must be generated to assemble those selected services before it is downloaded to the user's device. This section discusses an approach to specifying services for a page, assuming that the services are in a REST style.

The REST service standard describes well-defined data types and service method patterns (Richardson, 2007). REST web services have a url (universal resource location) for each resource type, and an identifier for each instance of the resource. For example, Schedule resource and its instances are denoted as .../Schedules, and ../Schedules/{id}. On the other hand, REST web services use HTTP method types, which are O_{SCRUD} = *{Search, Read, Update, Create, Delete}*. These methods have well-defined input and output data types and semantics.

In addition, we assume that all service methods have a predefined schema type for input parameters and the result data. This is called *Typed REST Service*. Table 1 shows standard types of REST service methods for a resource type *r*. The identifier of a resource instance is denoted as *id(r)*. Search methods require more consideration of diverse input parameter types, which usually include ids or keyword strings. For example, searching of a schedule resource by organizer, date, and keyword is possible. Moreover, the result type of a search method is *list(r)*, which is a list of pairs of an id and a simple data.

Table 1: REST style standard methods.

O_{scrud}	Method type	Input parameter type	Result type
Search	GET	Search parameters	list(r)
Read	GET	id(r)	r
Update	PUT	id(r), r	
Create	POST	r	
Delete	DELETE	id(r)	

Let W be the set of typed services, and T be the set of types. Also, let $R \subset T$ be the set of resources. Then, we can define some notations for a typed REST service model.

$m \in W$: web service method $m = (t, u, X, y)$,
where t : method type, $t \in O_{SCRUD}$,
u : the url of method m
X : the set of input parameter types of m, $X \subseteq T \cup \{\perp\}$
y : the result type, $y \in T \cup \{\perp\}$

In this notation, $\perp$ denotes that the service has no input (or output) data. Let T^W be the set of types to define W. Then, we can categorize T^W as follows:

$$T^W = R \cup ID(R) \cup T^W_{Search}\ U\ L(R),$$

where $ID(R) = \{id(r) \mid r \in R\}$ and $L(R) = \{list(r) \mid r \in R,$ where *list(r)* is the list type of the resource *r}*. Also, T^W_{Search} is the set of input types for search methods.

In this paper, we use the notation $\triangleright$ to denote a type inclusion relation. We use the ancestor-descendent relationships for type inclusions for the sake of simplicity. Therefore, if type t_1 is an ancestor of t_2, then $t_1 \triangleright t_2$. To extend this to a set X, if there is $t_2 \in X$ such that $t \triangleright t_2$, then t $\triangleright$ X. This is for the case when *X* is a disjunction of possible types. Other relationships can be used such as terminal data type inclusions.

The set of selected service methods for the client is denoted as $C \subset T$. Furthermore, T^C is a set of types referred in *C*.

2.3 Running Example and Scenario

REST services provide standard methods for resources, and resources are related to each other via fields. Relations between resources are represented by mapping fields of the contents diagram. In the running example, we have resources R = {Schedule, Room, Person}, and the content diagram is shown in Figure 2.

We use WADL (Web Application Description Language) to specify the selected set of client service methods. WADL can describe more than one resource, and contains information such as service locations, input parameter and result types. We assume the result type is either in XML for the service result, or in HTML for a new page. Every data type is defined in schema and namespaces for typed REST services. The following is part of the schedule service WADL, where xsd and schd is a namespace of the corresponding schema definitions.

```
<resource path="Schedules">

 <method name="GET" id="search_s">

   <request>

     <param name="query"

         type="xsd:string" .../>

     <param name="person_id"
        type="schd:pid" style="query"/>

     <param name="date"
        type="schd:date"
```

Table 2:Client service methods C.

Base url	Method type	Input paramer types	Result type
schedules/	Search	null, xsd:string	sch:slist
schedules/{sid}	Read	sch:sid	sch:schedule
schedules/{date}	Search	sch:date	sch:slist
schedules/{sid}	Update	sch:schedule	
schedules/	Create	sch:schedule	
people/	Search	null, xsd:string	sch:plist
people/{pid}	Read	sch:pid	sch:person
people/{pid}/schedules	Search	sch:pid	sch:slist
people/{pid}/rooms	Search	sch:pid	rm:rlist
rooms/	Search	xsd:string	rm:rlist
rooms/{rid}	Read	rm:rid	rm:room

null : ⊥

xmlns:xsd="http://www.w3c.org/test/schema.xsd"
xmlns:sch="http://miclab.kyonggi.ac.kr/RestServices /Schedules/Schedule.xsd"
xmlns:rm=http://miclab.kyonggi.ac.kr/RestServices/Rooms/Room.xsd

The set of service methods for the scenario page is shown in Table 2. It includes read and update methods for Schedule, and read method for People and Rooms. Result types and input parameter types are important for analysis of the navigational relations.

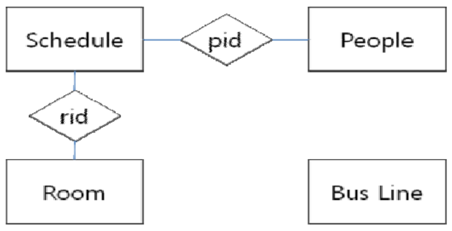


Figure 2: Content diagrams between resources of the scenario services.

From Table 2 and Figure 2, we can find that the id of Person is used as an input parameter of searches for Schedule or Room, and a pid of the Schedule resource allows the request of Person data. Connecting the result of the previous request to another method calls enables composition by users. This is an important goal of designing view navigations.

3 NAVIGATION DESIGN MODEL

In this section, we introduce view navigation design methods, the main contribution of this paper. Also, we present two diagrams to help navigation designs.

3.1 Views and View Navigations

Let C be the set of service methods for an active client page. Active client pages have input views or output views; for each method $m \in C$, they are denoted as $view_{in}(m)$ and $view_{out}(m)$. An input view is for editing or entering input parameter data, and an output view for checking the result data. To find a set of necessary views from *C*, we use simple rules: i) if the input parameter type is ⊥ or not editable, then the input view is not necessary, and ii) if there is no result data, then the output view is not necessary. Therefore, we have potential view types *VIEWS = {IS, IC, IU, OS, OR}* as shown in Table 3.

Table 3: View types for REST methods.

Method type	Input view	Input parameter types	Output view	Result type
Search	IS	T^r_{Search}	OS	*list(r)*
Read	-	*{id(r)}*	OR	*r*
Create	IC	*{r}*	-	⊥
Update	IU	*{r}*	-	⊥
Delete	-	*{id(r)}*	-	⊥

We denote a view instance by attaching the resource to a view type. For example, *IS(r)* is an input view for the search of the resource *r*. A set of all views for the resource *r* is denoted as *VIEWS(r)*. We define a view data type for a given view *v* as follows.

[Definition 1] For a method m and $v \in VIEWS(r)$, the view data type *type(v)* is defined as follows:

$type(v) = \{t \mid t \in X, m = (\tau, u, X, y), v = view_{in}(m)\},$

$= \{t \mid t = y, m = (\tau, u, X, y), v = view_{out}(m)\}.$

To invoke views, we need data prepared in advance. The precondition for the view v is defined as *pre(v)*.

[Definition 2] For the resource *r* and $v \in VIEWS(r)$, *pre(v)* is defined as follows:

$pre(v) = \{r \mid r \in R\}$, if $v = IU(r)$,

$= \{\perp\}$ if $v = IS(r)$ or $IC(r)$,

$= type(v)$ if $v = OS(r)$ or $OR(r)$.

Furthermore, we define view navigation controls for method calls and view moves.

[Definition 3] For a view *v*, a method *m* and a type *t*,

(1) A move control $c_{move} = (v, v', t)$, where $v' = view_{in}(m)$ for $m \in C$, *and t* is data transferred to v', $t \rhd pre(v)$,

(2) A method call $c_{call} = (v, m, t)$ where *t* is an input parameter type of *m*, $t \rhd type_{in}(m)$.

A method call for *m* is possible not only from the corresponding input view $view_{in}(m)$ but also from another view where the view data includes the input parameter types.

[Corollary 1] From the view v, a method call $c_{call} = (v, m, t)$ is possible if $type(v) \rhd t \in type_{in}(m)$.

(proof) When $v = view_{in}(m)$, $type(v) = type_{in}(m)$, and therefore the condition is satisfied. When the method m is called from *v*, then we need to transfer the input parameter data from *type(v)*. Therefore, $type(v) \rhd t$ should be satisfied. On the other hand, if the view *v* includes the data for the input parameter of m, it is possible to call m using the data, and bypassing the input view of *m*. □

On the other hand, the move $c_{move} = (v_1, v_2, t)$ is possible if $v_2 = view_{in}(m)$, and $type(v_1) \rhd t = pre(v)$. In this paper, a move from an output view to an input view is assumed. An output view is activated only when there is a returned result, so a move to an output view is not appropriate. Also, an input view is for preparing a method request, so it is not necessary to move between input views. However, there might be exceptions depending on application needs.

3.2 View Navigation Diagram (VND)

From discussions so far, we can compute a set of possible navigation controls. However, the above conditions for possible controls allow many navigation relations that are not crucial. For example, moving to a create input view is always possible, since $pre(IC(r)) = \varnothing$. Also, minimizing the number of view navigation controls is important for designing a better user interface. Therefore, we introduce a design model for view navigations utilizing REST patterns and the semantics of methods.

To depict the design result, we introduce a diagram that has methods, views and navigations. For a method m, we denote in(m) and out(m) as the entry point of the response result and the exit point of the request call, respectively. Then we have two sets for entry and exit points, $G_{in} = \{in(m) \mid m \in C\}$, $G_{out} = \{out(m) \mid m \in C\}$. The set of all possible navigations consists of three groups, which are as follows:

$\mathbf{M}_{move} = \{(v_1, v_2) \mid v_1 \in V_{out}, v_2 \in V_{in},$
$c_{move} = (v_1, v_2, t)$ is possible$\}$,

$\mathbf{M}_{call} = \{(v, out(m)) \mid v \in V, m \in C,$
$c_{call} = (v, m, t)$ is possible$\}$,

$\mathbf{M}_{callback} = \{(in(m), v) \mid v = view_{out}(m), m \in C\}$

For a set of views $V \subset VIEWS(R)$ and a set of methods *C*, a view navigation diagram VN*D* is defined as follows:

$VND = (V, v_{home}, G_{in}, G_{out}, M_{move}, M_{call}, \mathbf{M}_{callback})$,

where $v_{home} \in V$ is the home view of the client page, and $M_{move} \subset \mathbf{M}_{move}$, $M_{call} \subset \mathbf{M}_{call}$ are the sets of included view navigations.

Figure 3 shows an example of a view navigation diagram for usual data management pages. We assume that there are moves from every view to home, which are omitted in the diagram. In this example, M_{call} includes edges when possible except calling the same method again; OR to out (Read) is not included in Figure 3. On the other hand, move edges are added selectively depending on the method semantics; a move from OR to IU is required, since the source view can provide the precondition for the target view, a resource instance data. It is possible for an output view to move to Input create view, since input create view does not have a precondition. The diagram shows that only OS view has a move to IC.

3.3 Resource Interface Diagram

When a page includes more than one resource, the diagram above becomes quite complex as a large number of views and methods should be considered. Therefore, we introduce a more abstract diagram called a resource interface diagram (RID). We define the inter-resource method calls first; inter-resource view moves are considered later.

For two different resources $r_1, r_2 \in R$, let *m* be a method of r_2. Then, the inter-resource method call

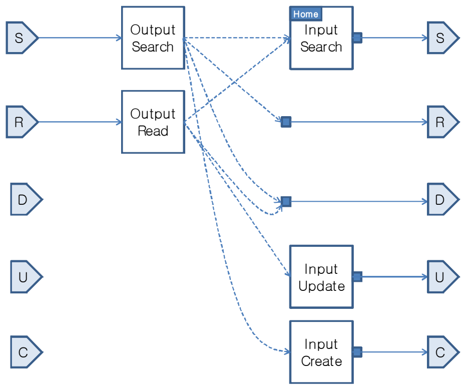


Figure 3: View Navigation Diagram.

c_{inter} is defined as follows:

$$c_{inter} = (r_1, m, t) \text{ where } t \in T^C.$$

To reduce the complexity, we use resource as a source instead of a specific view. Therefore, $c_{inter} = (r_1, m, t)$ means that a view of the resource r can call the method m, and the data of type t is transferred as an input parameter. Here, the condition $r \rhd t$ should be satisfied.

In the running example services of Figure 2, we have the following relations:

$type_{in}(search_{schedule}) = \{\perp, id(\mathtt{person}), \mathtt{date}\}$,
$type_{in}(search_{person}) = \{\perp, \mathtt{department}, id(\mathtt{person})\}$, $\mathtt{schedule} \rhd id(room)$,
$\mathtt{schedule} \rhd id(\mathtt{person})$,
$\mathtt{schedule} \rhd \mathtt{date}$.

Therefore, we can get three inter-resource method calls.

(schedule, $\mathtt{read_{person}}$, id(person)),
(schedule, $\mathtt{read_{room}}$, id(room)),
(person, $\mathtt{search_{schedule}}$, id(person)).

When the current view has data that is required to make an inter-resource call, adding the control is often helpful to users, since it allows service composition.

On the other hand, moving to an input view of another resource is only allowed as a special case. For example, at read view of person, we can add a move to a create view of schedule only when it is necessary due to application semantics.

To design the inter-resource relations, we introduce a new stereotype for a resource. The box in Figure 4(a) represents a resource and the corresponding methods.

The left/top/right parts are entries for Read/Search/Create method requests, respectively.

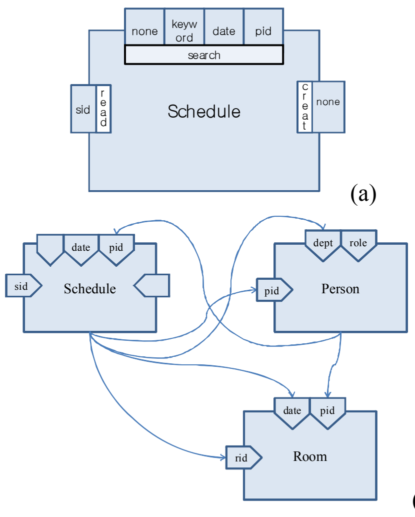


Figure 4: (a) Resource stereotype and (b) an example of resource interface diagram.

The left/top/right parts are entries for Read/Search/Create method requests, respectively. (We can assume that Delete and Update methods are not called from other resource views by the standard semantics of REST services.) The bottom of the resource box is the exit to method calls. If the method's entry does not contain a type, then the target view is the corresponding input view unless there is an input parameter with type ⊥.

In figure 4(b), a sample RID is shown for the running example scenarios. An edge denotes an inter-resource method call. From the diagram, we can obtain the design result in XML format, as follows:

```
<view resource="sched:schedule">
  <call method = "read" resource =
       "sched:person" param="sched:pid>
  <call method = "search" resource =
     "sched:person" param="sched:dept"/>
  <call method = "read" resource =
     "rm:room" param="rm:rid"/>
  <call method = "search" resource =
     "rm:room" param="xsd:date"/>
</view>
```

4 IMPLEMENTATION

In this section, we present an implementation of the proposed approach using WADL and XForms language. First, a WADL for the client page is assembled with the schema type definitions. Then, WADL and XML schema are analyzed to identify the view types and inclusion relations.

Table 4: The generated XForms code page (part).

```
<head><xf:model><xf:instance…>…</xf:instance>
   <xf:submission id= "m7" method= "GET" ref= "instance('inputParam')/param/method[@id='m7']/pid"
         action= "http://203.249.21.223:3003/people/{personId}.xml"
         replace= "instance" instance= "outputResult" target= "/result/method[@id='m7']/person">
                        <xf:setfocus control="m7-output"/>
   </xf:submission></xf:model></head>
<xf:group id="m3-output" nodeset="instance('outputResult')/result/method[@id='m3']">
   …
   <xf:submit submission="m7"><xf:label>GET person</xf:label>
         <xf:setvalue ref="instance('inputParam')/param/method[@id='home']/getParam/pid"
                 value="./schedule/attendees/shortPerson[repeat-index('shortPerson')]/pid"/></xf:submit>
   <xf:trigger><xf:label>IU</xf:label>
         <xf:copy ref="instance('inputParam')/param/method[@id='m4']/schedule" value="./schedule"/>
```

From service specifications, the client page code can be generated, including local data instances, submissions, and presentation codes. The previous research presented a means of generating these codes from web service specifications (Lee 2008). This paper focuses on the generation of navigational codes (<submit> and <trigger> elements in XForms language) using REST service patterns.

The system analyzes WADL to identify input / output types and O_{SCRUD} type of methods and their views. The type inclusion relations allow us to analyze the navigation controls c_{move} and c_{call} for each view according to the proposed method. We applied the web development convention to determine the standard REST patterns (as discussed in Section 3 and Figure 3, 4). Then, the necessary navigation controls can be obtained for each view.

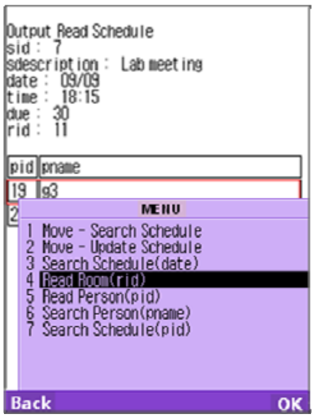
Table 4 shows the generated code for the running example and scenarios. It is worth mentioning that there are two XML instance trees, one for input parameter instances (instance('inputParams')) and one for output results (instance('outputResult')) for Read and Search methods. Therefore, the <submission> element has the input parameter location and the result save location as attributes.

Navigational controls are defined by <submit> and <trigger> elements, which have child actions to transfer view data for the control. The <submit> element in the shadowed part of the list is for c_{call} = (view$_{out}$(`m3`), `m7`, `pid`). In this case, the view has the Search result, and the method `m7` requires input data `pid`. Therefore, the transferred data is obtained from the selected item in the repeat list, and is copied from the view data to the input parameter location by <setvalue> Figure 5 shows the result screen shot OR(schedule) with navigation controls.

After checking the details of the lab meeting, the user can proceed with 7 possible tasks. The forth one is to check the location detail. Each navigation control allows the user moving to another view or submitting a new request. Most of navigations in Figure 5 are made by submitting requests, while bypassing explicit input views. This enables users to compose service methods conveniently.

5 RELATED WORK

Traditional web pages are generated at the server side, so the current page moves to another page after submitting a request. In such cases, web pages include presentation logics but not navigational controls. There are several well-established techniques for generating communication and presentation codes from web service specifications; Apache axis and Microsoft InfoPath are popular examples (refer the web sites).


Output Read Schedule
sid : 7
sdescription : Lab meeting
date : 09/09
time : 18:15
due : 30
rid : 11

pid	pname
19	g3

Figure 5: Screen shots for the scenario services.

In order to handle responses of services, an active client page should include codes for local data processing as well as include codes for local data processing as well as view navigations. The above frameworks do not help much in the design of view navigations. There has been intensive research on modelling web page navigations of web applications (Bosson 2006, Narad 2008, Gordillo 2006). Guell et.al presented a design method from requirement specifications to interaction and navigational modelling (Guell 2000). Several models are proposed for designing navigations of web applications, and some of them allow developers to reason about navigations using content models (Winckler 2003, Bozzon 2006). Narad et al. proposed an approach for enriching the navigation experience according to actual concerns (Narad et al. 2008). There has also been navigation research in the area of workflow and database. From workflow process analysis, navigations can be obtained from content models or task models (Garcia 2008). In addition, there is a study on view integration and cooperation using data types (May 2005). In this paper, we are interested in view navigations on a client page. Since this is a useful way to allow users to compose service methods, the research results from service compositions can be applied to the navigational design (Lee 2008). However, while service compositions need to find all possible combinations, the set of view navigations should be as small as possible to secure a better user interface. Our new approach aims to achieve this goal by using REST service patterns.

6 CONCLUSIONS

In this paper, we investigated a design method for view navigations in a client page. As more and more open web services are available, client pages tend to provide more than one service. To design view navigations in a web page, we can use similar approaches to conventional server-side development. We were motivated to use REST standard method and data types in order to find an appropriate set of view navigations.

To adopt REST service patterns into navigation design, we presented a new diagram called VND, which helps us to depict the relations between views and methods. When interfacing more than one resource, we introduced a resource interface diagram (RID) to design at a more abstracted level. Using these tools, we could better reason and describe navigation relations. In addition, we could determine a set of navigational controls for each view, applying conventions and REST method semantics. As a result, automatic code generation is possible for view navigational codes, without manual intervention.

We implemented the code generation system in XForms language to validate the proposed approach. The implemented result shows that the generated navigation codes allow users to easily and appropriately access all service methods.

REFERENCE

Bozzon, A., Comai,,S., Fraternali,P., Toffetti Carughi, G. (2006). Conceptual Modeling and Code Generation for Rich Internet Applications, In *ICWE 2006*, Menlo Park, California, USA.

R.T. Fielding. (2000). *Architectural Styles and the Design of Network -Based Software Architectures.* Doctoral dissertation, University of California Irvine.

Garcia, J. et al., (2008). Model-driven approach to design user interfaces for workflow information systems, *J. of Universal Computer Science*, 14(19), 3160-3173.

Gordillo, S, Rossi, G, Moreira, A., Araujo, J., Vairetti, C, Urbieta, M. (2006). Modeling and Composing Navigational Concerns in Web Applications: Requirements and Design Issues. In *Proc. of LA-Web*.

M. Guell M., et al. (2000). Modeling interactions and navigation in web applications, *Lecture notes in computer science*, 1921, 115-127.

Lee, L., Seo, K. (2008) Code Generation of an XForms Client for Service Integration, ASEA, 35-40.

May H. et al. (2005) View integration and cooperation in databases, data warehouses and web information systems, *LNCS 3730*, 213–49.

Narad, J. et al. (2008) Concern-sensitive navigation: improving navigation in web software through separation of concerns, *LNCS 5074*, 420-434.

Richardson, L., Ruby, S. (2007). *RESTful Web Services*, O'Reilly Media, Inc.

Vosloo, I., Kourie, E. (2008) Server-centric web frameworks: an overview, *ACM computing surveys*, 40(2), 4:1-4:33.

Winckler, M., Palanque, P. (2003) StateWebCharts : A formal description technique dedicated to navigation modelling of web applications. *DSVIS'2003*, Funchal.

Apache group, "Axis web services," http://ws.apache.org/axis/.

Microsoft office online, (2007). InfoPath: 2007, http://office.microsoft.com/infopath.

Web application description language(WADL), http://www.w3.org/Submission/wadl.

World-Wide Web Consortium standards including XForms, XML Schema, XPath and Cascading Style Sheets. http://www.w3.org.