Static Parameter Binding Approach for Web Service Mashup Modeling

Eunjung Lee and Hyung-Joo Joo
Department of Computer Science, Kyonggi University, Suwom, South Korea

Keywords: Service Composition, Multiple Parameters, Parameter Binding, Mashup.

Abstract: The most essential aspect of integrating web services to create mashups is determining parameter bindings for the connected requests. However, binding multiple parameters from a large and complicated xml tree is something that has not been discussed in the literature. In this paper, we presented a multi-parameter binding algorithm for repeated and nested xml trees. Moreover, we are interested in context-based parameter bindings, for scenarios where the user selects a certain context node. The proposed binding approach allows for the automatic integration of methods, even when the binding data is inside a repeated group or deep in the nested level. As a result, we can generate navigation menus depending on the contexts for the bound methods. In addition, we present a method for generating navigation codes (context menus) for the mashup views, using the parameter bindings. To demonstrate the usability of the proposed approach, we present an example of a course registration system.

1 INTRODUCTION

Service compositions and mashups have become one of the most important technologies in the development of new web applications and services. With the increasing availability of web services and the dynamic nature of these services, user-centric client-side mashups have attracted considerable attention (Pietschmann, 2010). On the other hand, a difficulty of client-side mashup pages is that they often have to interact with many services and resources.

To support a dynamic service environment, it is necessary to support the automatic generation of codes from a given set of service methods. In addition, the design of the client mashup page navigation may be complicated when it comes to handling several service requests and responses. To support the generation of navigational code for a mashup page, this paper aims to detect possible service compositions for a method's output data, as well as data bindings for the corresponding parameter passing.

In a previous paper (Lee, 2010), we introduced the concept of parameter binding the process of deciding data elements for parameters of the next request. We also introduced the concept of repeat binding, i.e., deterministic binding for the current context of the repeated part of the output tree. However, evaluating bindings for a context node is challenging if the tree has a complicated and nested, repeated structure.

This paper focuses on an algorithm for evaluating the parameter bindings of a nested, repeated structure xml tree. We introduce a top-down binding approach, using xml schema definitions, for the static evaluation of all possible bindings.

As an extension of the previous paper's code generation system, we implemented context menu generation for the multiple parameter bindings of each output view. Our approach can identify a useful set of mashup menus for a given client page context, minimizing user interactions. To the best of our knowledge, previous studies have not considered user interface issues that arise from such compositions.

This paper is organized as follows. Section 2 discusses related studies and provides background. Section 3 describes our models and introduces the concept of repeated bindings. Section 4 presents proposed method for context dependent XML parameter bindings. Section 5 briefs the

This work was supported by the GRRC program (GRRC Kyonggi 2012B03) of Gyeonggi province.

Lee E. and Joo H..
Static Parameter Binding Approach for Web Service Mashup Modeling.
DOI: 10.5220/0003991101950199
In *Proceedings of the 14th International Conference on Enterprise Information Systems* (ICEIS-2012), pages 195-199
ISBN: 978-989-8565-10-5

implementation of the parameter bindings as an extension of the code generation system. Section 6 concludes the paper.

2 RELATED WORKS

Service composition involves integrating services by connecting and relaying data. Mashups and data integration have recently been studied in depth (Pietschmann, 2010). In this paper, we are interested in service composition methods and user interface development.

There have been many pieces of research on user-interface development approaches for service compositions. Recently, several client-side service composition and execution frameworks have been published. MashArt is a framework that is intended to act as a component-based development tool for mashups, integrating all three layers of application, data, and presentation. Nestler et al. proposed a model-driven approach to develop a user interface for service compositions. Several main research frameworks have been compared and discussed, in terms of their application composition at the presentation layer (Pietschmann and Waltsgott, 2010). Lastly, the authors' of this work have published a previous paper that presented a code generation approach for client-side service navigation (Lee, 2010).

The most essential aspect of integrating web services to create mashups is determining parameter bindings for the connected requests. However, binding multiple parameters from a large and complicated xml tree is something that has not been discussed in the literature. There have been several studies on XML schema inclusion tests (Hosoya, 2003). In this paper, we presented a multi-parameter binding algorithm for repeated and nested xml trees. Moreover, we are interested in context-based parameter bindings, for the case when the user selects a certain context node.

3 MODELS AND SCENARIOS

In this paper, we are interested in finding parameter values for issuing requests to other service methods, from a given output xml tree. For an xml tree T and a method m, identifying parameter bindings in T means mapping values from T to the method parameters. Therefore, by binding parameters, a method becomes callable, since parameters are then ready to be provided. Computing a parameter binding is not straightforward if the number of parameters is more than one and if the xml tree has many repeated nodes and a complicated nested structure. In addition, our main concern is to consider the context of a selected node.

The running example in this paper is a course schedule xml tree, as shown in Figure 1, and a set of search methods for the courses and schedules and create/delete registration methods.

For example, in Figure 1, there are several repeated nodes in the tree, including grade, course, class, and slot. When a classcode node is selected, then department, year, course, and class are bound for the given context. Therefore, a user can request SearchClasses(dept, lecturer) and AddRegister(st_id, classcode) in that context. On the other hand, when the slot node is selected, we can find parameter bindings for SearchClasses(dept, day) or SearchClasses(dept, room), as shown in Figure 1.

Figure 2 shows the xml schema of the course schedule xml tree, which includes five repeated nodes, nested in depth. In this example, we assume some global values, such as student id and today's date.

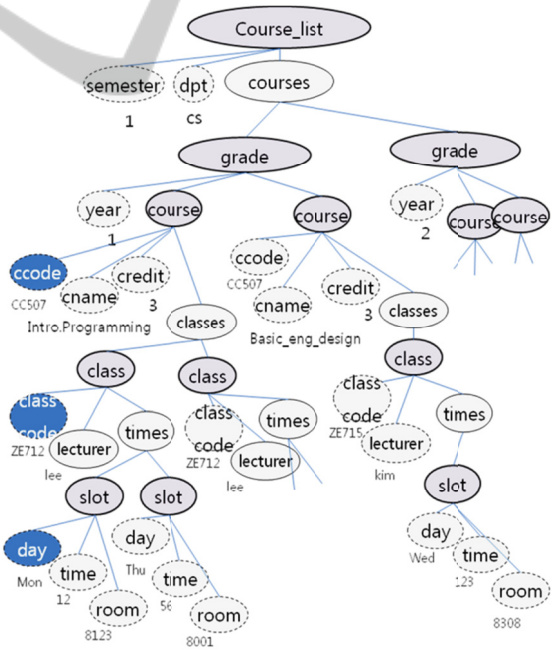

Figure 1: Tree instance and the available requests.

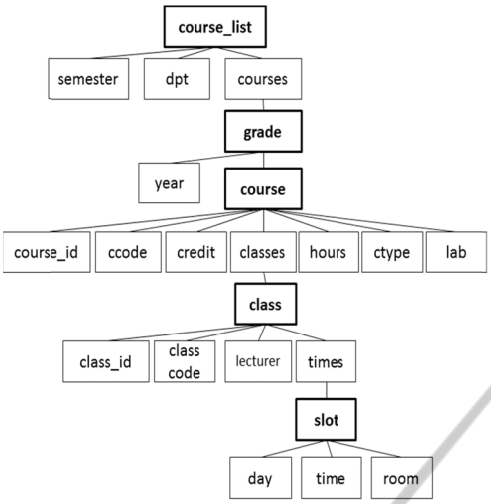

Figure 2: Schema of the xml tree in Figure 1.

4 CONTEXT-DEPENDENT XML PARAMETER BINDINGS

Composing methods by relaying output data to input parameters is an important technique in the implementation of web service mashups. For a complex, structured xml tree, it is difficult to find the corresponding parameter bindings for request methods. Moreover, dynamic parameter bindings for a selected node require inspections of every terminal element under the current context. Therefore, we object to present methods that can identify the context-dependent parameter bindings, statically.

4.1 Repeated Trees

When a schema definition is given, we can extract repeated tree structures (Lee, 2006), as shown in Figure 3. A repeated tree consists of repeated nodes; a repeated node is a node that is defined more than once in the xml schema. For computational convenience, the root node is considered a repeated node. Also, terminal nodes are grouped into repeated nodes, where each repeated node includes direct terminal descendants. Therefore, for a given repeated node r, the set of direct terminal descendants and the path from the parent repeated node are denoted by *direct_terminals(r)* and *rel_path(r),* respectively. In the example in Figure 4, repeated nodes are course_list, grade, course, class, and slot. For the class repeated node, the parent repeated node is course and the child is slot. Moreover, direct_terminals(class) = {classcode, lecturer} and rel_path(class) = course_classes/. Note that @st_id and @today are global static values.

4.2 Parameter Bindings

For the example in Figure 4, we present our approach of parameter bindings with following service methods:

- m1 = SearchCourses(dept, syear)
- m2 = SearchCourses(dept, ctype)
- m3 = SearchClasses(dept,lecturer)
- m4 = AddRegister(user_id, classcode)
- m5 = SearchClasses(dept, lecturer, day).

At each repeat node, we can find parameter mappings using direct terminals. For a given method *m* and its parameters, a binding table for m, denoted as btable(m), is defined as a tuple of elements as many as m's parameters. For the example of Figure 4, btables are shown in Table 1.

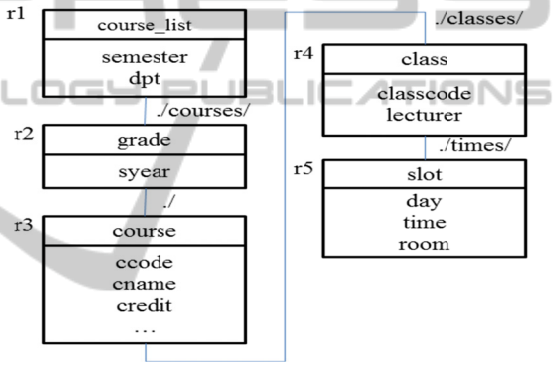

Figure 3: The repeat tree of the course schedule schema.

For each method, we can find the repeat node where the parameter binding is completed during a top down traverse. This repeated node is called "a base context node." For the above example, parameter bindings of m1 and m2 are finished at r2 and r3, respectively. Therefore, m1 and m2 have r2 and r3 as the base nodes, respectively.

Then, the binding path for each parameter is computed from the base context node. For the example m3, the binding paths for the parameters are `[//course-list/dept, ./lecturer]`. Now, we define *parameter_bindings* for a method *m* as follows.

Definition. Let $m \in Methods$ and r be a repeated node in *output(m)*. Moreover, let m have input parameter types $[p_1, p_2, ..., p_k]$. Then, the parameter bindings for m are defined as follows: *parameter_bindings(m)* = $\{(r, [\pi_1, \pi_2, ..., \pi_k] \mid r$ is a base context node of m in *output(m),* and π_i is a relative path of the node which is mapped to $p_i\}$.

Algorithm 1: Top-down computing of XML parameter bindings.

```
        Input: Methods = [m1, m2, ..., mk].
               params(mi) = [pi1, pi2, ..., pni], ni is the number of parameters of i-th method mi.
               RT: the repeat tree of the output type schema tree, root: the root node of RT.
        Output: parameter_bindings
1       Procedure findAllParamsBindings(root):
2           ∀ 1 ≤ i ≤ k,
3               Let btable[i] = [b1, b2, ..., bni], bj = null, 1 ≤ j ≤ ni.
4               parameter_bindings(mi) = null.
5           Call bind_repeat(root).
6       Procedure bind_repeat(r):
7           ∀ mi ∈ Methods, let btable[i] = [b1, b2, .., bni] and binding of mi is not finished,
8             ∀ px s.t. x-th parameter of mi where bx = null, 1 ≤ x ≤ ni,
9           ∃ tj ∈ direct_terminals(r), s.t. tj ~ px ,
10                bx = (r, tj).        // r is the base current node of bindings(mi)
11            If btable[i] is filled at this level,    // binding is now finished,
12          for 1 ≤ x ≤ ni , let bx = (r', tj ), r ≠ r'
13                πi = a relative path of (r', tj ) from the current repeated node r,
14            Let parameter_bindings(mi) ← add (r, [π1, π2, ..., πni]).
15          If there is any method where binding is not finished,
16            ∀ r' ∈ repeat_child(r),
17              call bind_repeat(r').
```


Table 1: Binding tables on each repeated node of Figure 3.

		r1	r2	r3	r4	r5
m1	dpt	O				
	syear		O			
m2	dpt	O				
	ctype			O		
m3	dpt	O				
	lecture				O	
m4	user_id	@st_id				
	classcode				O	
m5	dpt	O				
	lecturer				O	
	day					O

For the example of Figure 3, parameter bindings for methods m1 to m5 are as follows:

- parameter_bindings(m1) = {(r2, [../../dpt, ./year])}
- parameter_binding(m2)={(r3, [../../../dpt, ./ctype])}
- parameter_bindings(m3) = {(r3, [./cname])}
- parameter_bindings(m4)= {(r4, [../../../../dpt, ./lecturer])}
- parameter_bindings(m5)={(r4, [@st_id, ./classcode])}

Now, we are ready to present the parameter binding algorithms using the notations introduced thus far.

Algorithm 1 traverses the xml schema tree from top to bottom, to identify the mapping of data elements for the parameters of methods in *Methods*. The algorithm visits repeated nodes through their parent-child relations (line 17), gathering the node paths that are matched to method parameters (line 9). If the binding is completed at the repeated node *r* (line 11), then, r is the binding context base. Once we find the base context node of *m*, we evaluate the paths of the matched nodes (line 13) and parameter binding is completed for the method. The top down recursive call is continued while methods remain to be bound and while there are more descendants to visit.

4.3 Generating Context Menus

The *parameter_bindings(m)* refer to the parameters that are ready when the repeated node is bound. Therefore, the method *m* is callable when a user selects one of the repeated nodes. Context popup menus include all callable method requests for the selected context. Therefore, we need to compute all methods bound to a given repeated node from *Methods,* a set of available methods. Thus, we can define a set of callable methods for a given repeated node r as follows:

$$callable(r) = \{m \mid m \in Methods, (r, pb) \in parameter\_bindings(m) \text{ for some } pb\}.$$

For example, the schema and the parameter bindings in Figure 3 show that *callable(r4) = {m3, m4}*. On the other hand, the repeated node represents a repeated level, so if any of the terminal values at this level are selected, then the corresponding methods can be called. For example, selecting a node `ccode` determines the repeated node `course` and its direct terminal descendants.

For a given xml tree, we have an output view

rendered with terminal nodes, as shown in Figure 5. The context menu is provided for a request call for the bound method.

5 IMPLEMENTATION

In this section, we introduce the implementation result of the top down parameter binding methods by Algorithms 1 and 2, introduced in Section 4. In a previous study, the authors introduced the MashupBench system (Lee, 2010), which is a platform providing service selection, data mapping, and mashup code generation. Figure 4 shows the overall architecture of the system. In this paper, we extended the analyzer and the code generator allows multiple parameter bindings for complicated and repeated xml tree structures. We also enhanced the popup menu-generating algorithm to efficiently generate javascript code.

The system takes WADL (Web Application Description Language) files, which is the standard for describing REST style services (WADL, 2006), as service description inputs to specify the available services. Schema files are read by the analyzer to identify parameter bindings for the service methods.

To enable efficient code generation, we construct all popup menus when the view is created at the time of output data response. Since we statically computed parameter bindings beforehand, only menu visibility and event handling run dynamically.

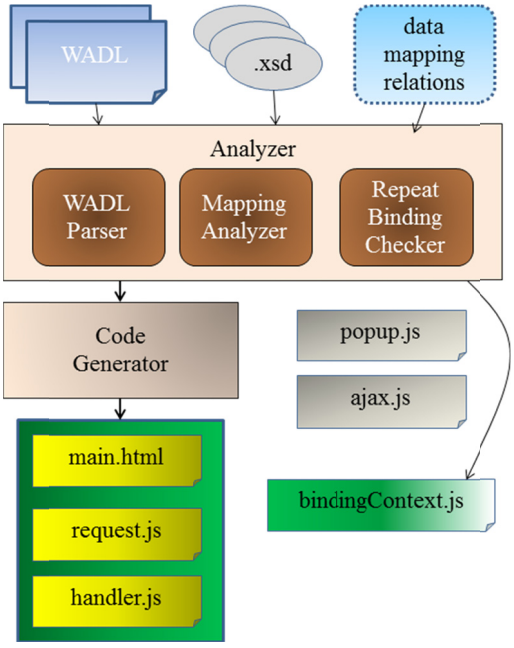

Figure 4: Architecture of code generation system using the proposed approach.

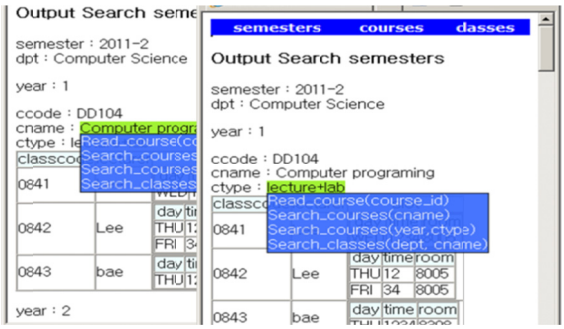

Figure 5: Static-time generated popup menus for the same repeated node.

6 CONCLUSIONS

In this paper, we presented a multi-parameter binding algorithm for repeated and nested xml trees. Moreover, we are interested in context-based parameter bindings, for scenarios where the user selects a certain context node.

The proposed binding approach allows for the automatic integration of methods, even when the binding data is inside a repeated group or deep in the nested level. As a result, we can generate navigation menus depending on the contexts for the bound methods. In addition, we presented a method for generating navigation codes (context menus) for the mashup views, using the parameter bindings.

Since current mapping approaches do not consider nested repeat structure, our methods could be applied to service mashup frameworks to enhance the mashup connections. We are working on implementing the incremental computation of the parameter bindings.

REFERENCES

Stefan Pietschmann, et al., 2010. A Thin-Server Runtime Platform for Composite Web Applications. *ICIW '10*.

Florian Daniel, et al., 2009. Hosted universal composition: models, languages and infrastructure in MashArt. *ER'2009*.

Eunjung Lee and Kyong-Jin Seo, 2010. Designing Client View Navigations Using Rest Style Service Patterns. *WEBIST'2010*.

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

Eunjung Lee, 2006. Inline binding of XML data. Proc. Of ICMOCCA'06.

Haruo Hosoya, 2003. Boolean operations and inclusion test for attribute-element constraint, *ICIAA'03*.