

Developing Lightweight Context-Aware Service Mashup Applications

Eunjung Lee and Hyung-Joo Joo

Computer Science Department, Kyonggi University,
San 94 Yiui-dong, Suwon-si, Gyeonggi-do, South Korea
ejlee@kyonggi.ac.kr, vagabond@kyonggi.ac.kr

Abstract— As web services increase in popularity, the number of available services expands and they become more dynamic. Thus, it is increasingly important to allow the dynamic development of mashups on demand for a given context. To allow mobile users convenient access to the necessary services requires simple and lightweight context-aware methods to discover, select, and compose these services. In this paper, we introduce a model of a mashup page with a JavaScript wrapper that is generated automatically without user intervention. MyServices, the mashup construction system, allows service providers to register context-aware selection rules. Moreover, the JavaScript codes generated by MyServices can run in any browser environment and support navigation between services with minimal user interaction.

Keywords—context-aware service composition, ubiquitous services, service mashups, parameter matching.

I. INTRODUCTION

As browser-enabled mobile devices become increasingly common, and the number of available web services expands, we can say that the web is becoming truly ubiquitous. Moreover, the development of mashups for user-customized services, using contexts such as locations and user-profiles, is anticipated to become more important to mobile users[1,2].

There have been a number of studies on context-aware service composition. In particular, the UbiCompForAll project [13] presented a semantics-based service description and composition platform that provided a user-programmable tool. The system consists of a user editor called EasyComposer and a mashup running environment on the client-side. They focused on a graphical editor for designing the flow of user activities using an ontology-oriented service description and recommendations [12, 13]. Therefore, the platform needs heavy server-side engines and requires the proprietary client tool on which the mashup runs.

In this paper, we introduce the MyServices system, whose functions are service selection and mashup generation. As we consider a mashup to be a collection of services without a definite sequence, we model service selection rules as recommendations that determine the order of activation for given contexts. As a result, we provide a simple and straightforward set of recommendation rules and their registration interface, as well as an efficient method for the computation of selections.

The system creates mashup applications in JavaScript and HTML that could run on any mobile device platform. Moreover, applying our previous research results from MashupBench [9,10], the mashup client contains a navigation interface to connect service compositions for the repeated part of the XML tree. As a result, the generated client code is simple and lightweight, with convenient menus for users to navigate mashup services. In addition, we present a method to insert context values in client codes and convey them as service composition parameters.

This paper also presents examples of campus services with contexts like locations, users, and times. In particular, the location data is gathered by WiFi access point IDs, which enables the recognition of indoor locations. The implemented system runs on the Android platform, and provides useful menus and interfaces.

This paper is organized as follows. The next section discusses some related work, and section 3 presents a context-aware service recommendation model and the client mashup code architecture. Section 4 describes the theory behind the proposed MyServices system, and section 5 discusses its implementation, as well as some performance issues. Finally, section 6 concludes the paper.

II. RELATED WORK

There are three areas of service composition related to our work: context-awareness, mashup modeling for dynamic service compositions, and code generation and the ubiquitous running environment of the resulting mashups.

The modeling, processing, and adaptation of context information has been proposed in many studies, both model-driven and ontology-based, in order to utilize contextual information in web service compositions. CODA and ContextUML [1] are model-driven approaches to the processing of context information in abstract specifications. In contrast, Buriano presented context-aware, ontology-based service discovery, and Medjahed developed a context-based matching method for web service compositions [5]. Also, Pietsmann et al. proposed workflow-style service compositions using semantic context descriptions [11]. UbiCompForAll is a service composition platform that utilizes semantic context information for selecting, matching, and sequencing services into mashups [12].

There has been intensive research into dynamic service composition. The quality of dynamic service compositions is measured by whether the services selected are those the users really want, whether users can specify the composition at run time, and the user-friendliness of the specification interface. Bronsted et al. mentioned that automatic service mashup approaches rarely align with users' expectations, so run-time user specification is essential [3]. Also, for context-awareness, we need to consider context-adaptation in the quality of service compositions. Dustdar proposed classes of service composition approaches [5]. Recently, workflow-style service compositions have become popular owing to advances in workflow modelling techniques [11]. The UbiCompForAll platform uses workflow-style service composition models as well as a user-programmable mashup editor [12].

We presented the RESTful collection of services as a form of mashups [6,10] in order to adopt useful, real-life mashup patterns, with the aim of generating client mashup codes automatically [9]. Also, we proposed a parameter binding algorithm for the repeated part of the XML data output by service methods. This enabled us to generate useful navigation menus for the mashups.

In this paper, we present an approach that allows service providers to give context-aware recommendations. Our work has the same objects as UbiCompForAll[12,13], as we are interested in developing ubiquitous and dynamic web service compositions using contextual information. Compared to UbiCompForAll, we target lightweight mashup codes native to mobile browsers, as well as a lightweight server system. The service selection and mashup is performed dynamically based on the client context values, without the need for intervention by the user.

III. WEB-SERVICE RECOMMENDATION SYSTEM

In order to provide location-based mashup services for mobile users, a service composition platform needs the following functions. First, we need a service directory containing the specifications of all available services. Secondly, the platform should be able to describe and register context-aware rules, via either the service providers or ordinary users (rule registration in Figure 1). Thirdly, the composition system should be able to gather context data and select an appropriate set of services (service selector). Finally, the platform should be able to generate mashup codes dynamically for the selected service set. Figure 1 shows the overall architecture of the proposed MyServices system developed in this research.

In this paper, we consider recommendation rules to be registered by service providers, in blocks of service specifications. Using recommendation rules, a server decides whether a given service should be activated or not based on the context. The rules are described in terms of the context types defined in the composition server. The MyServices system defines context categories, attribute types for each category, and a set of possible values for the attribute types. We denote a set of services in the domain as 'Services', and context values as $[c_1, \dots, c_k]$, $c_i \in \text{ContextValues}(C_i)$ for the i -

th context type C_i . For example, if we assume that the context types are a room-number and a location-name, then the corresponding context values might be " $8101 \in \text{ContextValues}(\text{room-number})$ " and " $\text{library} \in \text{ContextValues}(\text{location-name})$ ".

Recommendation rules can be either public or private. A rule is public if it is computable in the composition server system and private if it depends on information inside the service server systems. For example, if a service is activated for certain locations and/or users, then the rule could be public. In contrast, when lecture services are only activated for registered users, the rule might be private. Recommendation rules have the following form:

$$\text{enabled}(S_i, [c_1, c_2, \dots, c_k]) = \{\text{true}, \text{false}\},$$

where $S_i \in \text{Services}$ and $c_i \in \text{ContextValues}(C_i)$ for $1 \leq i \leq k$.

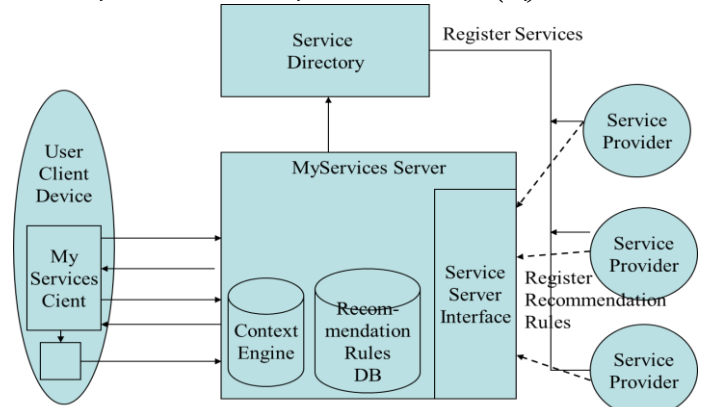


Figure 1. Context-aware service composition platform architecture.

The private rule is a retrieval method of the service server to inquire whether the service is activated or not for given contexts. MyServices reserves 'isActivated' for the private rule's name and sends context data as parameters. Therefore, the service selection is as follows:

$$\text{selectServices}([c_1, c_2, \dots, c_k]) = \{S \mid S \in \text{Services},$$

$$\bigwedge_{i=1}^k$$

$$\text{enabled}(S, c_i) \text{ and } S \in \text{isActivated}([c_1, c_2, \dots, c_k])\}.$$

Based on the context types, we construct a recommendation rule registration interface, as shown in Fig. 2. This allows service providers to specify recommendations for their services, where the services are given by the URL of specification files in WADL (Web Application Description Language) [13].

The top row allows users to select the categories for which context rules are to be specified. In Fig. 2, the user selects the 'location' and 'user' categories, so the corresponding specification panels are shown in the next rows. Each category's attribute types are shown as selection controls with candidate values. In this paper, the recommendation rule specifies a set of activation conditions. This context information could be easily modeled in any semantic language.

Finally, users can activate the private recommendation rule if it exists.

configure: csAdmin

Registration of Recommendation Rules

wadl URL:

```
<?xml version="1.0" encoding="UTF-8" ?>
- <application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:qna="http://203.249.22.109:8080/qna.xsd">
- <resources base="http://203.249.22.109:3005">
- <resource path="surveys">
- <method name="GET" id="search_research_001">
- <request>
```

category	☒ location ☒ user
public rule	building: 8 floor: 8-3 room: 8302 X
	type: lec_room OK X
	type: std dept: cs X
private rule	serviceActivate: OK

Figure 2. Recommendation rule registration interface for service providers

IV. LIGHTWEIGHT MASHUP CLIENT CODE GENERATION

The mashup page should provide an integrated interface for the selected services, including support for the use of composed service calls. This page includes input/output views for selected service methods and navigation menus between methods. As the views and input/output connectors can be generated at static time [10], only the mashup page wrapper and the mashup composition information should be generated dynamically. This paper assumes that each view occupies the whole window, as multi-view layouts are beyond the scope of this paper.

The mashup HTML page is a wrapper for selected service resources. As services are selected dynamically, navigation menus should be provided for all services and their methods (the top menu bar in Figure 3). When a service method is selected, the client displays an input view if the request requires parameter values to be entered. Otherwise, the request is sent immediately and the client view switches to the output of the returned result. Input/output views and request/response codes are generated statically and stored in the code base of the MyServices system code generator module.

The meaningful context values gained during service selections are often employed as parameter values. As in the example of Figure 3, the user ID and location are used as parameters for the composed service request. In addition, they are shown in the output view so that users can check their context.

The main advantage of the mashup codes generated by our system is that they have embedded menus in each output view for calling composed service methods, as shown in Fig. 3(b). The embedded menus enable users to request methods whose parameters are uniquely determined by the selected node in the repeated part of the XML tree. This is the main result of our previous paper on multi-parameter bindings in the repeated XML tree data [9]. This approach allows service composition using the currently selected context of the XML tree, which alleviates the need for users to select or enter parameter values for the next service request.

However, as our previous paper dealt with code generation of a given set of services, the performance issues of multiple synchronous requests for code generation were not considered. This paper extends the previous result to allow such multiple requests by reusing the parameter binding information. In order to reuse such information for subsequent code generation, we emit the bindings as JavaScript hashmaps. Table 1 shows the structure of the parameter bindings.

Table 1. Information structure of parameter bindings (param-bindings.js in Figure 4).

For all methods m with output XML data type T ,
 for all repeat node r in T ,
 for all methods m' callable at r ,
 for all parameters $param_i$ of method m' ,
 the relative path of the node e from r such that
 whose value has a type compatible with $param_i$.

hoon kim, computer science, room 8504

lecture **CSD**

[lecture]Output Search todos

testElement :

day	lec_name	title
Mon	compiler(quiz)	NFA partice
Thu	comp	...

hoon kim, computer science, room 8504

lecture **CSD**

[CSD]Output Search live survey

survey_name	lec_name	due
final suvery	...	...

hoon kim, computer science, room 8504

lecture **CSD**

[CSD]Output Read live survey

subject_name : compiler
 survey_name : final suvery
 term : 2011-6-12

question_content	shortAnswer
What your grade?	termanswer_context
	1 2
	2 3
	3 4
Satisfied with the content of the lectures?	termanswer_context
	1 very good
	2 good
	3 normal
	4 bad
	5 kerv bad

Figure 3. Recommendation rule registration interface for service providers

When the service response arrives, the output XML data is processed by the response-handler JavaScript code. While generating a domain tree for rendering the output data, each node attaches menu triggers to provide composed service method calls. The event-handler code has the same function for selected node contexts, as it applies parameter binding information as in Table 1 to generate the menus.

V. IMPLEMENTATION

In this section, we discuss the implementation of the MyServices system, and evaluate its performance. The system architecture, as shown in figure 4, consists of the following four components:

Service provider interface module.. This enables service providers to register their recommendation rules in the system. Figure 2 shows the interface for context information types defined in the composition server. The rules are stored in a rule engine.

Context evaluation module. When the client initiates a connection, the system collects the necessary context data sent from the MyServices client program. The context engine enables logical context values to be computed from the physical (not processed) context data. The context engine can be external to the system for cooperation with other systems.

Service selection module. This module computes the recommendation rules using the logical context values. The rules are applied to every available service in the directory to select a set of recommended services.

Code generation module. Using the services selected in the previous step, this module generates the mashup client code in HTML and JavaScript, as explained in section 4.

The resulting mashup code is provided to the client so that the user can access services via the unified web page interface.

We evaluated the code generation performance for multiple contiguous requests. Since we extended the code generator module to reuse parameter binding information for the same

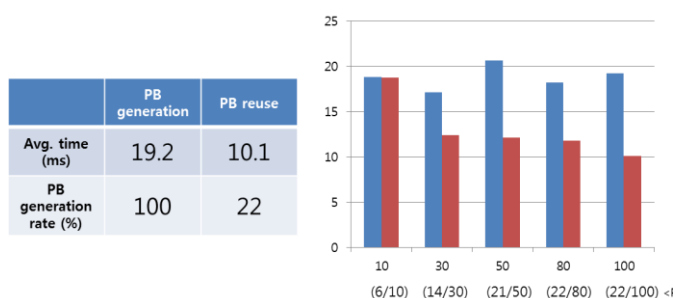


Figure 5. Performance evaluation of contiguous code generation requests. Average elapsed times for number of contiguous request. (blue: generate PB everytime, red: reuse PBs)

set of services, the question is whether the code generation time is decreased by this reuse. We measured the time taken for the selection and code generation, i.e., from the request arrival to the response. To exclude the effect of service selection, we randomly chose 3–5 services from 10 candidates.

As shown in figure 5, the time taken for code generation decreases as the number of generation requests increase. For 100 contiguous requests, the average time elapsed is reduced to 48% of that for a single request. This is due to the parameter binding computation time; it requires analysis of the XML schema definitions for service method outputs.

VI. CONCLUSION

In this paper, we presented a context-aware service composition platform for a ubiquitous and mobile computing environment. The developed system, MyServices, includes the following functions: 1) it allows users to register context-based recommendation rules via an easy to understand interface; 2) the registered rules are combined with context values provided by the client to compute a set of recommended services. Also, we generated a lightweight client mashup code in JavaScript and HTML, which are native

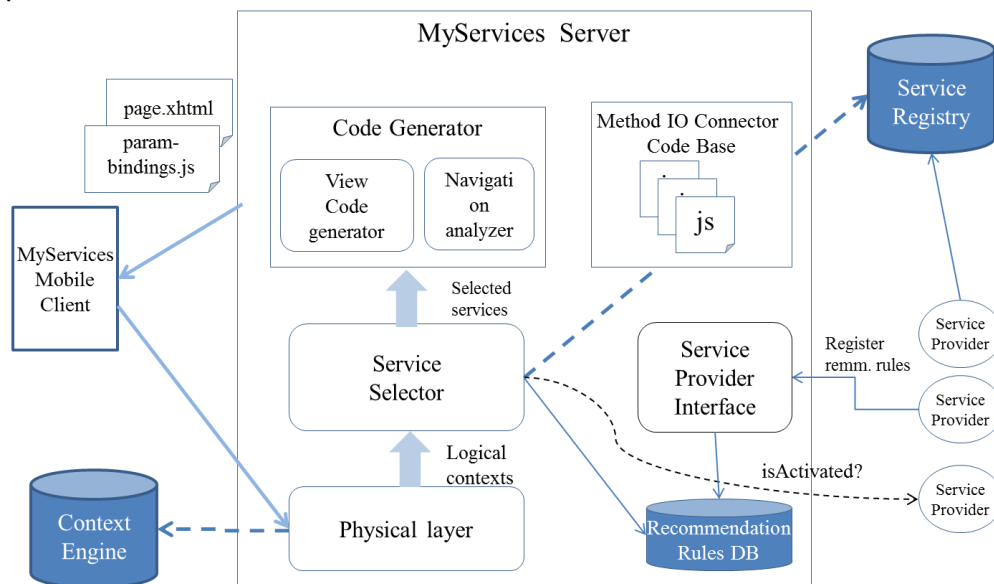


Figure 4. MyServices system architecture

to any mobile browser.

Therefore, we have been able to provide a simplified context information model and specification interface that enables users to control the selection of services. Moreover, experimental results for sample services showed this selection process and the generation of code to be efficient.

By applying the authors' previous result on modeling embedded navigation menus, the output mashup client code was able to provide popup menus for calling composable service methods on the selected node. This technique enhances the usability of the mashup output for dynamic service composition. Moreover, users can connect the output data to the input parameters of another request with minimal interaction via the popup menu.

Though we presented experimental results for the performance of code generation, further evaluation of the output results is necessary. We plan to extend this research to formally evaluate the usability and convenience of the generated codes in the ubiquitous environment.

References

- [1] Sheng, Q., Yu, J., Dustdar, S.: Enabling Context Aware Web-Services. CRC Press. (2010)
- [2] Alamri, A., Eid, M. and El Saddik, A.: Classification of the state-of-the-art dynamic web services composition techniques. *Int. J. Web and Grid Services*, Vol. 2, No. 2, pp.148–166. (2006)
- [3] Bronsted, J., Hansen, K. and Ingstrup, M.: A survey of service composition mechanisms in ubiquitous computing. *UbiComp 2007*. (2007)
- [4] Dustdar, S. and Schreiner, W.: A survey on web services composition. *Int. J. Web and Grid Services*, Vol. 1, No. 1, pp.1–30. (2005)
- [5] Medjahed, B. and Atif, Y.: Context-based matching for web service composition. *Distrib Parallel Databases*, Vol. 21, pp.5–37. (2007)
- [6] R.T. Fielding: Architectural Styles and the Design of Network-Based Software Architectures. Doctoral dissertation, Dept. of Computer Science, Univ. of Calif., Irvine. (2000)
- [7] Keidl, M. and Kemper, A.: Towards context-aware adaptable web services. *WWW'2004*, pp.55-65. (2004)
- [8] Lee, E.: Inline binding of XML data. In: *Proc. ICMOCCA'06*, (2006)
- [9] Lee, E. and Joo, J.: Designing mashup view navigations using data binding approach. In: *LNCS / ICHIT'2012*, to be appeared. (2012)
- [10] Lee, E., and Seo, K.: Designing Client View Navigations Using Rest Style Service Patterns. In: *WEBIST'2010*. (2010)
- [11] Pietschmann, S., Nestler, T., Daniel, F.: Application composition at the presentation layer: alternatives and open issues. In: *iiWAS'2010*, pp.461-468. (2010)
- [12] Rune, S., Ullah, K., Herrmann, P.: End-user Composition of Web-based Services: The "Plus Alpha" Approach. *Proc. of INWWCT 2011*, pp. 51-61. (2011)
- [13] UbiCompForAll homepage, <http://www.sintef.no/Projectweb/UbiCompForAll/>.
- [14] 14. Hadley, M.: Web application description language (WADL), <http://www.w3.org/Submission/wadl>, (2009).