Consumption and Composition of Web Services and non–web services

Rohit Kishor Kapadne

Abstract— Nowadays service oriented architecture used everywhere even in mobile applications for application domains like telematics and smart home. Developers puts numerous efforts in making composite SOAP Services, some more importance has been given to consuming combination of SOAP, Rest and non–web services into a composite process for execution of complex tasks in different mobile devices. Two major challenges in this approach is, first one consuming and composing heterogeneous web services with different protocols and media type which includes SOAP, Restful and OSGi services and other one is how to integrate non-web services, like Web contents and mobile applications, into a composite service process. We are proposing approach for consuming and composing SOAP, Restful and non-web services with two unique features an extended BPEL engine bundled with adapters to enable direct consumption and composition of SOAP, Restful and OSGi services by following adapter pattern; and enabling conversion of web content and android activities into OSGi services. Basically we are reducing network traffic and turnaround time as compared to traditional approach.

Keywords— Heterogeneous service composition, web service, service composition engine.

I. INTRODUCTION

In recent days, while developing mobile application developer preferred to use service-oriented architecture (SOA) in domains like telematics, smart home, business and internet of things (IOT). Integrated heterogeneous web services provide composite services in domains like IOT. These heterogeneous services include Simple Object Access Protocol (SOAP) services, RESTful services, and OSGi services. SOAP is protocol used for exchanging information between two systems, information during the implementation of web services, and SOAP services can be used for composing web services using Business Process Execution Language (BPEL) [8]. RESTful services is an architectural style which can be used to construct services for processing clients requests. Usage of RESTful services in increasing day by day due to its easier interface for communicating with external services and displaying object features. OSGi technology provides an open service platform for service installation, activation, and management in devices. It offers OSGi services support point-to-point remote service delivery programs that can be beneficial for assembly, dynamic binding, and execution of device services. In same manner efforts has been taken on developing composite SOAP services[8] as well, Need to concentrate on consuming and composing on combination of various flavors of web services like SOAP, Restful and non web services into a composite process in order to complete complex tasks on a various of mobile applications. Main show toppers can be explained from the following two perspectives:

- How to consume and compose heterogeneous web services with different protocols and content types like SOAP, RESTful and OSGi services. A complex process may involve consumption of SOAP, RESTful and OSGi services over SOAP protocol, HTTP and Java method call, respectively. Additionally, it may contains various message content types, like SOAP, JSON, YAML, Protocol Buffer and Java objects, in composing the heterogeneous web services.

- How developers can integrate non-web services, along with Web contents and mobile apps to a composite web service process? Non-web services in a mobile environment basically contain Web contents and mobile applications. Web contents are in the form of HTML docs. Execution of program in mobile applications is start with GUI components.
A framework for invoking and composing SOAP, non-SOAP and non-web services on mobile devices with two key features: a BPEL engine extended and bundled with adapters to enable the direct invocation and composition of SOAP, RESTful and OSGi services based on Adapter pattern, which provides a flexible mechanism for adding newly developed adapters for invoking some other kinds of services without modifying the core BPEL engine; and two transformation mechanisms devised to enable the transformation of Web contents and Apps activities into OSGi services that can be composed by the extended BPEL engine. Figure 1 shows the conceptual model of the framework for invoking and composing SOAP, non-SOAP and non-web services. Three adapters for invoking SOAP, RESTful, and OSGi services are developed to extend our BPEL engine, and two transformation mechanisms for converting Web contents and Android activities are undergone a conversion into OSGi services before being invoked and composed at runtime by the extended BPEL engine through adapters.

II. RELATED WORK

It is beneficial that combining various web services into a composite service than finding a complex and preparatory atomic service that which can be used to satisfy a special request. The resultant services we can say composite services can be used as atomic services by themselves in composition with other services to satisfy client’s requests. BPEL4WS provides combination of block and graph structured process models, and variables associated with message types which can be specified as output or input variables to call, receive, and reply web services.

In Recent time, BPEL has extended support for modeling the composition of various web services, like RESTful and OSGi services. [9] REST (REpresentation State Transfer) is an architectural style of software architecture for distributed hypermedia systems like the World Wide Web. It defines a set of architectural principles by using those any one can design web services that which focuses on a system’s resources, also includes resource states which can be addressed and transferred over HTTP by various clients developed in variety of languages.

In [4], F. Curbera describe a composite model for composing RESTful services. While calling RESTful services, the response message from a service are stored as BPEL process variables. However, how it can be transformed in to different content types for follow-up service invocations is not specified.

In [3], K. He describes integration of both SOAP and RESTful services with a hybrid orchestration based on REST orchestration engine and a BPEL engine. The composite service
workflow categorized into two types of sub-workflows according to the types of web services to be composed. However, the message transformation between the SOAP and RESTful is not specified.

In [2], S. Farokhi proposed a framework, named MDCHeS, for supporting dynamic composition and for usage of both SOAP and RESTful Web services together in composite services with three different views: component view, process view and data view. However, it’s also not describe about the message transformation between the SOAP and RESTful services.

In [5], J. Nitzsche proposed extended BPEL 2.0 with a WSDL-less interaction model, known as BPELlight, for enabling business logic and Web service technology together, including WSDL, by introducing a new and one type of interaction activity which resumes all BPEL interaction activities. The main aim of the proposed work is to reuse and bound specific service interfaces in any interface description language enable modeling processes or process fragments. Our proposed approach focuses more on how to model and implement the binding relationships among heterogeneous services.

In [6] [7], C. Pautasso proposed a process-based composition language for composing RESTful and traditional WSDL-based services based on BPEL. The local adapters (e.g. XSLT, JavaScript) are used to process the data and transform it to make it compatible with what the other service requires. In our approach, SOAP, RESTful and OSGi services are all supported for invocation while executing a composite process. Furthermore, SOAP, JSON, YAML, Protocol Buffer, and Java object messages returned from a service can be transformed into Java objects used as variables of a BPEL process. After the first transformation, the variables can then be transformed again into messages of different content types for follow-up service invocations.

### III. FRAMEWORK FOR COMPOSING HETEROGENEOUS SERVICES

In this work, we are applying Adapter pattern on BPEL engine for invoking and transforming SOAP, RESTful and OSGi services. Figure 2 shows the heterogeneous service composition framework system architecture. An overview of BPEL engine architecture has been given below.

![Figure 2: System architecture of the heterogeneous service composition framework](image)

The entry point of the BPEL engine is Composite Service Activator component with inputs of WSDL and BPEL documents. Deploy component deploys the documents with BPEL properties and uses Service Generator component to transform the BPEL process defined in the BPEL document into Java classes.
Reader component is in charge of generating a BPEL model by analyzing the WSDL and BPEL documents and instantiating the Java class. BPEL Manager Component starts the BPEL process by creating a BPEL process instance from the BPEL model with instances of the Java classes.

BPEL Runtime component runs BPEL processes at run time. A BPEL process consists of two kinds of activities: basic and structured. A basic activity describes elemental steps of the process behavior. A structured activity encodes control-flow logic and can contain other basic or structured activities recursively. Thread Manager Component handles the multi-threads based on XPath and XQuery.

SOAP, RESTful, and OSGi Adapters are responsible for invoking SOAP, RESTful, and OSGi services, respectively. The three adapters implement the Partner Link Handler interface, which enables the BPEL engine to delegate service invocation behaviors to the services. Mobile Applications Transformer transforms Mobile apps executable files into OSGi services, and Web Content Asset Transformer transforms web contents into OSGi services.

A. Composing RESTful Services

Figure 3 shows the architecture of the RESTful service adapter. During the execution process, the extended BPEL engine parses the WSDL document of the RESTful service. The properties of the elements are obtained from the WSDL. The protocol property determines the protocol of the RESTful service. The url property determines the network address of the RESTful service. The format property returns the content type of the RESTful service messages. When the messages return from the RESTful service, they are transformed into Java objects as runtime variables of the composite process through the JSON/YAML/Protocol Buffer to Java object module based on open source org.json and org.ho.yaml packages. Messages are converted into JSON, YAML or Protocol Buffer messages before they are sent as inputs to another web service.

B. Composing OSGi Services

Figure 4 shows the architecture of the OSGi service adapter. By referring to the extended WSDL, the extended BPEL engine reads the properties of the OSGi service by using the OSGi adapter, which inherits a partner link handler interface. The adapter searches for the OSGi service
through the OSGi registry to get the service’s reference, and then invokes the services through method call.

![Figure 4: OSGi adapter architecture](image)

C. Composing Web Component

As Web contents in HTML are usually readable only for human and are with diverse information, the web content asset transformer enables a system designer to identify the assets in the web pages and transform them to OSGi services at design time so as to be composed in a composite process at runtime.

![Figure 5: Web Content asset transformation architecture](image)

Figure 5 shows the architecture of the web content asset transformer. The transformer consists of three components: a Webpage list manager, a data parser and a bundle packager. The Webpage list manager enables a user to identify and select the assets in the web pages, such as images or texts. The data parser parses the selected web pages and extracts the assets. The bundle
packager transforms the extracted assets into an OSGi bundle. In the following, we detail how to extract web content assets and how to transform these assets into an OSGi bundle.

IV. EXPERIMENTAL RESULTS

In the experimental environment, the client-side mobile device with 1 GHz Cortex A8 CPU and 512 MB memory. The server runs Java virtual machine 6.0 and the OSGi platform is Apache Felix. We calculate the turnaround time and the network traffic by adopting our approach and the traditional approach over a wireless network with a 256 KB/s bandwidth which is estimated and controlled by bandwidth controller software. A site denotes a smart living control environment. There is up to 40 sites in the experiment. For each kind of service with a certain number of sites, the turnaround time is the sum of the time gathered for all the sites.

### TABLE 1: TURNAROUND TIME OF INVOKING RESTFUL SERVICES

<table>
<thead>
<tr>
<th>Number of Sites</th>
<th>Turnaround Time (milliseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traditional Approach</td>
<td>Proposed Approach</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>350</td>
</tr>
<tr>
<td>10</td>
<td>1200</td>
</tr>
<tr>
<td>25</td>
<td>1400</td>
</tr>
<tr>
<td>30</td>
<td>1800</td>
</tr>
<tr>
<td>35</td>
<td>2300</td>
</tr>
<tr>
<td>40</td>
<td>2600</td>
</tr>
</tbody>
</table>

The experimental results shown in Table 1 the turnaround time of invoking the controller RESTful services in our approach is less than those of the traditional one due to the request message data are directly transferred in between the extended BPEL engine and the RESTful services without being transferred through server-side SOAP services.

V. CONCLUSION AND FUTURE SCOPE

The request and response messages of the invocations of RESTful services do not need to be transferred via server-side SOAP services. Extended BPEL engine with adapters allow direct invocation and composition of SOAP, RESTful and OSGi services with heterogeneous content type.

As per current system it generates less network traffic and spends less turnaround time than those of the traditional approaches. With the extended BPEL engine based on Adapter pattern, SOAP, RESTful and non-Web services can be integrated in well manner.

In future, the engine can be considered as a more suitable framework for composing heterogeneous services than the traditional ones from the perspective of resource consumptions in mobile environments.

REFERENCES


