This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication. The final version of record is available at http://dx.doi.org/10.1109/TSC.2014.2310213

1

A Framework for Composing SOAP, Non-SOAP and Non-Web Services

Jonathan Lee, Shin-Jie Lee, and Ping-Feng Wang

**Abstract**—Recently, there is a trend on developing mobile applications based on service-oriented architecture in numerous applicationdomains, such as telematics and smart home. Although efforts have been made on developing composite SOAP services, little emphasis has been put on invoking and composing a combination of SOAP, non-SOAP, and non-web services into a composite process to execute complex tasks on various mobile devices. Main challenges are two-fold: one is how to invoke and compose heterogeneous web services with various protocols and content types, including SOAP, RESTful, and OSGi services; and the other is how to integrate non-web services, like Web contents and mobile applications, into a composite service process. In this work, we propose an approach to invoking and composing SOAP, non-SOAP, and non-web services with two key features: an extended BPEL engine bundled with adapters to enable direct invocation and composition of SOAP, RESTful and OSGi services based on Adapter pattern; and two transformation mechanisms devised to enable conversion of Web contents and Android activities into OSGi services. In the experimental evaluations, we demonstrate network traffic and turnaround time of our approach are better than those of the traditional ones.

**Index Terms**—Heterogeneous service composition, web service, service composition engine.

✦

**1 INTRODUCTION**

Recently, there is a trend on developing mobile appli-cations based on service-oriented architecture (SOA) in numerous application domains, such as telematics [1], business [2], smart home [3][4] and internet of things (IOT) [5]. In IOT, for example, heterogeneous web ser-vices are supposed to be integrated to provide composite services. These heterogeneous services may include Sim-ple Object Access Protocol (SOAP) [22] services, RESTful services [29], and OSGi services [30]. SOAP is a proto-col specification for exchanging structured information during the implementation of web services, and SOAP services can be employed to build composite web ser-vices using Business Process Execution Language (BPEL) [6]. RESTful services are also considered an architectural style that can be used to construct software for clients to request services. An increasing number of Internet services have been developed as RESTful services to provide easier interfaces for displaying object features and communicating with external services. OSGi tech-nology provides an open service platform for service in-stallation, activation, and management in devices. OSGi services support point-to-point remote service delivery programs that enable dynamical binding, assembly, and execution of device services.

Although efforts have been made on developing com-posite SOAP services [6][7], little emphasis has been put on invoking and composing a combination of SOAP,

*Jonathan Lee is with Department of Computer Science and Information En-gineering, National Taiwan University, Taiwan, email:jlee@csie.ntu.edu.tw. Shin-Jie Lee is with Computer and Network Center, National Cheng Kung University, Taiwan, email: jielee@mail.ncku.edu.tw. Ping-Feng Wang is with Department of Computer Science and Information Engineering, National Central University, Taiwan, email: 93542011@cc.ncu.edu.tw*

non-SOAP, and non-web services into a composite pro-cess in order to complete complex tasks on a variety of mobile devices. The challenges can be best explained from the following two perspectives:

* How to invoke and compose heterogeneous web services with various protocols and content types, such as SOAP, RESTful, and OSGi services? A com-posite process may involve invocations of SOAP, RESTful and OSGi services over SOAP protocol, HTTP and Java method call, respectively. In addi-tion, it also involves a variety of message content types, such as SOAP, JSON, YAML, Protocol Buffer and Java objects, in composing the heterogeneous web services.
* How to integrate non-web services, including Web contents and mobile applications, into a composite web service process? Non-web services in a mobile environment typically consist of Web contents and mobile applications. Web contents are usually in the form of HTML documents. The program exe-cution entry points of mobile applications are usu-ally developed as GUI components, e.g., Activities in Android platform. Both of these two non-web services are difficult to be composed by a service composition engine, like a BPEL engine.

In this work, we propose a framework for invok-ing 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

Copyright (c) 2014 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication. The final version of record is available at http://dx.doi.org/10.1109/TSC.2014.2310213

2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | **BPEL Engine** | |  |  |  |
|  |  |  |  |  |  |  |
|  | **Adapters** |  |  |  |  |  |
|  |  |  |  |  |  |  |
| **SOAP Service** | SOAP |  |  |  |  |  |
| Service |  |  |  |  |  |
|  |  |  |  |  |  |
| **Non SOAP Service** | RESTful | OSGi | | **Transformation** | |  |
| Service | Service | | **Mechanisms** | |  |
|  |  |
| **Non Web Service** |  |  |  | Android | Web |  |
|  |  |  |  | Application | Content |  |

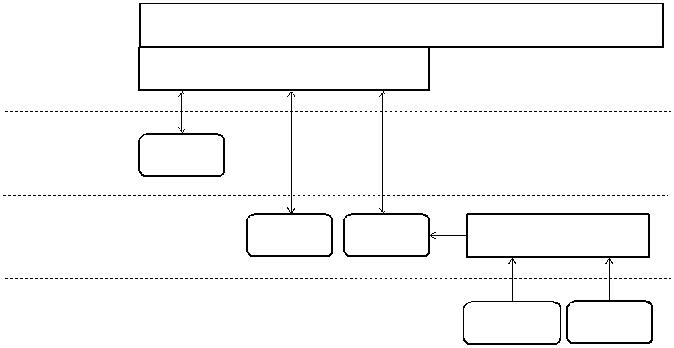


Fig. 1. Conceptual model of the framework for invoking and composing SOAP, non-SOAP and non-web services

services without modifying the core BPEL engine; and two transformation mechanisms devised to enable the transformation of Web contents and Android activities into OSGi services that can be composed by the extended BPEL engine.

Figure 1 shows the conceptual model of the frame-work 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.

This paper is organized as follows. We discuss the related work in Section 2. Section 3 describes fully the proposed framework. In Section 4, we conduct two experimental evaluations. Finally, in Section 5, we sum-marize the contributions of the proposed framework.

**2 RELATED WORK**

It is widely accepted that combining multiple web ser-vices into a composite service is more beneficial to users than finding a complex and preparatory atomic service that satisfy a special request [26], [27], [28], [18], [17]. The resulted composite services can be used as atomic services by themselves in other service compositions to satisfy clients requests. BPEL4WS [6] provides a mixture of block-structured and graph-structured process mod-els, and variables associated with message types can be specified as input or output variables to invoke, receive, and reply web services.

Recently, BPEL has been extended to support model-ing the composition of heterogeneous web services, such as RESTful and OSGi services. REST [29] (REpresenta-tional State Transfer) is a style of software architecture for distributing hypermedia systems such as the World Wide Web. REST defines a set of architectural principles [29] by which one can design web services that focus on a system’s resources, including how resource states are addressed and transferred over HTTP by a wide range of clients written in different languages.

In [8], F. Curbera et al. offer a service-composite development model for composing RESTful services. During the invocations of RESTful services, the messages returned from a service are stored as BPEL process variables. However, how to transform the messages to different content types for follow-up service invocations is unclear.

In [16], K. He proposes a composite system for in-tegrating both SOAP and RESTful services with a hy-brid orchestration based on a BPEL engine and a REST orchestration engine. The composite service workflow is divided into two kinds of sub-workflows according to the types (SOAP or RESTful) of services to be com-posed. However, the message transformation between the SOAP and RESTful is not discussed.

In [11], S. Farokhi. et al. propose a framework, called MDCHeS, to support dynamic composition and to use both SOAP-based and RESTful Web services simultane-ously in composite services with three different views: data, process, and component view. However, the ap-proach, like the other approaches, does not discuss the message transformation between the SOAP and RESTful.

In [33], J. Nitzsche et al. extended BPEL 2.0 with a WSDL-less interaction model, call BPEL*light*, to enable

coupling business logic and Web service technology, including WSDL, by introducing a new and single type of interaction activity resuming all BPEL interaction activities. The main focus of the paper is to enable modeling processes or process fragments that can be reused and bound to specific service interfaces in any interface description language. Our approach, however, focuses more on how to model and implement the binding relationships among heterogeneous services.

In [31][32], C. Pautasso proposed a process-based com-position 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 ex-ecuting a composite process. Furthermore, SOAP, JSON, YAML, Protocol Buffer, and Java object messages re-turned 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.

**3 HETEROGENEOUS SERVICE COMPOSITION**

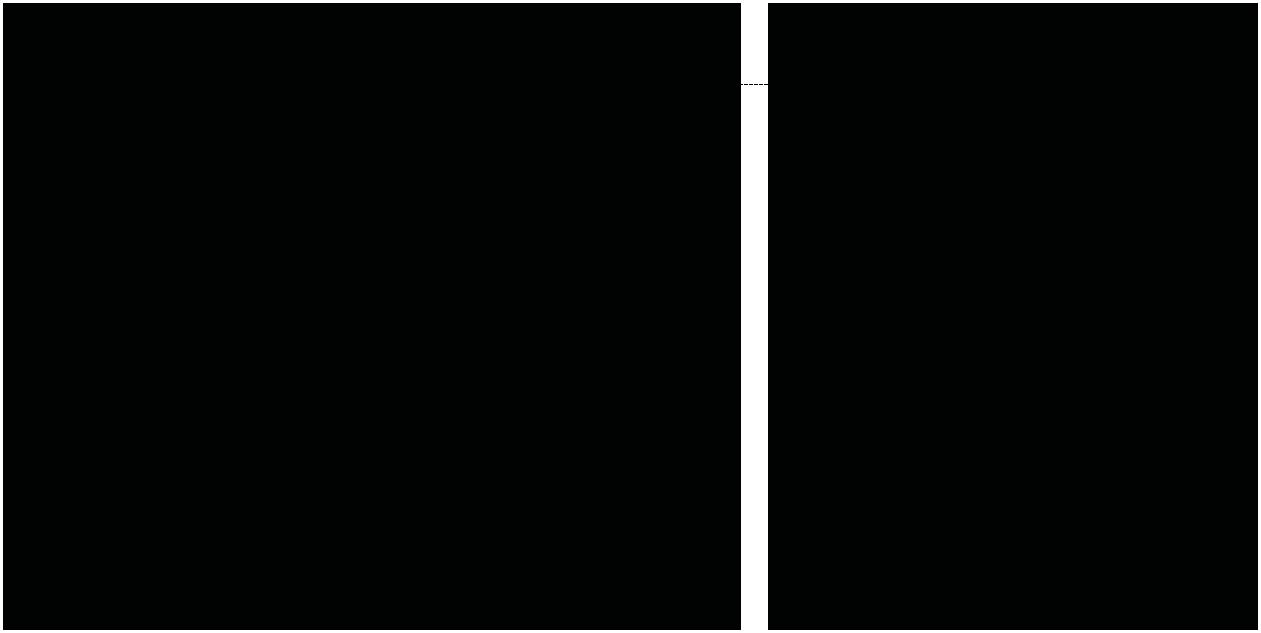
**FRAMEWORK**

In this work, an extended BPEL engine is developed by applying Adapter pattern [21] to compose SOAP, RESTful and OSGi services. Figure 2 shows the system architecture of the heterogeneous service composition framework. A concise description of our BPEL engine is given below without going into details of its modules.

Copyright (c) 2014 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication. The final version of record is available at http://dx.doi.org/10.1109/TSC.2014.2310213

3



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | *Partner Link Handler* | | | | |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |
|  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | | | |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  | |  | | |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  | |  | | |  |  |
|  |  |  |  |  |  |  |  | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  | |  | | |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  | | |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | | | |  |  |  |  | |  |  |  |  | | |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  | |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  | |  |  |  | |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | | | | | | |  |  |  |
|  |  |  |  |  |  | | | | | |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  | | |  |  |  |  |  | | | |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | | | |  |  |  |  |  | | | | | | | |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | | |  |  |  | |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Fig. 2. The system architecture of the heterogeneous service composition framework

The adapters and transformers will be further elaborated in the sections 3.2, 3.3, 3.4 and 3.5.

The entry point of the BPEL engine is Composite Service Activator component with inputs of WSDL and BPEL documents. Deploy component deploys the docu-ments 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 en-codes control-flow logic and can contain other basic or structured activities recursively. Thread Manager com-ponent handles the multi-threads based on XPath and XQuery.

SOAP, RESTful, and OSGi Adapters are responsible for invoking SOAP, RESTful, and OSGi services, respec-tively. 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 Android APKs into OSGi services, and Web Content Asset Trans-former transforms web contents into OSGi services.

Our BPEL engine, complied with BPEL version 2.0, is implemented in Java with 44 packages, 426 classes, 3750 methods and a total of 93609 lines of code. The adapters and transformers are implemented with 11 packages, 105 classes, 461 methods, and 6453 lines of code.

**3.1 Experimental Scenario**

In this work, a smart living control system with a number of heterogeneous services is developed as the experimental environment for comparing our proposed approach with the traditional approach. The system obtains sensor data in order to calculate the predicted mean vote (PMV) indicator for temperature, humidity, illuminance, etc. The PMV system is supplemented with HVAC (heating, ventilation, and air conditioning) tech-nology, lighting, and motorized roller blinds to provide an energy-saving living environment. We develop two smart living system scenarios by adopting the traditional approach and the proposed approach (see Figure 3 and Figure 4).

In the traditional approach (Figure 3), RESTful services and Web contents are wrapped as SOAP services and deployed in the server side. As for the Android activity running on Android platform, they can not be directly invoked by the BPEL engine. Hence, a SOAP service is required as a proxy for retrieving the data generated from the activity. The steps of composing the services are as follows:

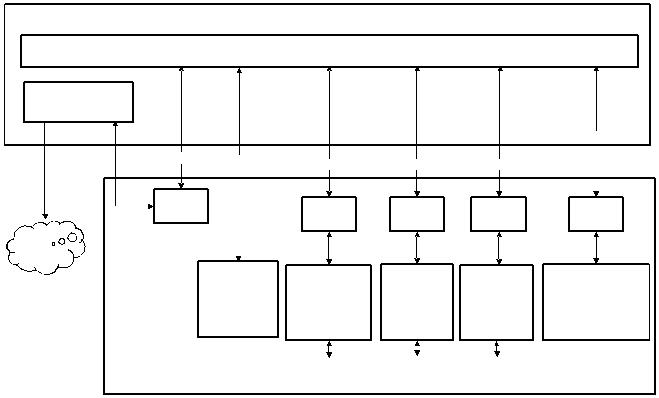
1. Steps 1-3: The web information about current weather information including temperature, hu-midity, sunlight, and illumination of the area are retrieved through an Android activity. After that, the activity sends the weather data to a SOAP service. The BPEL engine will retrieve the data through invoking the SOAP service.
2. Step 4: The BPEL engine sends the weather data to the calculated PMV SOAP service. Based on the temperature, humidity, and illumination value, the optimal parameter values for the HVAC technol-ogy, lighting, and motorized roller blinds will be determined and returned to the BPEL engine.
3. Steps 5-7: The RESTful services of Motorized roller

Copyright (c) 2014 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication. The final version of record is available at http://dx.doi.org/10.1109/TSC.2014.2310213

4

**Client Side (Mobile Device)**



BPEL Engine

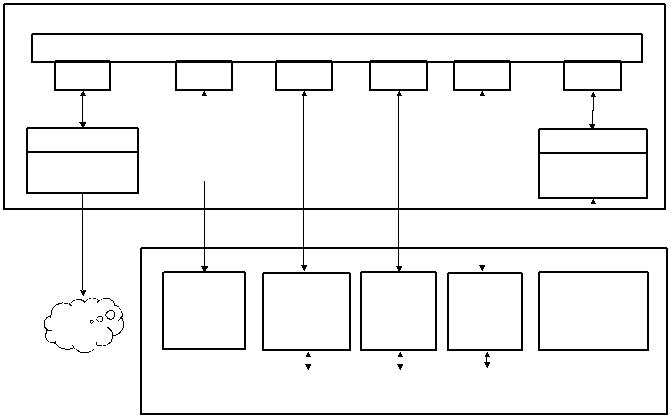
Weather Data

Android Activity

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| SOAP | SOAP | SOAP | SOAP | SOAP | SOAP |  |
|  |  |
|  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | SOAP | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | SOAP | | | SOAP | | | SOAP | | |  | SOAP | | |  |  |
|  |  | Service | |  |  |  |  |
|  |  |  | Service | | | Service | | | Service | | |  | Service | | |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | | |  |  |  |  |  |  |  |  |  |  | |  |
|  |  |  |  |  |  | **HTTP** |  |  | HTTP |  |  | HTTP |  |  |  | HTTP |  | |  |
|  |  |  |  |  | Motorized | | | Lighting | | | HVAC | | |  |  |  |  |  |  |
| weather , |  |  |  | Calculate | Recommendations | | | | |  |
|  |  |  | roller blinds | | |  |
| temperature, |  |  |  | PMV | controller | | | Controller | | | Controller | | | on energy saving | | | | |  |
| lumen..and so on. |  |  | SOAP service | | RESTful | | | RESTful | | | RESTful | | | info. Web page | | | | |  |
|  |  |  |  |  | service | | | service | | | service | | |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | **Server Side** | | | | Electrical roller | | | Lightings | | | HVACs | | |  |  |  |  |  |  |
|  | blinds | | |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Fig. 3. The heterogeneous service compositions sce-nario by the traditional approach



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Client Side (Mobile Device)** | | | | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  | | |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  | Extended BPEL Engine | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  | | |  |  |  |  |  |  | | |  |  |  | | |  |  |  |
|  |  | Adapter | | |  |  | Adapter | | |  |  | Adapter | | | | Adapter | | | | Adapter | | | |  | Adapter | | | | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | OSGi |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | OSGi |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | OSGi service | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | OSGi service | | | | | |  |
|  |  | Weather Data | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | Energy saving info. | | | | | |  |
|  |  | Android Activity | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | Web Content | | | | | |  |
|  |  |  |  |  |  |  |  |  |  | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | |  | |  |
|  |  |  |  |  |  |  | SOAP |  | HTTP |  |  |  | HTTP |  |  |  | HTTP |  |  |  | Packaged as | | |  | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | Calculate | | | |  | Motorized | | |  | Lighting | | |  | HVAC | | |  | Recommendations | | | | | |  |
|  |  |  |  |  |  |  |  | roller blinds | | |  |  |  |  |
|  |  |  |  |  |  |  |  |  | Controller | | |  | Controller | | |  |  |
|  |  |  |  |  |  |  | PMV | | | |  | controller | | |  |  |  | on energy saving | | | | | |  |
|  |  |  |  |  |  |  |  |  | RESTful | | |  | RESTful | | |  |  |
|  |  |  |  |  |  | SOAP Service | | | | |  | RESTful | | |  |  |  | info. Web page | | | | | |  |
|  |  |  |  |  |  |  |  | service | | |  | service | | |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  | service | | |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | weather , | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | temperature, | | |  |  |  |  |  |  |  | Electrical roller | | | | Lightings | | | | HVACs | | | |  |  |  |  |  |  |  |
|  |  | lumen..and so on. | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | **Server Side** | | | | | | | blinds | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Fig. 4. The heterogeneous service compositions sce-nario by our approach

blinds controller, Lighting Controller, and HVAC Controller will be invoked with the parameters of the optimal parameter values by the SOAP ser-vices.

1. Step 8: The BPEL engine invokes a SOAP service to collect the energy saving information. The SOAP service will parse the web page of the information and return the result back to the BPEL engine.

In our proposed approach (Figure 4), the scenario in-cludes an extended BPEL engine bundled with adapters that enable directly invocations of the heterogeneous services in the experimental environment (see Figure 4). The steps of composing the services are as follows:

1. Step 1: As the Weather Data Android Activity is transformed into an OSGi bundle through the mo-bile applications transformer, the extended BPEL engine can directly invoke the OSGi service to retrieve the weather information.
2. Step 2: The extended BPEL engine sends the weather data to the calculated PMV SOAP service. Based on the temperature, humidity, and illumina-tion value, the optimal parameter values for the HVAC technology, lighting, and motorized roller blinds will be determined and returned to the extended BPEL engine.
3. Steps 3-5: The RESTful services of Motorized roller

<wsdl:service name="WeatherService"> <wsdlrestful:address protocol="http" format="json"

url="http://query.yahooapis.com/v1/public/yql?q=\*" /> </wsdl:service>

Fig. 5. An example of describing a RESTful service

blinds controller, Lighting Controller, and HVAC Controller will be directly invoked with the pa-rameters of the optimal parameter values by the extended BPEL engine.

* 1. Step 6: The energy saving information web page will be transformed into an OSGi bundle through the web content asset transformer, and to be com-posed by the extended BPEL engine through the OSGi adapter.

1. **Composing RESTful Services**

* Protocol: The protocols that a RESTful service sup-ports, e.g., HTTP.
* Message Format: The message formats that a REST-ful service supports, e.g., JSON, YAML, and protocol buffer.
* Url: The URL of a RESTful service.

The extension of WSDL for describing RESTful ser-vices includes protocols, urls, and message formats. Fig-ure 5 shows an example of describing a Weather RESTful service. The service supports HTTP protocol and is with the content format of JSON.

Figure 6 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 *<*wsdlrestful:address*>* 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 Bufffer to Java object module based on open source org.json and org.ho.yaml packages. Messages are con-verted into JSON, YAML or Protocol Buffer messages before they are sent as inputs to another web service.

1. **Composing OSGi Services** 
   * type: The type that an OSGi service locates, e.g., local.
   * serviceName: The service name that an OSGi ser-vices registers in the OSGi register.
   * filter: The filter is an optional property that identifies a concrete OSGi service.

In the framework, we extend WSDL to describe OSGi services including their types, service names and filters.

Copyright (c) 2014 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.