Enriching OSGi Service Composition with Web Services

Choonhwa LEE†, Sunghoon KO†, Eunsam KIM††, Nonmembers, and Wonjun LEE†††, Member

SUMMARY This letter describes combining OSGi and Web Services in service composition. According to our approach, a composite service is described in WS-BPEL. Each component service in the description may be resolved to either an OSGi service or Web Service at runtime. The proposal can overcome current limitations with OSGi technology in terms of its geographical coverage and candidate service population available for service composition.

key words: SOA, service composition, OSGi, dynamic Web service proxy, dynamic service binding

1. Introduction

Service-Oriented Architecture (SOA) is widely recognized as a viable solution to cope with the dynamics and complexity inherent in the future computing environments. The SOA paradigm supports the process of service publishing, discovery, and binding by which desired application functionalities are mapped to appropriate service(s) available in the environments. It further leads to a composite service that can be formed by internetworking component services to provide basic functionality. With the initial but meaningful rollouts of SOA technologies such as OSGi and Web Services in recent years, the issue of heterogeneous service composition poses itself as one of the most urgent to be addressed. This is because the number of participating services from a particular technology at the early stage is likely limited to some extent. Immediate but enormous benefits would be made possible, if we could tap into other worlds for component services.

This letter presents our approach to service composition support in OSGi-based environments with emphasis on interbred composition and dynamic binding.

2. Combining OSGi Services with Web Services

Originally introduced as a home gateway platform to provide a managed service execution environment, OSGi [1] is now being considered as one of the most prominent technologies to enable the SOA paradigm. However, Web Service is never less than OSGi in terms of technology potential and deployment. Having noticed their influential position, we decided to take advantage of combining the two leading technologies. This letter starts off by bringing up a service composition scenario, before getting into the details of how to enable the combination of OSGi and Web Services. The scenario serves to guide the discussion of our proposal all the way from research motivation to architectural design to prototyping.

2.1 Service Composition Scenario

First, let us consider an exemplary scenario of automatic utility bill payment in Fig. 1 that intends to highlight the benefit from the interbred composition of OSGi and Web Services. A task “automatic utility payment” on To-do List service takes care of paying the utility bill every month. On a designated day, each metering service reads the usage of gas, water, and electricity for the month, and sends the meter readings to Utility Billing Service at the utility company. Then, the company calculates a utility bill for the usage to charge Bob. The payment is handled by Payment Service at Bob’s bank.

The scenario involves service compositions between the two kinds of OSGi and Web Services. The top four services (i.e., To-do List, Gas Meter, Water Meter, and Wattmeter) are OSGi services residing on Bob’s home gateway, while the bottom two (i.e., Utility Billing Service and Payment Service) are Web Services hosted on the Web servers of the utility company and bank. The application for the scenario can be constructed by assembling these component modules of two different types.

The scenario emphasizes the benefit and possibility
from the crossbred service composition. Without such support, a resultant composition would fall short of what is needed for the automatic utility payment. In other words, we will end up having two fragmented compositions, one for each domain, which requires some follow-up efforts to splice the two. Moreover, the heterogeneous composition should widen the pool of candidate services that can participate in the composition. The interbred composition would achieve better results in application functionality and performance. In addition, the scenario implies that the confined coverage of OSGi domains can be effectively extended beyond LAN environments by allowing Web Services to join the compositions.

2.2 Architectural Design for Interbred Compositions

Figure 2 diagrams our architectural design to enable the power combination of OSGi and Web Services. Based on several relevant technologies such as OSGi, Web-Addressing [4], and WS-BPEL, we propose a new architecture that advances the state of the art in the interbred service composition for OSGi environments. Two key ideas of it are the dynamic Web Service proxy and the dynamic service binding that are supported by Web Gateway Service and Plumber Service in the figure, respectively.

First, a service composition is specified using WS-BPEL, and the description is passed down to Plumber Service in charge of overseeing the composite service buildup process. Plumber Service is designed to realize a composite service comprising OSGi and Web Services based on the WS-BPEL description.

The figure shows major steps involved in the composition process. Given a service composition description from applications above, Plumber Service has to instantiate the composite by interconnecting component services using OSGi Wire-backed pipes. If a component service in question is of OSGi type, the service registry is searched for, and the discovered service become part of the composite service. If it is a Web Service, Web Gateway Service is requested to generate an OSGi proxy for the service. Based on the service WSDL description, it creates a proxy on the fly and registers the proxy with the service registry. After then, the proxy joins the composition.

2.2.1 Dynamic Web Service Proxy

Web Gateway Service retrieves a service’s WSDL file pointed to by Plumber Service, and parses it to get information necessary to generate its proxy, including service name, interface, service URL, encoding type, method names and variables, etc. The proxy represents the Web Service functionality to the OSGi domain, having itself to become part of service compositions on OSGI frameworks. In order to interface with its corresponding Web Service as a client, it speaks SOAP protocol. As depicted in Fig. 3, the proxy takes a dual role of a producer and consumer, requiring pipe binding support. The producer pushes a data item by invoking update() method on an OSGi Wire-backed pipe object which in turn calls updated() method on the consumer side. In reverse direction, data can be pulled by making a call to poll() method on the pipe that triggers the producer’s polled() method.

The proxy consists of three sub-components: OSGi wiring module, message translation module, and Web Service interface module. Neighbor services are bound to the proxy’s OSGi wiring module through the pipe connection. Any pipe interaction is translated into a SOAP interaction by the message translation module on the proxy. The invocation is then passed on to the proxy’s Web Service interface module which has been built on Axis Call class [5] in order to support dynamic invocation interface. The Call object’s properties, including service name, port type, service URI, encoding type, operation name and parameters, and so on, are set to represent a Web service in question. A response to the Web Service invocation is converted back to OSGi data format to propagate to the consumer.

2.2.2 Dynamic Service Binding

Static service binding requires that the address of services be determined ahead of time, and it is often considered as a limitation. Recognizing the need of dynamic binding, WS-Addressing [4] introduces the notion of an endpoint reference to dynamically identify and describe service endpoints. The scheme allows for choosing one among multiple candidate services or even for defining a service endpoint at runtime. Current WS-BPEL allows an actual partner service to
be dynamically determined through the endpoint reference. Furthering this dynamic partner link binding, we propose a new scheme referred to as dynamic service binding that utilizes OSGi service registry for dynamic generation of service endpoint descriptions. Consequently, a service can now be resolved to even an OSGi service.

At start, a service is indicated by an endpoint reference which is going to be fully specified later on. According to the dynamic partner link scheme, the endpoint reference is resolved to an actual service endpoint identified by service name and address information from the WSDL file pointed to by the process’s descriptor. We extend the scheme so that a reference may be resolved to a service instance discovered from OSGi service registry. (Hence, the name “dynamic service binding”.) More specifically, an endpoint reference in a WS-BPEL description is made complete with endpoint information retrieved from OSGi service registry.

Figure 4 illustrates the binding steps. Encountering an endpoint reference (step 1), the process checks partner links (step 2). Based on the binding information such as service name and address from the WSDL file pointed to by the process descriptor (step 3 & 4), an actual endpoint can be determined for an operation invocation. Our dynamic service binding makes use of OSGi service registry if allowed (step 4’). As shown in Fig. 5, either of an OSGi and Web Service may be permitted. OSGi services are preferred over their Web Service equivalent in our current prototype, when both service implementations are available. This default choice can be changed by policy settings. Given the descriptor in the figure, Plumber Service looks up services in OSGi service registry by calling get_Service_Reference() method, and loops through the service list to see if there is one with its org.osgi.service.PartnerLinkType property equal to BankTransferService.

2.3 Prototype Implementation

Our prototype has been built on top of Knopflerfish 2.05 framework [3] and Active-BPEL 5.0 [6] along with Tomcat 5.5. In addition, Apache Axis 2 [5] is used for SOAP messaging.

Figure 6 shows a simplified BPEL process description for the utility bill payment scenario presented in Fig. 1. The process starts off with the user bank account information passed to in AutoPaymentVar. Then, a service associated with To-Do List partner link is invoked to check whether which utility payment is due. After calculating the charge, a payment service of BankTransferRequest partner link type is called. It is noted that our proposal of dynamic service binding is used for this service.

The screen shot in Fig. 7 is the service composer tool of our prototypical implementation that enables service compositions. Shown on the left-hand side is a graphical representation of the composite service being composed, while the right-hand side displays detailed information about the composition itself and participating services.
3. Related Work

The latest OSGi specification offers some level of support for service composition through Wire Admin Service. The system service defines a set of APIs that can be used to wire a pair of services. Beyond that, further efforts were made to provide system support for service dependency management and automatic service composition on OSGi platforms [2], [7]. However, their concern was confined to the service composition among the same breed, i.e., between OSGi services. With the first realistic rollouts of SOA platforms underway, the need and benefit of interactions between heterogeneous services has grabbed the attention of the research community. For example, a Web Service can be provided on top of OSGi platform by Knopflerfish Axis port [3], bridging OSGi and Web Services domains. The proxy scheme is on an individual service basis, while our architecture is geared towards the service composition of multiple, heterogeneous services.

The adoption of WS-BPEL as composition language on OSGi platforms is also found elsewhere [8]. One main difference is that their efforts aimed at the purebred composition of OSGi services, while our primary concern is the internetworking of OSGi and Web Services.

Finally, there are other noteworthy efforts to bridge OSGi with such SOA platforms as SCA and Spring framework. The most prominent examples include Apache Tuscany and Newton component model.

4. Conclusion

With the increasing rollout of SOA technologies, the issue of service composition deserves our attention despite whether it is either an inter- or intra-platform composition. This letter is about integrating the two most prominent SOA technologies: OSGi and Web Services. The dynamic Web Service proxy allows a Web Service to transparently participate in OSGi service compositions. Also, our dynamic service binding scheme enables a service to be resolved to either an OSGi or Web Service. By proposing an approach to tap into the reservoir of richer Web Services, our scheme will be able to accelerate the deployment of SOA platforms and services.

Acknowledgments

This work was supported by the Korea Science and Engineering Foundation (KOSEF) grant funded by the Korea government (MEST) (No. R01-2008-000-10692-0).

References