LETTER

Cross-Domain Service Composition in OSGi Environments

Choonhwa LEE[†], Seungjae LEE[†], Eunsam KIM^{††}, *Nonmembers*, and Wonjun LEE^{†††a)}, *Member*

SUMMARY This letter presents a new approach to provide inter-domain service compositions for OSGi environments. Our proposal of remote wire objects extends OSGi's wiring capability across the framework boundaries, so that even remote services can join in the composition. Hence, a better composition is made possible with a richer set of candidate services from foreign domains.

key words: service composition, OSGi, cross-domain service composition, remote wire object

1. Introduction

Originally introduced as a home gateway platform, OSGi Service Platform [1] provides a managed service execution environment on which services can be dynamically installed, invoked, and then uninstalled. By promoting dynamic discovery and collaboration of various devices and services through the standard execution environment and agreed-upon service interfaces, it is now being considered as one of the most promising technologies to build SOA (Service-Oriented Architecture) systems. An OSGi service platform is designed to serve a community of service providers and clients within a rather clear boundary. This implication that an OSGi-supported community is more or less confined to the framework boundary may not have been an issue in self-sufficing networking scenarios like home networks. However, as OSGi positions itself as one of the mainstream infrastructure solutions for SOA systems, the assumption does not hold any longer. Therefore, the technology should be extended to enable service exchanges across the domain boundary so as to benefit from inter-framework service interactions.

This letter presents our approach to support service compositions across OSGi frameworks. We first review the state of the art in OSGi service composition technology, before discussing our proposal and architectural design of cross-domain service composition.

2. OSGi Service Composition Technology

OSGi has seen remarkable growth over the past few years with some advanced features, including service composition, service availability monitoring, and service dependency management, being added to the specification. But as far as cross-domain service interactions are concerned, the latest OSGi specification does not do enough, leaving the need of inter-framework service discovery, invocation, and composition untouched.

2.1 Wire Admin Service

OSGi's Wire Admin Service introduces Wire object that can be used to model data flow between a producer and consumer service. As illustrated in Fig. 1, an application module (i.e., Composer) calls the *createWire()* method of Wire Admin Service with producer and consumer services specified in the arguments, which results in a Wire object being created. A data item can be pushed by the producer's call to *update()* method on the Wire object which in turn triggers *updated()* method on the consumer object. In the opposite direction, data items can be pulled by *poll()* and *polled()* methods defined on the Wire object and producer, respectively.

Since OSGi's Wire object just offers the bare minimum to interconnect services, further supports are necessary to deal with the intricacy involved in service composition process.

2.2 Plumber Service

Along with Wire Admin Service, the current OSGi specification offers some level of supports for service availability monitoring and composition. For example, Service Tracker Service keeps track of changes to service availability, and Declarative Service is capable of managing dependencies

Fig. 1 Producer-consumer pairing of OSGi Wire object.

Manuscript received September 10, 2008.

Manuscript revised January 10, 2009.

[†]The authors are with the College of Information and Communications, Hanyang University, Seoul 133-791 Korea.

^{††}The author is with the Department of Computer Engineering, Hongik University, Seoul 121-791 Korea.

^{†††}The author is with the Department of Computer Science and Engineering, Korea University, Seoul 136-701 Korea. (corresponding author)

a) E-mail: wlee@korea.ac.kr

DOI: 10.1587/transinf.E92.D.1316

among services. Plumber Service (i.e., a central piece to our previous work to develop a lightweight, failure-resilient service composition middleware architecture [2]) is built on them in order to automate service composition process. A composite is described in XML as an internetworking of component services, each containing its own service logic. The components are interconnected via a pipe that is backed by OSGi's Wire object. Given a composite description, Plumber Service can not only provide an optimal composition solution. Once successfully built, it also takes care of the maintenance need of the composite such as partial replacements or failure recovery to adapt to changes to the environment.

3. Inter-Domain Service Composition

The aforementioned technologies may be able to provide adequate support for local service composition within a single OSGi framework. But they are of no help, when it comes to inter-framework service federation support. The discussion on our inter-domain service composition support starts off by bringing up a remote service composition scenario.

3.1 Remote Service Composition Scenario

We consider a digital diary scenario to take note of activities in our daily life. As illustrated in Fig. 2, the scenario involves three OSGi frameworks for a home network (Home Servers), a personal network (Smart-phone), and a vehicle area network (Automobile). In the figure, clouds represent the three domains, and a circle corresponds to a service located in the area. The Digital Diary service is able to record all sorts of events during the course of the day captured by various sensors such as cameras and GPS units. For example, a picture taken by the Camera service on the phone can be associated with a travel plan placed on the person's schedule. In addition, the travel is tracked by the GPS sensor in the car. The Activity Tracer service at home collects all these events from foreign frameworks, so that it can correlate them for the diary. The scenario emphasizes the need of service compositions spanning multiple OSGi frameworks.

Fig. 2 Digital diary scenario.

3.2 Remote Wire Object

OSGi's Wire Admin Service deals with only local compositions by means of Wire objects as a pipe between producer and consumer services. We extend the Wire object for the purpose of effectively enabling distributed service compositions. More specifically, the extended Wire object now spans two frameworks, interconnecting a pair of services residing in each different domain. The extension is referred to as a Remote Wire object. As we can see in Fig. 3, OSGi Wire object is extended to inherit *java.rmi.Remote* interface in order to handle remote exceptions because of inter-domain service interactions. It is important to know that our extension makes the remote wiring transparent to applications whether it is within a single domain or between different domains.

3.3 Distributed Service Composition Architecture

Our architectural design for inter-domain compositions is depicted in Fig. 4. At the heart of the architecture is Plumber Service that has been re-architected to provide cross-domain composition support by means of the Remote Wire object.

Service Composer is an application module that allows users to create a composite by specifying component wirings. The composition task is assisted by Service Discovery module that supplies candidate component services

```
public interface Wire extends Remote {
    public void update( Object value ) throws RemoteException ;
    public Object poll() throws RemoteException;
    public Class[] getFlavors() throws RemoteException;
    public Object getLastValue() throws RemoteException;
    public Dictionary getProperties() throws RemoteException;
    public String[] getScope() throws RemoteException;
    public boolean hasScope(String name) throws RemoteException;
    public boolean isConnected() throws RemoteException;
    public boolean isValid() throws RemoteException;
}
```

Fig. 3 Remote Wire definition.

Fig. 4 Remote service composition architecture.

discovered from either local or foreign frameworks. According to a composition graph from Service Composer, Plumber Service does the plumbing by creating a Remote Wire object. Notice that the wiring is supported by OSGi's Wire Admin Service, and no change to its APIs is made. Event Synchronizer is responsible for ensuring producer and consumer frameworks are in synchronization regarding the remote composition.

A remote wiring to a foreign framework is set up in the following steps. First, becoming aware that a remote wire is needed between itself and the producer framework, Plumber Service on the consumer framework side instantiates an object of Remote Wire class, and registers its stub with a local RMI registry. The proxy object can then be downloaded to the producer side, which occurs as a result of RMI registry lookup. This lookup is performed using *wirePID* as the key that includes *producerPID*, *consumerPID*, and additional information. Finally, the producer Plumber Service completes the wiring by calling the *createWire()* method of Wire Admin Service.

To assist in managing remote service compositions, Plumber Service keeps track of a list of remote services sitting at the opposite end of Remote Wire objects along with the status of the wiring. In order to ensure that both ends are on the same page regarding the status of a remote composition, service composition events should be relayed to and replayed on the other side as if it occurred locally. As a result, it remains transparent to the composer application whether it is either a local or remote composition. For that purpose, OSGi Wire events need be re-defined to accommodate remote compositions, just as the Wire object has been extended. The semantics of the events are extended as follows, to fit the need of event synchronization between the two parties.

- WIRE.CREATED, WIRE.DELETED, and WIRE.UPDATED events are sent to the other end, when a Remote Wire object is created, deleted, and updated, respectively.
- WIRE.CONNECTED event is generated, when both the producer and the consumer become interconnected and ready for data exchange. WIRE.DISCONNECTED event occurs when the pipe wiring breaks. Based on these two events, our Plumber Service is able to keep track of the availability of services at the other end of the wire.
- WIRE.TRACE event signifies data exchanges between the producer and the consumer via *update()* method.
- CONSUMER.EXCEPTION and PRODUCER.EXCEPTION events are used to communicate exceptions or unrecoverable errors on either side of the wire.

The discovery module is responsible for cross-framework service discovery. In addition to service lookups in a local OSGi registry, it should be able to search for remote services from foreign domains. However, this discovery issue is orthogonal to the cross-framework composition problem we tackle in this work. It is noted that some

works on cross-domain service discovery are reported elsewhere [3], [4]. When complemented with adequate discovery support, our remote composition architecture will be able to provide a viable solution.

4. Prototype Implementation

To demonstrate the effectiveness and feasibility of our approach to the problem of remote compositions, we have developed a prototype of our remote service composition architecture based on Knopflerfish 2.0 [5]. Figure 5 shows a screen shot of the prototype running in Home Server domain. On the left of the window is a list of OSGi frameworks, including itself and two neighbor frameworks of SmartPhone and Automobile. Resident services are shown under each framework. Producer and consumer services of a pipe can be specified by clicking on service names, which is shown in the middle. The pipe is then incorporated into the service composition graph on the right. Once the composite description gets completed, its actual assembly can be started anytime by Compose menu.

To assess performance impact from our remote composition support, we compare start-up time for both local and remote composition cases. We measure OSGi framework's initial start-up time, service installation time, including system services and producer/consumer services, and wiring object creation time. The result is plotted in Fig. 6, as we vary the number of the wiring from 1 to 10.

The x-axis represents the number of wirings. The instance of zero wiring serves as the base case to assess the wiring overhead. As the number of the wirings increases, we can see a faster rise in the case of remote compositions. There is an increase on average of 0.6 second per a Remote

Fig. 5 Screen shot of service composer.

Fig. 6 Performance comparison of local and remote wires.

Wire object, which accounts for the time for registering the wire's stub object with a RMI registry and downloading it onto the opposite-side framework.

5. Related Work

Various approaches to OSGi service compositions have been explored over the past years. Service Binder [6] and Plumber Service [2] are prominent among the local composition efforts for a single OSGi domain. It has also been demonstrated that WS-BPEL can be adopted as a service composition language for OSGi environments [7]. But the work was confined to local compositions as well.

On the other hand, there exist other efforts to push the boundary of OSGi technology beyond a single domain. OSGi support for service interactions across the platform boundaries is under consideration as a part of the next version of the specification, which is named as the Distributed OSGi [9]. The crux of it is to enable remote service discovery and access beyond the framework boundary. It permits a service to be published and discovered across the framework instances, and automatically generates a proxy for the remote service. As a result, an application developer will be able to make use of the service, as if it were a local service, preserving the OSGi service programming model based on a common service contract shared by clients and service providers. A service contract consists of service interface and associated properties. In addition to the service contract-based interaction model, OSGi also includes support for the producer-consumer pattern-based interaction model in the form of Wire objects, as we already know. Our proposal of Remote Wire objects extends the original wiring capability to accommodate multi-framework environments. Therefore, the distributed OSGi and our Remote Wire should be considered as a complementary effort to enable cross-domain service interaction.

Besides, the possibility and benefits of integrating OSGi with other SOA technologies like SCA and Spring framework are actively being explored. For instance, an OSGi-SCA binding [8] enables internetworking between OSGi services and SCA components, meaning that OSGi services can access SCA components regardless of whether they are OSGi or non-OSGi components. A similar approach has been taken by Newton component model

where OSGi containers are bridged by SCA wiring technology. Our Remote Wire object-based composition approach differentiates itself from those efforts in that it is geared towards a lightweight composition solution for OSGi environments.

6. Conclusion

This letter presents a new distributed service composition architecture capable of supporting cross-domain wirings for OSGi environments. At the heart of it is our proposal of Remote Wire objects that extends OSGi's wiring capability beyond a single framework. The extension allows producer and consumer services to be linked together across the framework boundaries, effectively removing the domain barrier for remote service compositions.

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

- [1] OSGi – The Dynamic Module System for Java, <http://www.osgi.org>
- [2] C. Lee, S. Ko, S. Lee, W. Lee, and S. Helal, "Context-aware service composition for mobile network environments," Proc. 4th International Conference on Ubiquitous Intelligence and Computing, July 2007.
- [3] S. Czerwinski, B. Zhao, T. Hodes, A. Joseph, and R. Katz, "An architecture for a secure service discovery service," Proc. 5th IEEE MobiCom, Aug. 1999.
- [4] C. Lee, J. Yi, and W. Lee, "Bridging OSGi islands through SLP protocol," Proc. 4th International Conference on Ubiquitous Intelligence and Computing, July 2007.
- [5] Knopflerfish Open Source OSGi, <http://www.knopflerfish.org/>
- [6] H. Cervantes and R. Hall, "Autonomous adaptation to dynamic availability using a service-oriented component model," Proc. 26th International Conference on Software Engineering (ICSE'04), 2004.
- [7] R. Redondo, A. Vilas, M. Cabrer, J. Arias, and M. Lopez, "Enhancing residential gateways: OSGi service composition," IEEE Trans. Consum. Electron., vol.53, no.1, pp.87–95, Feb. 2007.
- [8] Tuscany Project, <http://incubator.apache.org/tuscany/>
- [9] OSGi Alliance, OSGi Service Platform Release 4 Version 4.2 - Early Draft, <http://www.osgi.org/download/osgi-4.2-early-draft.pdf>, Aug. 2008.