Towards Runtime Adaptation in a SOA Environment

Florian Irmert, Marcus Meyerhöfer, Markus Weiten

Friedrich-Alexander University of Erlangen and Nuremberg
{Florian.Irmert,Marcus.Meyerhoefer}@cs.fau.de,markus@weiten.de

Abstract. Service Oriented Architecture (SOA) promotes the utilization of available services to develop completely new applications in a context which has not been foreseen as these services were implemented. Unfortunately the interfaces respectively the behaviour of a service often do not fit exactly to the new domain. Slight changes would be necessary to reuse them in the new environment. This paper presents an approach to integrate dynamic AOP into a SOA platform to adapt existing services at runtime to new requirements. Services can then be reused without the need of stopping and redeployment.

1 Introduction

Service Oriented Architecture (SOA) has recently gained widespread attraction because of its promise to allow for easier and more flexible adaptations of the software infrastructure of a company to fast changing business requirements. The core idea of SOA is to decouple functional units and expose them as independent services to other programs and thereby foster reuse. Moreover, this allows to create new applications or services by merely combining already available ones in new ways.

However, in a business environment there are of course some services which have been build specifically for concrete applications (e.g. client management). For such services the interfaces are usually designed without reuse in mind but are determinated by the requirements of the service users; often service user and service provider design the interfaces in a collaborative process. In order to be usable by new services or applications, those company internal services would have to be adapted. In modern environments with high demands on application availability this leads to the necessity to modify running services. It is not viable to take a service offline and deploy an updated version as possibly many other services depend on such a service. Furthermore, it might also happen that different services have different or even conflicting requirements on a service they use. In such cases specific modifications should be applied for each consumer individually.

In this paper we present an approach to adapt services at runtime. We examine this problem from a technical point of view[1]. Runtime adaptation [1,2,3] can be achieved with dynamic Aspect Oriented Programming (d-AOP) [4] and therefore we have integrated a d-AOP framework into a praxis proven SOA environment, the OSGi Service Platform.

2 Technical Background

In the next Section we give a short introduction to the OSGi Service Platform and d-AOP.

[1] Semantical issues—e.g. which modifications should be disallowed in order to ensure correct application behaviour—are beyond the scope of this paper and area of future work.

OSGi Service Platform The OSGI Service Platform is described as a ”Java based application server for networked devices, however small or large they are” [5]. Originally designed for embedded devices and home service gateways, it has become prominent for building SOA applications. Several applications (deployed in a special bundle format) can coexist inside the OSGi Service Platform, while each is loaded by a different class loader. Consequently only one Java Virtual Machine is needed. Also lifecycle management is supported for all hosted applications; there is an API to install, start, stop and de-install the application bundles without restarting the framework. This ”Hot-Deployment” feature is very interesting in the context of SOA, because adding new services to the platform does not affect running services. Bundles can provide their functionality as a service by publishing their interfaces in the OSGi Service Registry. Other bundles employ this registry to discover and bind services (fig. 1). While there are a number of implementations of the OSGi Specification [6,7] we decided to use Equinox [8], published by the Eclipse Foundation, because it is a well proven implementation which offers all necessary features for our approach (Section 4).

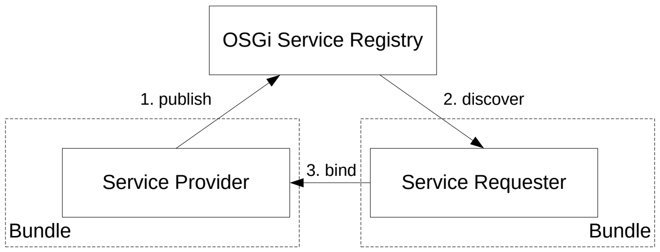

Fig. 1. OSGi Service Registry

Dynamic AOP The term ”dynamic aspect-oriented programming” is most commonly used if aspects can be deployed and activated at runtime. Dynamic AOP can be realized e.g. with a modified JVM [9] or bytecode modification [10]. We decided to use JBoss AOP in our approach, because its successful application is exemplified by its usage in the JBoss Application Server. JBoss AOP inserts hooks at potential joinpoints. Each time such a hook is called in the program flow, a central Aspect Manager is called (fig. 2). This instance manages the aspects and decides whether to apply them depending on the joinpoint. Obviously new aspects can be added to the Aspect Manager at runtime.

3 Problem Domain

Figure 3 illustrates a typical scenario where available services are used to build new services respectively applications. The new service **E** utilizes available services **A** and **C**. Often services cannot be used exactly as they are. A common approach is to implement wrappers to adapt their behaviour. As a simple example service **E** in figure 4 uses service **A** but has to transform the result from miles to kilometers. A wrapper class performs the necessary transformation. If there are many services which want to use service **A**, but also require kilometers instead of miles, it would be desirable to enhance the interface of **A**. Sometimes it would also be desirable to modify service **A** (slightly) to deliver information which is readily available to the service,

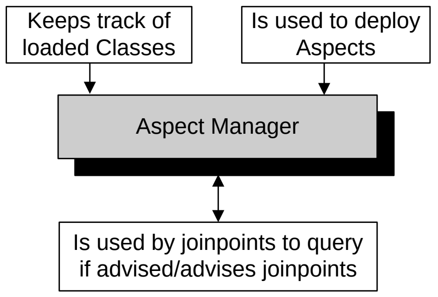
Keeps track of loaded Classes
Is used to deploy Aspects
Aspect Manager
Is used by joinpoints to query if advised/advises joinpoints

Fig. 2. Central Aspect Manager instance

but not exposed yet. In the example of figure 3 this would be no problem, because service **A** is used only by service **E**. Service **A** could therefore be taken offline and replaced by a modified version without affecting other services. Unlike service **A**, the adaptation of service **C** would be much more difficult, because service **D** depends on service **C**, too. If service **C** is stopped to apply code changes for adaptation, service **D** would be perturbed as well. A runtime mechanism is needed to enable modifications of the behaviour of a service "online". Runtime enhancement of services allows to adopt existing services to unforeseen requirements. Therefore it facilitates modifications of services just-in-time and without interference of other services which depend on services under modification.

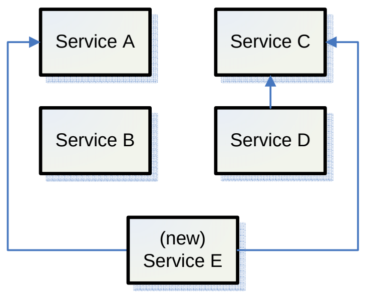

Fig. 3. SOA example

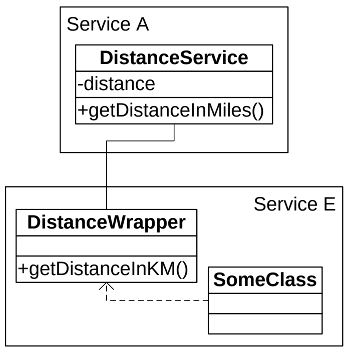

Fig. 4. Usage of a wrapper class

Previtali [11] exploits aspects for dynamic updates and employs AOP's features like method or field interception. With d-AOP these modifications can be applied at runtime. The integration of d-AOP into a SOA framework constitutes the technical basis for adaptations of running services to new system conditions as well as changing business requirements. The main challenge addressed in this paper is to integrate d-AOP seamlessly to ease the development process of service oriented applications.

4 Integration of JBoss AOP into the Equinox OSGi Framework

In our approach we integrate the JBoss AOP framework into the OSGi Service Platform to realize dynamic adaptation at runtime. To provide an integrated environment it is necessary to deploy the aspects as OSGi bundles. Therefore the deployment must be mapped to OSGi lifecycle operations. Each bundle contains a bundle activator, which is invoked when the bundle is started and stopped. The bundle activator implements an interface defining two callback methods (start/stop). We utilize these methods for deploying and undeploying the aspects by invoking the corresponding methods of the Aspect Manager. Deploying/undeploying aspects is mapped to installing/starting and stopping/uninstalling the corresponding bundle that encapsulates the aspects. Other projects [12,13] have also integrated AOP into OSGi deploying aspects as bundles, but do not support dynamic AOP.

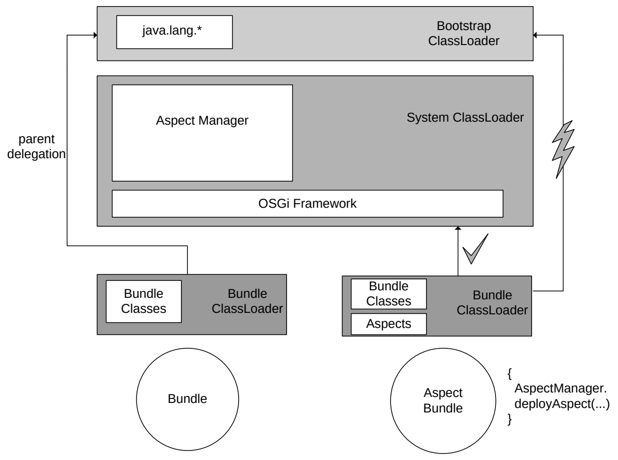

Fig. 5. Solving the class loader problem

OSGi makes use of different class loaders to realize the complete separation of bundle classpath and namespaces. By using different class loaders, bundles can encapsulate classes with identical names and naming conflicts are avoided. Different bundles can coexist, without interfering each other, but this complicates the integration of JBoss AOP. In Java a symbolic reference is loaded by the same class loader which has loaded the defining class. The bundle activator would not be able to access the Aspect Manager, because the bundle activator is always defined by the bundle class loader.

In our prototype we use the Equinox OSGi Framework [8], which allows to define the parent class loader for bundles. We solved the class loader problem by defining the system class loader as parent class loader. Everytime a class is going to be loaded, the class loader initially delegates the search for the class to its parent class loader. By default, the parent class loader for an OSGi bundle is the bootstrap class loader containing all `java.lang.*` classes. This is sufficient for all ordinary OSGi bundles (see figure 5 left side). But an aspect bundle must have access to the methods of the global Aspect Manager. The delegation to the bootstrap class loader would prohibit that access, because the Aspect Manager is loaded and defined by the system class loader. That is why parent delegation has to be reconfigured to the system class loader as parent for the OSGi bundle class loader (see figure 5 right side).

Frei and Alonso present in [14] an approach to integrate a d-AOP framework, which uses dynamic proxies to implement the aspects, into the OSGI Service Platform. They modified the OSGi API and the class loading of the used d-AOP framework (to solve the class loader problem). In contrast to their approach, we do not change the API and we are able to deploy the aspects as bundles, which we consider of utmost importance for a seamless integration.

5 Conclusion

This paper presented an approach to integrate JBoss AOP (which supports d-AOP) into the OSGi Service Platform—an open, modular and scalable SOA environment—represented by Equinox. Deploying and undeploying aspects is mapped to OSGi bundle installation and deinstallation. With our integration of the d-AOP framework, adaptation with respect to a changing environment can be achieved at runtime without stopping or redeployment of active services. In this paper the technical aspects of a seamless integration have been presented. Currently, we are working on semantic problems arising when d-AOP is applied and plan to evaluate our framework with a case study.

References

1. Cámara, J., Canal, C., Cubo, J., Rodriguez, J.M.M.: An Aspect-Oriented Adaptation Framework for Dynamic Component Evolution. In Cazzola, W., Chiba, S., Coady, Y., Saake, G., eds.: RAM-SE, Fakultät für Informatik, Universität Magdeburg (2006) 59–70
2. Greenwood, P.: Dynamic Framed Aspects for Dynamic Software Evolution. In Cazzola, W., Chiba, S., Saake, G., eds.: RAM-SE, Fakultät für Informatik, Universität Magdeburg (2004) 101–110
3. Liu, R., Gibbs, C., Coady, Y.: MADAPT: Managed Aspects for Aynamic Adaptation based on Profiling Techniques. In: ARM '04: Proceedings of the 3rd Workshop on Adaptive and Reflective Middleware, New York, NY, USA, ACM Press (2004) 214–219
4. Popovici, A., Gross, T., Alonso, G.: Dynamic weaving for aspect-oriented programming. In: AOSD '02: Proceedings of the 1st International Conference on Aspect-Oriented Software Development, New York, NY, USA, ACM Press (2002) 141–147
5. OSGiAlliance: About the OSGi service platform: Technical whitepaper. `http://www.osgi.org/documents/collateral/TechnicalWhitePaper2005osgi-sp-overview.pdf` (November 2005)
6. Knopflerfish: Knopflerfish OSGi homepage. `http://www.knopflerfish.org/` (March 2007)
7. Apache.org: Apache Felix homepage. `http://cwiki.apache.org/FELIX/index.html` (March 2007)
8. Eclipse Foundation: Equinox OSGi Framework Homepage. `http://www.eclipse.org/equinox` (March 2007)
9. Popovici, A., Alonso, G., Gross, T.: Just-In-Time Aspects: Efficient Dynamic Weaving for Java. In: AOSD '03: Proceedings of the 2nd International Conference on Aspect-Oriented Software Development, New York, NY, USA, ACM Press (2003) 100–109
10. Vasseur, A.: Dynamic AOP and Runtime Weaving for Java - How does AspectWerkz Address It? `http://aspectwerkz.codehaus.org/downloads/papers/aosd2004-daw-aspectwerkz.pdf`, AOSD 2004 International Conference on Aspect-Oriented Software Development, Invited Industry Talk (March 2004)
11. Previtali, S.C.: Dynamic Updates: Another Middleware Service? In: MAI '07: Proceedings of the 1st Workshop on Middleware-Application Interaction, New York, NY, USA, ACM Press (2007) 49–54
12. Lippert, M.: AJEER: An AspectJ-Enabled Eclipse Runtime. In: OOPSLA '04: Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications, New York, NY, USA, ACM Press (2004) 23–24
13. Webster, M.: Equinox Incubator: Aspects and OSGi. `http://www.eclipse.org/equinox/incubator/aspects/index.php` (February 2007)
14. Frei, A., Alonso, G.: A Dynamic Lightweight Platform for Ad-Hoc Infrastructures. In: PERCOM '05: Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications, Washington, DC, USA, IEEE Computer Society (2005) 373–382