Integration of Dynamic AOP into the OSGi Service Platform

Florian Irmert
University of Erlangen-Nuremberg
Department of Computer Science
Computer Science 6 (Data Management)
Martensstrasse 3
91058 Erlangen, Germany
florian.irmert@cs.fau.de

Frank Lauterwald
University of Erlangen-Nuremberg
Department of Computer Science
Computer Science 6 (Data Management)
Martensstrasse 3
91058 Erlangen, Germany
frank.lauterwald@cs.fau.de

Matthias Bott
University of Erlangen-Nuremberg
Department of Computer Science
Computer Science 6 (Data Management)
Martensstrasse 3
91058 Erlangen, Germany
mail@matthiasbott.de

Thomas Fischer
University of Erlangen-Nuremberg
Department of Computer Science
Computer Science 6 (Data Management)
Martensstrasse 3
91058 Erlangen, Germany
thomas.fischer@cs.fau.de

Klaus Meyer-Wegener
University of Erlangen-Nuremberg
Department of Computer Science
Computer Science 6 (Data Management)
Martensstrasse 3
91058 Erlangen, Germany
kmw@cs.fau.de

ABSTRACT

The ability to adapt to different computing environments or external changes is an important requirement for both stationary and mobile computing. Without this ability, all requirements have to be foreseen, which is often not possible in practice. Classical software engineering approaches often lead to redeployment or even software migration. Therefore loosely coupled software design and a dynamic adaptation model are required. Dynamic aspect-oriented programming (d-AOP) in conjunction with service oriented programming is well suited to face this demand. One well known approach providing a service-oriented component model is the OSGi Service Platform.

This paper introduces our approach to combine an OSGi Framework with d-AOP to establish dynamic adaptation of core concerns as well as crosscutting concerns. Seamless integration of current d-AOP frameworks is managed by mapping aspect deployment and undeployment to OSGi bundle lifecycle operations without affecting the deployment model. Unlike former proposals, this approach retains the strict separation of bundles as mandated by OSGi.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures; D.1.5 [**Programming Techniques**]: Aspect-Oriented Programming; D.3.3 [**Programming Languages**]: Language Constructs and Features

General Terms

[Design, Languages]

Keywords

[aspect-oriented programming, OSGi Framework, dynamic adaptation, aspect weaving, modularization, crosscutting concerns, aspect mechanism, software components, reuse]

1. INTRODUCTION

Middleware platforms like JEE or .NET have seen widespread adoption in the industry to build robust and flexible software applications. They provide services, e.g. for security, logging, transaction etc., which are transparently invoked during the execution of the business logic. This separation of concerns is a proven design principle for modern software architectures. But these middleware platforms often have only a predefined set of services available. Eichberg and Mezini [5] identify this lack of openness as a problem and introduce the combination of middleware and aspect-oriented programming (AOP) [9].

The architecture of middleware systems also struggles between generality and specialization [19]. Vendors offer support for many application domains resulting in a comprehensive set of features. Therefore these systems often exhibit large resource demands. The additional services can be seen as orthogonal requirements with respect to the core functionality of middleware systems. Orthogonal implementations have the drawback that it is not possible to decouple them completely from the rest of the business logic code.

In contrast to the business logic, which can be modularized with patterns like "layers" [3], orthogonal implementations are scattered throughout the code resulting in tangled code that is excessively difficult to maintain [9].

A well-known technique to decouple these so-called cross cutting concerns is AOP. We share the view of Zahn and Jacobsen [19] that middleware architecture is one of the ideal places to apply AOP methods to obtain a modularity level that cannot be obtained with traditional programming techniques. An analysis of modularity in aspect-oriented design can be found in [12].

In particular domains like e.g. embedded devices, real time applications etc. it is necessary for the developer to have an optimized architecture. Services which are not used should not be deployed, because otherwise they would occupy resources. Dynamic AOP (d-AOP) extends the original notion of AOP by allowing the integration of aspects at runtime [15, 2, 17]. Although some dynamic behaviour can be achieved by static weaving, d-AOP is more powerful in the sense that weaver targets can be declared at runtime [17]. D-AOP is receiving growing interest for the creation of adaptive software.

In the last years nearly every vendor enlarged its middleware system to a SOA-platform (service-oriented architecture). SOA is a recent effort to foster the reuse of software. One platform for building SOA-applications that has become prominent recently is the OSGi (Open Services Gateway Initiative) platform. Especially since being used as the core of Eclipse 3, this platform has become popular. Originally designed for embedded devices and home service gateways its footprint is very small compared to "traditional" middleware platforms. The use on mobile devices etc. requires the possibility to adapt the behaviour of an application to different environments. Due to memory limitations it is not always possible to deliver a fully featured device. Therefore it is desirable to update the device at runtime triggered by changing circumstances. With its small footprint (about 800 kb) and the possibility to add software components at runtime the OSGi framework is suitable for such environments.

In [7] we introduced an approach for the integration of d-AOP into the OSGi framework resulting in a SOA platform where environment specific requirements can be modified and/or updated at runtime. This paper enhances our previous work by providing a more flexible deployment model. As OSGi allows running different applications at the same time, we extended the definition of pointcuts, enabling us to install application specific aspects. To show the viability of this approach, we have integrated JBoss AOP [8] into the Equinox OSGi framework [4] in a prototype implementation.

The remainder of this paper is organised as follows: In Section 2 the OSGi framework as well as the basics of aspect-oriented programming are introduced. Section 3 explains the difficulties in integrating d-AOP and OSGi as well as our solution to these problems. In Section 4 related work is shown and briefly discussed. We conclude the paper in Section 5.

2. TECHNICAL BACKGROUND

In this Section we give a short introduction to the OSGi Service Platform and d-AOP before we summarize our previous work on the integration of these two technologies.

2.1 OSGi Service Platform

The OSGI Service Platform is described as a "Java based application server for networked devices, however small or large they are" [13]. Originally designed for embedded devices and home service gateways, it has become prominent for building service-oriented applications. Several applications respectively services can coexist inside the OSGi Service Platform. In OSGi the deployment unit is called *bundle* and the bundles are seperated from each other by loading them with different class loaders. Since Java 1.2 a type of a class is defined by its name (e.g. `fau.test.HelloWorld`) and additionally the class loader. Therefore it is possible to deploy classes with the same name in different bundles without interfering with each other. 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. While there are a number of implementations of the OSGi Specification [10, 1], we decided to use Equinox [4], published by the Eclipse Foundation, because it is a well proven implementation which offers all necessary features for our approach (Section 3).

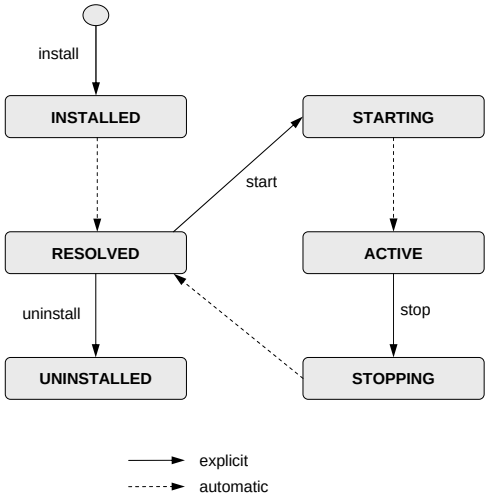

Figure 1: Bundle lifecycle

2.2 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 [14] or bytecode modification [17]. We decided to use JBoss AOP in our approach, because its successful application is exemplified by its usage in the JBoss Application

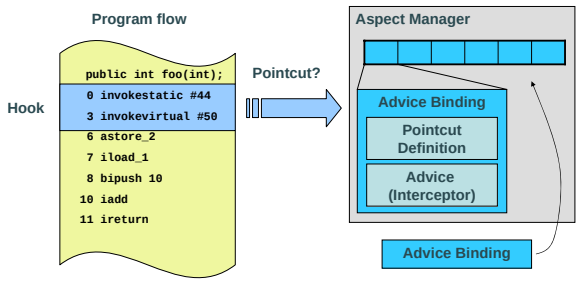

Figure 2: Central Aspect Manager

Server. JBoss AOP inserts hooks at potential joinpoints at loadtime. Each time such a hook is reached in the program flow, a central Aspect Manager is invoked (fig. 2), which manages the aspects (advice bindings) and decides whether to apply them depending on the pointcut definition. The Aspect Manager provides a method to add new advice bindings (pointcut definition + advice) at runtime.

2.3 Previous Work

In [7] we introduced our approach to combine the OSGi framework with dynamic aspect-oriented programming in order to realize dynamic adaptation at runtime. The basic idea is to deploy aspects as OSGi bundles. Thus we mapped the deployment/undeployment of an aspect to the OSGi lifecycle operations install/start and stop/uninstall. The OSGi core specification defines a bundle activator class for each bundle which is executed when the bundle is started and stopped. We utilized the start/stop method for deploying/undeploying the aspects by invoking the appropriate methods of the Aspect Manager. For this integration no changes on the source of the OSGi respectively the d-AOP implementation were necessary.

In this paper we show an enhancement to our approach by extending the pointcut definition to achieve a bundle aware definition of aspects. In [7] it is not possible to specify which bundles should be affected by an aspect. An aspect is executed at every joinpoint which is defined within a pointcut. The definition of different aspects for classes with the same name (deployed in different bundles) is not provided. To support such a definition, an extension to the pointcut definition is necessary.

3. BUNDLE-AWARE INTEGRATION OF D-AOP INTO THE OSGI FRAMEWORK

In this section we present our integration of d-AOP into the OSGi framework. At first we identify the requirements. After a short discussion about an alternative solution, we present our realization. The specific examples make use of the JBoss AOP and the Equinox OSGi framework.

3.1 Requirements

Our requirements for the integration were:

1. Aspect deployment: Aspects should be deployed as bundles. The OSGi deployment model must not be altered.
2. No restriction for bundles: The "normal" OSGi bundles should not have to be prepared. It should be possible to install aspects also onto third-party bundles and there should be no limitation to possible joinpoints.
3. Non invasiveness: We do not want to make changes to the used OSGi or the d-AOP frameworks. Otherwise the implementation of our integration code would need to be changed every time a new version of the OSGi framework or the d-AOP framework implementation is shipped.
4. Bundle aware deployment: It should be possible to specify a set of bundles where the aspect should be installed. This necessitates an enhancement of the pointcut definition with respect to bundle names.

In our previous work [7] we have implemented the first three requirements. The challenge in our new approach is to enhance the pointcut definition of a d-AOP framework to specify the affected bundles while remaining non-invasive.

Normally a pointcut matches joinpoints like a regular expression matches strings. Due to the fact that each bundle is loaded with its own class loader in OSGi, it is possible that classes with the same name and the same package structure can coexist inside the OSGi framework (e.g. different versions of the same library). If a joinpoint in a bundle is selected with a "normal" pointcut definition, the advice would be woven into all versions of that bundle. Therefore it is necessary to specify the bundle name additionally. A possible definition for the bundle name is given in the code example in listing 1, which uses a similar syntax to JBoss AOP.

```
String[] bundles =
  new String[]{"fau.test.firstbundle"};
ScopedBinding sbinding =
  new ScopedBinding(bundles,
                ("execution(POJO->new(..))",
                TracingAdvice.class);

AspectDeployment deployer =
  AspectDeployment.instance();
deployer.deploy(sbinding);
```

Listing 1: Code Example

In this example the names of the bundles that are to be affected by an advice are put into a String array. The aspect is defined by a so called *ScopedBinding* with the parameters: bundles, pointcut, and advice. Then the ScopedBinding is registered with the AspectManager (in this example called *AspectDeployment*).

3.2 Obvious Solution

The obvious solution is an extension of the Aspect Manager. As shown in figure 3 the hook asks the Aspect Manager if it is adviced and the decision depends on the extended pointcut definition, which consists of the "normal" pointcut and the bundle name. It would be necessary

- that the hook delivers the name of the bundle to the Aspect Manger and

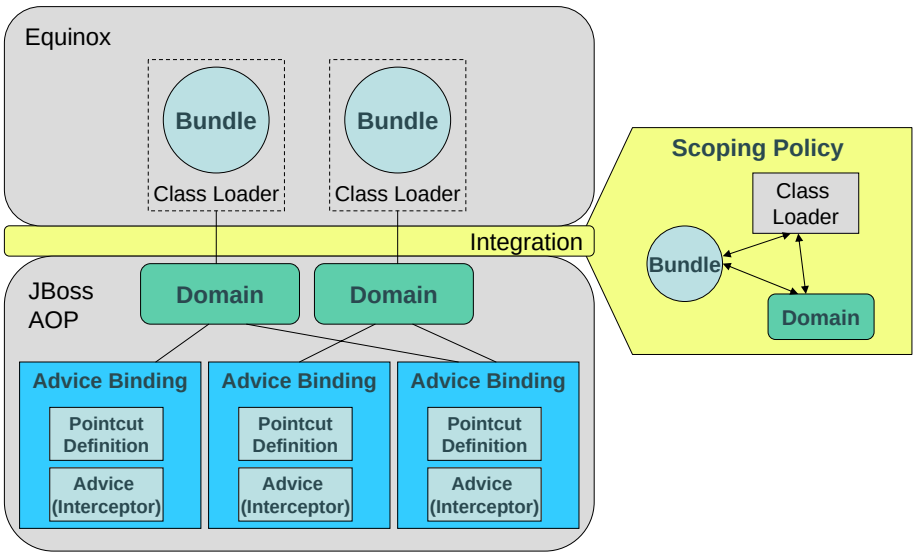

Figure 4: Integration architecture

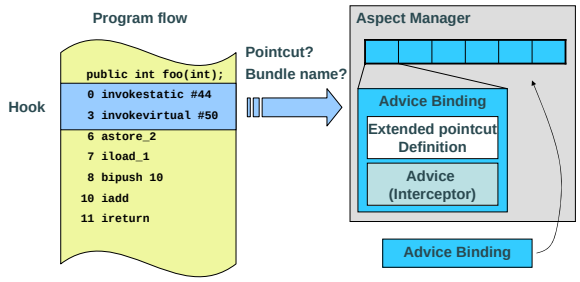

Figure 3: Extended pointcut definition

- that the advice binding can handle the extended pointcut definition.

Therefore it would be inevitable to change the source code in JBoss AOP. But one of our design goals was not to change the source code of the OSGi or the d-AOP framework. To achieve this goal it is necessary to develop the enhancement on top of these frameworks.

3.3 Realization

JBoss AOP was originally designed for use inside the JBoss Application Server. To run different applications inside the application server while preventing them from interfering with each other, JBoss also separates the applications by loading them with different class loaders. To install different aspects for these applications, it is possible to manage multiple Aspect Managers (called Domains) inside JBoss AOP, whereby a Domain is simply a specialization of the Aspect Manager and each Domain instance is responsible for a set of class loaders.

In our prototype we have implemented a layer on top of JBoss AOP to manage a Domain for each bundle. The aspects are assigned to the different domains depending on the bundle definition in the `ScopedBinding` (listing 1). Figure 4 shows the architecture of our integration concept. The correlation between Domain and class loader is defined in a "scoping policy" (`AOPClassLoaderScopingPolicy`), which can be set at the central Aspect Manager (fig. 5).

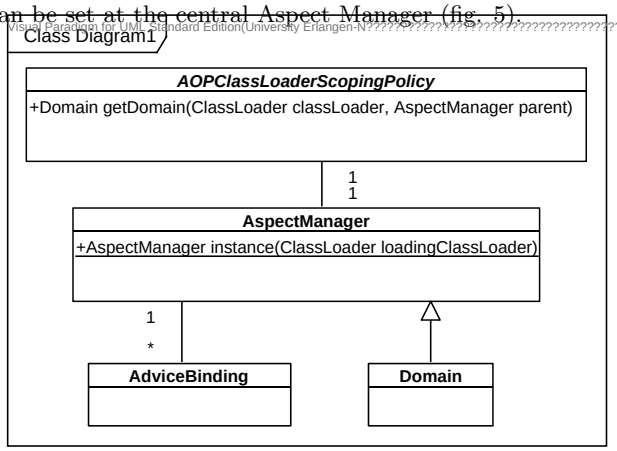

Figure 5: JBoss AOP AspectManager

Each JBoss AOP hook is bound to a specific domain, depending on the scoping policy. When a hook is reached the first time it invokes the factory method `instance()` in the Aspect Manager and passes its own class loader. The factory method returns the appropriate Domain. Because the hook passes only its own class loader as argument to the `instance()` method, the scoping policy has to decide which Domain should handle the advice bindings for that specific hook on the basis of the class loader. The "default" domain is the central Aspect Manager itself.

To be able to specify the bundle name in the `ScopedBinding` it is necessary to establish a binding between bundle names, class loaders and domains in our integration layer. This is done by mapping bundle names to Domains and mapping bundle names to class loaders.

To establish the mapping between bundle name and Domain the event system of OSGi is used. During an *BundleChanged* event that is thrown by the OSGi framework everytime before it starts a bundle the following tasks are executed:

1. Register Domain: The bundle name and the corresponding Domain are registered in the integration layer.
2. Aspect deployment: All previously installed aspects for that bundle are registered at the Domain.

There are two reasons for installing all aspects at the corresponding Domain when the bundle is started: If a bundle is updated (uninstalled and then reinstalled with a new version) the class loader for this bundle changes and a new Domain is created for this class loader. All aspects that affect the bundle have to be installed at this new Domain. The second reason is the deployment of aspects for uninstalled bundles. It is possible to install aspects for bundles which are not available at aspect installation time, just by specifying a bundle name that does not exist. If a bundle with a corresponding name is installed later, the aspects have to be added to the newly created Domain.

The mapping between the class loader and the Domain is established while a bundle is loaded. In Equinox it is possible to intercept the class loading with a so-called ClassLoadingHook. We provide such a *ClassLoadingHook*, which performs two tasks.

1. Bytecode instrumentation: It invokes the bytecode instrumentation for the JBoss AOP hooks. Within this extension the hooks are inserted into the bundles at loadtime. Bundles do not have to be aware that they will be aspectized in any way.
2. Register class loader: The class loader of the bundle is registered in our integration layer with the corresponding bundle name.

Utilizing the OSGi event system and a *ClassLoadingHook* the mapping between bundle name, class loader and Domain is established respectively updated every time a new bundle is installed in the OSGi framework without the need to change the source code of JBoss AOP or the Equinox OSGi implementation.

4. RELATED WORK

This section presents other approaches which combine the benefits of OSGi and aspect-oriented programming.

4.1 AJEER

Martin Lippert presented his "AspectJ Enabled Eclipse Runtime" (AJEER) [11] a few years ago. AJEER was originally designed to integrate AspectJ [16] into the Eclipse framework. When AJEER was implemented, Eclipse did not yet make use of the OSGi framework. The weaving part of the AspectJ 1.2 compiler implementation was added to the Eclipse runtime. Although Lippert discusses the options of AJEER being extended to support runtime weaving, it is currently limited to load-time weaving. Additionally he presents the idea of "runtime-like" weaving: Aspects can be added to bundles at runtime and are activated when the bundle is restarted. Currently this option is not implemented either. Since Eclipse 3 is based upon the OSGi framework, AJEER has been reengineered to support the integration of AspectJ into the new Eclipse kernel. Therefore AJEER can be used to write aspects in AspectJ as deployable OSGi bundles. The use of load-time weaving however hinders the integration of aspects into already running bundles.

4.2 Jadabs

In [6] Frei and Alonso present an approach to integrate a d-AOP framework, which uses dynamic proxies to implement the aspects, into the OSGI Service Platform. These proxies replicate the bundle interfaces. Therefore it is not possible to add aspects inside the bundles. 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. We also facilitate the installation of aspects inside a bundle, even if it is a third-party bundle.

There are also approaches for integrating AOP into other middleware systems [18, 5]. Nevertheless all of them have in common that they (i) do not support dynamic AOP and that they (ii) rely on own implementations. In contrast, our approach uses a well proven middleware framework which can be combined with popular d-AOP implementations.

5. CONCLUSION

This paper presented our proposal for an integration layer that combines dynamic aspect-oriented programming and OSGi while retaining the OSGi deployment model. We extended the pointcut definition to restrict the scope of aspects to a subset of bundles by specifying the affected bundles by their names. Our proof-of-concept implementation is based on the Equinox OSGi implementation and the JBoss AOP framework. The integration is accomplished through an additional layer that maps between bundle names, class loaders and AOP domains. This layer is purely implemented as an extension to JBoss AOP and Equinox, i.e. without changing code or even APIs of the existing frameworks. Aspects and "normal" bundles may be deployed and undeployed in any order. Currently we evalute the performance and the handling of our approach in a real world case study.

6. REFERENCES

[1] Apache.org. Apache Felix homepage. `http://cwiki.apache.org/FELIX/index.html`, March 2007.

[2] J. Bonér. AspectWerkz - Dynamic AOP for Java. `http://codehaus.org/~jboner/papers/aosd2004aspectwerkz.pdf`, October 2003.

[3] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. John Wiley & Sons, August 1996.

[4] Eclipse Foundation. Equinox OSGi Framework Homepage. http://www.eclipse.org/equinox, March 2007.

[5] M. Eichberg and M. Mezini. Alice: Modularization of middleware using aspect-oriented programming. In T. Gschwind and C. Mascolo, editors, *Software Engineering and Middleware: 4th International Workshop, SEM 2004*, volume 3437, pages 47–63, Linz, Austria, March 2005. Springer-Verlag.

[6] A. Frei and G. Alonso. A Dynamic Lightweight Platform for Ad-Hoc Infrastructures. In *PERCOM '05: Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications*, pages 373–382, Washington, DC, USA, 2005. IEEE Computer Society.

[7] F. Irmert, M. Meyerhöfer, and M. Weiten. Towards Runtime Adaptation in a SOA Environment. In W. Cazzola, S. Chiba, Y. Coady, S. Ducasse, G. Kniesel, M. Oriol, and G. Saake, editors, *Proceedings of ECOOP'2007 Workshop on Reflection, AOP and Meta-Data for Software Evolution (RAM-SE'07)*, pages 17–26, Berlin, Germany, July 2007.

[8] JBoss. JBoss AOP homepage. http://labs.jboss.com/portal/jbossaop/, February 2007.

[9] G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In M. Akşit and S. Matsuoka, editors, *Proceedings European Conference on Object-Oriented Programming*, volume 1241, pages 220–242. Springer-Verlag, Berlin, Heidelberg, and New York, 1997.

[10] Knopflerfish. Knopflerfish OSGi homepage. http://www.knopflerfish.org/, March 2007.

[11] M. Lippert. 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*, pages 23–24, New York, NY, USA, 2004. ACM Press.

[12] C. V. Lopes and S. K. Bajracharya. An analysis of modularity in aspect oriented design. In *AOSD '05: Proceedings of the 4th international conference on Aspect-oriented software development*, pages 15–26, New York, NY, USA, 2005. ACM Press.

[13] OSGiAlliance. About the OSGi service platform: Technical whitepaper. http://www.osgi.org/documents/collateral/TechnicalWhitePaper2005osgi-sp-overview.pdf, November 2005.

[14] A. Popovici, G. Alonso, and T. Gross. Just-In-Time Aspects: Efficient Dynamic Weaving for Java. In *AOSD '03: Proceedings of the 2nd International Conference on Aspect-Oriented Software Development*, pages 100–109, New York, NY, USA, 2003. ACM Press.

[15] A. Popovici, T. Gross, and G. Alonso. Dynamic weaving for aspect-oriented programming. In *AOSD '02: Proceedings of the 1st International Conference on Aspect-Oriented Software Development*, pages 141–147, New York, NY, USA, 2002. ACM Press.

[16] The AspectJ Team. The AspectJ Development Environment Guide. http://www.eclipse.org/aspectj/doc/released/devguide/, March 2007.

[17] A. Vasseur. 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.

[18] G. Vaysse, F. André, and J. Buisson. Using aspects for integrating a middleware for dynamic adaptation. In *AOMD '05: Proceedings of the 1st Workshop on Aspect-Oriented Middleware Development*, New York, NY, USA, 2005. ACM Press.

[19] C. Zhang and H.-A. Jacobsen. Refactoring Middleware with Aspects. *IEEE Transactions on Parallel and Distributed Systems*, 14(11):1058–1073, 2003.