What are the key issues for commercial AOP use - how does AspectWerkz address them?

Jonas Bonér
BEA Systems

ABSTRACT

AspectWerkz is a framework targeted towards dynamic aspect-oriented programming (AOP) in commercial applications. Based on our experience designing and supporting AspectWerkz, we have identified what we believe are key issues for the commercial adoption of AOP. These are usability, agility, integration, expressiveness, performance, tool support and the concept of an aspect container that manages issues like security, isolation, visibility, deployment and manageability for the aspects. This invited talk will discuss these issues and how AspectWerkz tries to address them.

Keywords

Aspect-Oriented Programming, AOP, AOSD, dynamic AOP

1. INTRODUCTION

In this invited talk we present our experience regarding the requirements and key issues for commercial aspect-oriented programming (AOP) [13] and how they are addressed by the AspectWerkz framework [12], a framework targeted towards dynamic AOP for Java [5] in commercial, real-world applications. AspectWerkz is still a work in progress; we are continually improving the degree to which AspectWerkz meets the criteria we have identified.

2. KEY ISSUES

The AspectWerkz framework has been developed directly in the commercial market with feedback from users on a daily basis. User feedback and personal experiences has led us to believe that we have identified and to some extent addressed key issues for commercial AOP and its adoptability.

2.1 Usability

One of the most important things to allow AOP to be widely used and adopted in the commercial world is usability. It has to be simple, easy to learn and easy to use. Software projects in the commercial world often have a very tight timeframe and introducing a new technology always introduces a risk. Time needs to be invested in learning the technology and new technologies always increases the possibility of something going wrong. It can also add potential maintenance problems since all developers might not be familiar with the new technology. All these things together make it harder for AOP to gain acceptance in the commercial world. AspectWerkz's main objective has been simplicity and adoptability. We believe that being compatible with the Java Language Specification [5] and use of technologies that are intuitive to users of Java and J2EE [7] will increase the adoptability and flatten out the learning curve for developers unfamiliar with the concepts of AOP. This is the reason why we are using technologies like XML [6] and runtime attributes [3] as the different definition formats, but with a syntax that comes close to the benefits of AspectJ's [1] language extension. Since runtime attributes do not currently exist in the Java language (they are part of JSR-175 [10] which will be released in Java 1.5) we have implemented a custom solution based on JavaDoc tags which are parsed and inserted in into the bytecode of the class. This allows us to have the aspects written as a regular Java classes annotated with runtime attributes containing the aspect definition.

2.2 Agility

Lightweight development processes, also called agile software development are gaining popularity in the commercial world (one example is Extreme Programming [11]). Agile software development is very focused around testing and more and more developers start to practice Test-Driven Development [8]. An equally important tool in agile software development is refactoring [4], the possibility of changing the design of existing code. This means that it is important that the aspects can be easily tested and are easy to refactor. In AspectWerkz this is supported through the use of plain Java as development language as well as the use of runtime attributes for defining the meta-data. Using this approach means that the aspects are easy to test and integrate into existing test frameworks as well refactor using regular refactoring tools found in IDE's.

2.3 Integration

Integration is another issue that is crucial for the success of commercial projects. Minimizing the time in the development cycle spent on integration is very important. It has to be easy and fast to integrate new code into the existing code base, easy to make changes and fixes. Another important issue is easy integration with existing systems (legacy and new) as well as being able to function equally well on any JVM or application server. Seamless integration has been one of the main design goals of AspectWerkz, which achieves this through load-time weaving at bytecode level (compatible with Java 1.3 and above). This allows the weaving to be to some extent transparent to the user and minimizes the integration effort and time. AspectWerkz achieves this by for example inserting a hook in the `java.lang.ClassLoader` system class to enable a Java Virtual Machine (JVM) wide hook mechanism. It also makes use of platform specific solutions when available, for example the class redefinition feature in the JRockit JVM. It also has support for the upcoming Java 1.5 JSR-163 (JVMTI) [2] and its class redefinition mechanism. AspectWerkz supports all major JVMs (HotSpot, JRockit, IBM) and application servers (WebLogic, WebSphere, JBoss, Orion, Tomcat) on the market.

2.4 Expressiveness

We believe that an AOP framework is not more powerful than its join point model is expressive and that the ability to define fine-grained pointcuts and to compose them into even more expressive pointcuts is crucial to be able to meet the sometimes very complex needs and requirements of commercial applications. The current AspectWerkz join point model is getting closer to, but still does not have, the expressiveness and orthogonality of the AspectJ model. AspectWerkz currently supports *execution*, *call*, *set*, *get*, *throws*, *handler* and *cflow* pointcuts. It allows these pointcuts to be composed using logical operators to form new pointcuts. It supports *around*, *before* and *after* advice as well as introductions in the form of mixins.

2.5 Performance

AOP plays a vital role in any system that uses it since it has the ability of affecting any part or subsystem in the whole system. This means that the overhead of the AOP framework has to be minimized to be able to reach acceptable performance and throughput. It is also important that the load-time weaving process, if used, has low overhead to minimize startup time of the system. AspectWerkz makes use of caching and lazy loading to minimize the overhead, but is still in need for optimizations both at runtime and during the load-time weaving process. For example invoking an empty advice takes ~0.000287 ms/call, invoking an empty introduced method takes ~0.000256 ms/call while invoking a regular empty method takes ~0.000003 ms/call when running Java 1.4.1 on a Pentium4 1.6 MHz box.

2.6 Tool support

One thing that we believe will be crucial in the future is tools for managing aspect complexity, tools for managing aspects both at development time (debugging, refactoring, aspect browsing etc.), deployment time, as well as tools to monitor and administrating aspects at runtime. AspectWerkz supports debugging within an IDE and integrates well with the Ant build system [9], but apart from this is lacking good tool support.

2.7 Aspect container

We have experienced that issues like security, isolation, visibility, deployment and manageability for the aspects becomes crucial when multiple applications are deployed in an application server or cluster. These issues needs to be handled both vertically, e.g. follow the class loader hierarchy in Java and horizontally, e.g. within one class loader. These experiences have brought us to the definition of the concept of an ‘aspect container’, which is responsible of handling these issues. Another responsibility for the container is to allow the aspects to be deployed in deployable units on different levels e.g. system level, application level, component level etc. The issues addressed by the ‘aspect container’ concept has so far only been designed and to some extent prototyped in AspectWerkz.

3. FUTURE WORK

Satisfying the key issues for commercial AOP we have discussed is still very much a work in progress in AspectWerkz. Work is still needed to optimize performance, enhance expressiveness and orthogonality, provide more dynamic AOP, improve tool support, security, provide isolation and deployment support, improve management of the aspects and the added complexity etc.

We are currently working on a Just-In-Time (JIT) compiler that detects advice sequences that are frequently executed and can be optimized and compiles a custom class at runtime that invokes this specific advice chain (and the target method) statically.

Although AspectWerkz supports some features of dynamic AOP; addition, removal and reordering of advices as well as swapping the implementation of mixins at runtime, there is a lot of work to do in this field. Both support for dynamic weaving, where the classes can be weaved at any point and not only at load time, allowing new pointcuts to be defined at runtime, as well as ways of dealing with the added complexity these powerful features are adding. AspectWerkz has a prototype for dynamic weaving that solves most of these problems but it is not production ready yet.

We believe that there will also be a need for best practices and design patterns on how to use AOP to solve common problems as well as a need for reusable aspect libraries.

4. ACKNOWLEDGMENTS

Thanks to Alexandre Vasseur, for many very important contributions to the design and implementation of AspectWerkz as well as feedback on how to improve this paper; and to Gregor Kiczales, for feedback on how to improve this paper.

5. REFERENCES

[1] Kiczales, G., E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold, An Overview of AspectJ, ECOOP 2001.

[2] JSR-163. At http://www.jcp.org/en/jsr/detail?id=163, 2004.

[3] Newkirk, J., Vorontsov, A. How .NET’s Custom Attributes Affect Design. 2004.

[4] Fowler, M. Beck, K., Brant, J., Opdyke, W., Roberts, D. Refactoring: Improving the design of existing code. Addison-Wesley, 1999.

[5] Gosling, J., Joy, B., Steele, G. The Java Language Specification (2nd edition) Addison-Wesley, 2000.

[6] W3C, Extensible Markup Language (XML) 1.0 (Second Edition). At http://www.w3.org/TR/REC-xml, 2004.

[7] Java 2 Enterprise Edition, At http://java.sun.com/j2ee/.

[8] Beck. K, Test-Driven Development. Addison-Wesley, 2003.

[9] Ant Build System, At htttp://ant.apache.org, 2004.

[10] JSR-175. At http://www.jcp.org/en/jsr/detail?id=175, 2004.

[11] Beck. K, Extreme Programming Explained: Embrace Change. Addison-Wesley, 1999.

[12] AspectWerkz homepage http://aspectwerkz.codehaus.org.

[13] Kiczales,G. Lamping,J., Mendhekar,A.,Maeda,C.,Lopes,C., Loingtier,J.M., Irwin,J., Aspect-Oriented Programming, ECOOP1997