How to Convince Industry of AOP

Daniel Wiese, Uwe Hohenstein, Regine Meunier

Siemens AG
CT SE 2
Otto-Hahn-Ring 6
D-81730 München
Germany
<Firstname>.<Lastname>@siemens.com

Abstract

This paper presents a proposal for convincing industry of aspect-orientation, as it has been applied within Siemens. The proceeding stresses on the immediate benefits and ease of usage. Starting with an existing application, we show how to improve the performance and how to extend the behavior with only a few code modifications by bringing aspect-orientation into the game. An adequate infrastructure helps to use aspects easily within IDEs such as Eclipse.

1. Introduction

Aspect-Orientation (AO) is not a brand-new technology. Nevertheless its usage in industry is not wide-spread and often covers only use cases such as logging and tracing. That is, dissemination has not been as successful as it should be.

[JST06] provides an analysis of how Moore's work [Mo91] on adoption of new technologies applies to aspect-oriented software development (AOSD). The authors conclude that Moore's model can be considered as an optimistic approximation of AOSD adoption. A recent study of adoption of AOP within non-academic projects indicates that the majority of the interviewed developers were "early adopters" of this technology [Du06]. We can acknowledge the same for the use of AOP within Siemens. The current stage of adoption is that occasionally developers learn the AO concepts and try to apply them in non-critical phases of development projects. Very rarely the project management deliberately decides to use AO technologies in a project.

We encountered the following problems when trying to bring AO into business projects:

- Industry is often afraid of AO: AO mechanisms change code, e.g., by means of interception or byte-code modification; this has a touch of being obscure and dangerous.
- Furthermore, the better concerns are decoupled, the harder it is to understand their run-time interactions. Software developers experienced the same when moving from procedural programming to object-oriented programming. AO allows modularizing a system even better, which results in good comprehensibility for single concerns, but has the effect that the overall interactions between the concerns at run-time are harder to evaluate. For example, it is hard to see where a pointcut is applicable or how the context between a pointcut and an advice is exchanged. A recent paper by Mik Kersten et al. [Ke06] shows that AspectJ development is heavily depending on tool support in contrast to OOP languages, which can be successfully used with plain text editors. Meanwhile, this tool support is available for some AOP approaches.
- There is a lack of success stories, which keeps one of the obstinate myths living: *"AO is good only for logging/tracing"* [La06]. Some evangelists are using AO in industrial projects, but unfortunately, little experience about success or problems encountered is reported to a broader audience. [BF06], [CC04] and [Le06] report on experience with AO in industrial settings. [Bo05] and [GN+06] implement a tool for monitoring and a tool for performance management using AO techniques, respectively. This is exciting work that benefits from AO a lot, but often not enough to prove a broad acceptance of the technology.

We see a kind of a vicious circle here: Industry needs large scale success stories to be convinced. But, to produce such success stories you have to apply AO in industrial projects. Two additional options are suggested in the AO community to convince managers of applying AO in projects:

- One possibility is to provide concrete measurements about benefit and success in terms of modularity, reusability, LOC, development time etc. in real applications that already exist.

- Ron Bodkin and others [Bo06, Ki05] describe several stages for AOP adoption. These stages guide single developers who want to get familiar with AO in several steps. This will work if a critical mass of developers can be convinced, which then in turn influences decisions of their management.

The proposal we present here suggests an additional path of dissemination. Our approach is to provide support for using AOP with low effort and high benefit, and to demonstrate the ease of use and the benefit by means of a demonstrator to interested groups of developers and managers.

The support for making AO easy to use consists of the following:

- Use a programming language that provides extended IDE support such as AspectJ.
- Provide an overall AO infrastructure, based on Eclipse and Maven, which eases the application of AO without spending time for setting up a programming environment.
- Make reusable aspects available, easy to apply by everybody without knowing much about AO.
- Provide an adequate infrastructure that supports aspect reusability, namely an Eclipse plug-in, called Aspect Manager, which allows for an easy and immediate application of aspects.

The demonstration of AO techniques and the provided support follows a schema we found very effective:

- Gather a group of developers and managers interested in AO.
- Explain the benefits of AO and the support and infrastructure we provide.
- Show a live demo: Improve the performance of an existing application dramatically by using AO and add additional behavior quickly.

The remainder of the paper is organized as follows. We recapitulate the well-known stages of AOP adoption in Section 2. In Section 3 we present our path of AOP adoption by describing a problematic application in detail, which can be improved by reusable aspects. Furthermore, we discuss the infrastructure that supports aspect reusability effectively. In Section 4 we conclude and present our future work.

2. Stages of AOP Adoption

It is known that aspects are handy for logging and instrumentation, and it is promised that they can be applied to more complex problems as well.

Bodkin [Bo06] presents practical guidelines for taking the next step with AOP after having just started out with simple aspects. Most are unsure of how to apply it to their daily development practices or to convince decision-makers in an organization to adopt it.

He presents practical guidelines for taking that next step with aspects. He introduces different stages of AOP adoption and offers examples of learning applications and guidelines for success at each stage:

- Stage 1. Learning and experimenting
- Stage 2. Solving real problems
- Stage 3. Integrating aspects into core development

Throughout the stages of adoption, a few key principles apply:

- Adopt incrementally: Learn to use aspects a little bit at a time. Start with "development aspects" that do not put your production system at risk. Then apply them to advantage. Finally, expand from there. At each stage, it's important to build on what has already worked and to find new opportunities.
- Reuse, and then create: Configuring pre-built components is a great way to benefit from the power of aspects, just as it was a great way to benefit from the power of objects. As you gain experience, you will want to customize and ultimately create your own reusable components.
- Invest in pleasant surprises: Provide no-cost examples of how aspects can solve the thorny problems in your system before asking colleagues or higher-ups to commit to aspects.

Naturally, becoming more experienced with AOP, one will gain the skills needed to use it for more interesting solutions, with correspondingly greater benefits. This can mean using aspects more broadly or more deeply.

As awareness of AOP grows within the organization, other developers naturally learn more and start writing their own aspects.

3. A New Path of AOP Adoption

The previous proposal will work if a critical mass of developers can be convinced, which then in turn influences decisions of their management. Our approach does not rely on this assumption.

Our suggested path for AOP adoption consists of two elements. On the one hand, we provide a technical solution easing the use of aspects. The aim is to enable even AO newcomers to immediately apply the technology. The solution consists of:

- reusable aspects
- use of annotations (avoiding a pointcut language)
- Aspect Manager and other infrastructure
- support and training

On the other hand, we provide an effective presentation of our solution and support. This consists of

- an adequate application to show the benefits of the AO solution;
- a live demonstration showing how easily and effectively the application can be improved.

3.1 A problematic application

Before dwelling on the technical solution, we shortly present the application. The aim of the application is to manage personal data for employees working in a department. Figure 1 shows a screenshot of this application marking the problematic areas of the application.

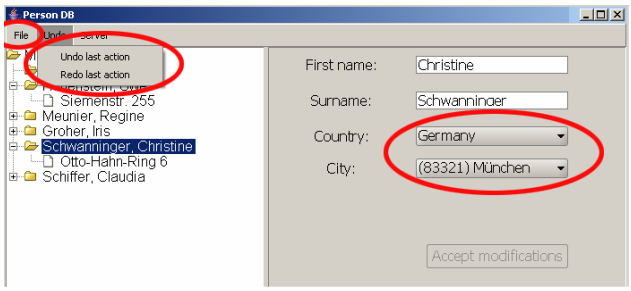

Figure 1: Personal data application

The left hand side displays all employees working in one department. If a user selects a person, details will be displayed on the right side. Every employee can have multiple address records.

The performance of this application was quite poor. For instance, every time, when an employee was selected, the user had to wait several seconds until he could continue his work.

In a nutshell, the major performance problems of this application were:

1. Loading all cities of a country takes a lot of time. Especially every time when a person record was displayed, all countries and all cities of this country (displayed in a drop down box) were loaded from the database again and again.
2. Similarly, storing modified addresses to the database is time-consuming and blocks further operations.
3. There is a very unstable Undo/Redo management present in the system because of the complexity to implement it.

This application could be improved by:

1. Caching: Country and city names are quite stable and do not need to be fetched from the database every time a person record is loaded. Instead, it is enough to fetch the addresses only once for further accesses.
2. Storing asynchronously: Saving addresses can be done asynchronously in order to not block other operations any longer.
3. Stable and tested out of the box undo/redo management for changing person and address records.

All these features can be implemented by means of aspects. Of course, other techniques such as design patterns [BM+96] are applicable, too. However, the use of pre-defined aspects is convincing because it shows how easy the additional functionality can be added by means of a few annotations.

But even if the benefits of using AO are high, we found that the right infrastructure is a very critical issue to convince managers and developers.

3.2 Aspect Manager and infrastructure

AO promises modularity and reusability of software. These properties count in the long run. The importance of a good infrastructure for developers in industrial projects should not be underestimated if you want to achieve high adoption of a new technology. Developers are not willing to pay for modularity and reusability by loosing the comfortable features of IDEs in their daily work. Therefore, the extra benefits promised by AO technology will only be accepted if it is delivered with an infrastructure that is not inferior to the current state-of-the-art.

Infrastructure for using aspects is already available in some common IDEs. We essentially base the discussion on Eclipse and AspectJ, but the basic ideas can also be adapted to other environments.

The basic block of our infrastructure consists of a central download site with all relevant plug-ins for different Eclipse versions. This central plug-in bundling infrastructure enables project teams to setup their IDE´s for AO usage in a few minutes. Currently we are providing the following plug-ins:

1. An Eclipse plug-in, called AJDT, allows to compile AspectJ code within Eclipse. Moreover, there is full IDE support for AspectJ: Graphical support helps to select the joinpoints where an aspect is changing behavior. The other way around, when selecting a method, all aspects that affect that method are immediately visible.
2. Another plug-in allows for an easy integration of build tools such as Ant or Maven. They enable the definition and automation of the build process. Particularly, Maven is useful due to its dependency management concept: A user can define dependencies to required JARs in a pom.xml file. Maven is then automatically downloading the JARs in specified versions from a list of servers that can be defined; one of them is certainly our download site. Furthermore, it can be defined into which JARs aspects should be woven.
3. The Siemens Aspect Manager is the central component of our infrastructure. It is also the seam to connect the AJDT plug-in and the external build system Maven, when using predefined aspects (AJDT and Maven have different build approaches).

The first two plug-ins already exist, and we just added the download mechanism. Unfortunately, both plug-ins have some deficiencies: AJDT support is not really

given – although it seems so. Building an application takes place in Maven in addition to Eclipse compilation. If Maven is downloading an aspect given as a JAR, then the Maven compiler will be aware of applying aspects, but the Eclipse compiler is not. This will confuse developers because there is no graphical visualization of join points. Furthermore, the developer has to change the pom.xml file in order to achieve the weaving of aspects.

The Siemens Aspect Manager is an Eclipse plug-in that makes the configuration task much easier and supports the reuse of predefined aspect libraries effectively. The Aspect Manager uses an *aspect repository server* from which aspects can be queried and imported. This aspect repository hosts several aspects in form of JARs. Then, the Aspect Manager contacts the server and gets knowledge of all available reusable aspects. The Aspect Manager displays all available Java projects in the Eclipse workspace. By right-clicking on a project, all available aspect libraries in the repository (such as "Caching method invocations", "Asynchronous method invocation" or "Jndi and Remoting") will be displayed to the programmer. The programmer can then select aspects to be included in his project. This is the only task he has to perform; all other work is done internally by the plug-in, e.g., to maintain the Maven dependency to the aspect libraries in the pom.xml, to let the compiler weave in aspects that are defined in a different JAR to his Java project, and to let Eclipse become aware of instrumented code. Now, aspects become immediately usable.

Figure 2 presents a screenshot of the Aspect Manager.

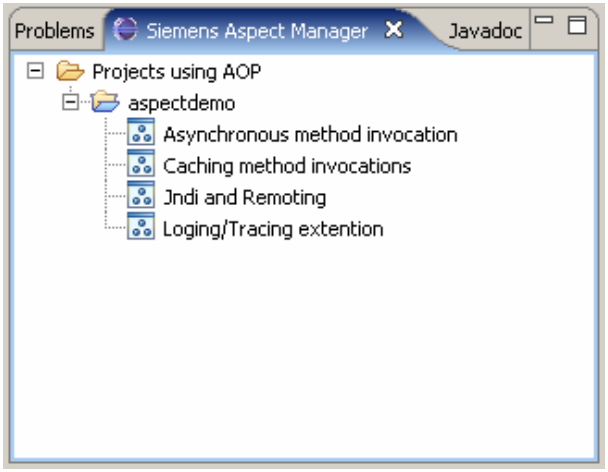

Figure 2: Siemens Aspect Manager

If the user selects one of the predefined aspects such as "Caching method invocations", then the Aspect Manger will perform the following operations:

- The Caching aspect will automatically be downloaded from the repository server as a JAR; the JAR is then installed on the local developer machine.
- The Maven pom.xml file is automatically adjusted so that other build systems (like Continuum) can automatically build the application including the caching functionality. This means that the Aspect Manager handles the dependencies to the aspect library, and allows AspectJ to weave in dependencies.
- All pointcut definitions in the aspect are exposed to the AspectJ weaver. The AspectJ weaver is triggered by the Aspect Manager and weaves the selected functionality into the current application, in this example, the caching functionality. Join points are visible in Eclipse just as the other AJDT support is provided.

3.3 Easy usage

As already stated in [La05], Java-5 annotations are great when starting with AO. Even completely inexperienced AO programmers can benefit from the technology.

In the previous section, we have already described how predefined aspects can be included into projects using the Siemens Aspect Manager. To apply pre-defined aspects, the concept of Java-5 annotations is used:

```
@Cached
void loadAddresses(...)

@Asynchronous
void saveAddresses(...)
```

`@Cached` and `@Asynchronous` annotations are used to mark methods for applying the corresponding aspect. The specific caching and asynchronous behavior is implicitly added. The effect is immediately visible: The application performs faster, and the only thing to be done is to use the Aspect Manager to import these aspects into an existing Eclipse project.

This means that the user can directly apply the `@Cached` annotation to his code without caring about the Eclipse configuration, Maven, and their relationship to AspectJ.

3.4 A Collection of aspects

Pre-built library aspects are relatively new on the scene, although some good aspect-oriented applications are now becoming available, including the Glassbox Inspector [Bo05], the JBoss Cache, and the GoF patterns library [HK02]. However, these are coarse-

grained implementations that could be used directly without seeing AO.

We want to start fine-grained by offering smaller useful aspect libraries, which are a good way to start learning and applying them in projects. We implemented the following aspects to this end:

- Asynchronous method invocation
- Caching
- Undo/Redo management

and others not explained in this paper such as Remoting, Transactions or Failover. Figure 4 and 5 at the end of the paper present the major part of coding.

Caching

The idea is to trap any method that has been annotated with `@Cached`. In this case, the caching aspect will return, if available, the cached value of the method instead of invoking the original method again. The principle can be illustrated by an example:

```
@Cached
public List method(String param1,
                   int param2) {...}
```

Let us assume, this example method performs a long running operation. If the method is invoked twice with the same parameter values, then the cached value should be returned for the second invocation.

This idea can be implemented very nicely using aspect orientation. The following pointcut traps any method that has been annotated with `@Cached`:

```
pointcut execCachableOperation() :
       call(@Cached * *(..))
   && !within(@Cached *);
```

Using an around advice, we can check whether the method has already been invoked for the given parameter values instead of executing the original method immediately:

```
Object around():execCachableOperation(){
   Object[] args = thisJoinPoint.getArgs();
   MethodSignature m = (MethodSignature)
            thisJoinPoint.getSignature();
   Object key= MethodBodyCacheEntry.
        generateKey(m.getMethod(), args);
   If the key is inside the cache, return
   the cached entry, else invoke the
   method and put the key and result to
   the cache;
}
```

Please note that `!within(@Cached *)` is required in the `execCachableOperation` pointcut to exclude the call of the cached operation in the advice from being trapped.

The problem now is how to detect whether a method is called twice with the same parameter values. To perform such a check, we are using a key generator (`MethodBodyCacheEntry.generateKey(...)`) which guarantees that:

- `key(f(x1, .., xn)) = key(g(y1, .., yn))`
  `=> f=g ∧ x1=y1 ∧ x2=y2 ... ∧ xn=yn`
- `f!=g ∨ x1!=y1 ∨ ... ∨ xn!=yn`
  `=> key(f(…)) != key(g(…))`

The key will be equal if the method and the set of parameters are equal; the keys are unequal otherwise.

The key is generated by the following static method, which just returns an instance of `MethodBodyKey`:

```
public static Object generateKey(
          Method method, Object[] args) {
  return new MethodBodyKey(method, args);
}
```

The class `MethodBodyKey` uses Java `hashCode()` and `equals()` contracts to guarantee the conditions above:

```
public int hashCode() {
  final HashCodeBuilder hcb =
                    new HashCodeBuilder();
  hcb.append(this.method);
  hcb.append(this.args);
  return hcb.toHashCode();
}

public boolean equals(Object obj) {
  ...
  final EqualsBuilder eqb =
                     new EqualsBuilder();
  eqb.append(this.method, obj.method);
  eqb.append(this.args, obj.args);
  return eqb.isEquals();
}
```

We use the `MethodBodyKey` to check if the same key is already inside the cache. If the key does not exist in our cache, we invoke the original method by using `proceed()` and put the result into the cache. Otherwise, we return the cached entry.

The cache itself is pluggable; we use the EHCache (for example, used in Hibernate), but any other cache implementation can also be plugged in.

Asynchronous method invocation

The idea of the asynchronous invocation aspect is to invoke methods asynchronously. Here, we only handle methods without return values. The principle is the same as for caching: A pointcut traps all methods annotated with `@Asynchronous`, and an around advice cares about the asynchronous execution:

```
pointcut invokeAsynchronously() :
         call(@Asynchronous * *(..))
  && !within(@Asynchronous *);
```

An around advice extracts the relevant parameters such as the object on which the method should be invoked, the method itself and the parameters:

```
void around() : invokeAsynchronously() {
  Object[] args = thisJoinPoint.getArgs();
  MethodSignature m = (MethodSignature)
            thisJoinPoint.getSignature();
  Object target =
               thisJoinPoint.getTarget();
  AsynchMethodInvoker.execute(
           m.getMethod(),target, args);
}
```

The class `AsynchMethodInvoker` provides a static method `executeAsynchronously` which puts the methods to be invoked asynchronously in a thread pool:

```
ExecutorService threadPool =
     Executors.newFixedThreadPool(LIMIT);

public static void executeAsynchronously
  (Method m, Object obj, Object[] args) {
  final MethodExecuter toExecute =
          new MethodExecuter(m,obj,args);
  synchronized (threadPool) {
       threadPool.execute(toExecute);
  }
}
```

The class `MethodExecuter` is just a container which implements the `Runnable` interface and invokes the method asynchronously via reflection inside the thread pool:

```
public void run() {
  if (!this.method.isAccessible()) {
    method.setAccessible(true);
  }
  try {
    method.invoke(onObject, arguments);
  } catch(...){...}
}
```

Furthermore, we have to take into account that the thread pool must be shut down at the end of the application. An annotation `@Shutdown` is used to define the place where to shut down. An after advice performs this job. A possible place for the `@Shutdown` annotation can be the `main()` method of an application.

Undo/Redo

A lot of applications, especially UI applications, need a reliable undo/redo management. That is why we provide an aspect that supports software developers to realize undo/redo management on object graphs. From our experience, a lot of developers find it challenging to implement such functionality every time from scratch.

But why it is difficult to implement an Undo management?

Assume `Person` is an "undoable" class, for which possible changes on a `Person` object can be reverted or the reverted changes can be restored:

```
Person p = new Person();
List<Address> l = p.getAddresses();
l.add(new Address(...));
p.setAddresses(l);
```

To implement such functionality, we have to detect any modification of a `Person` object. For simple attribute value changes, we can augment the implementation of setter methods (if there are any) by some notification mechanism. Changes in sets or lists such as `l.add(new Address(...))` above are more difficult to handle, since we have no direct access to their implementations. A possible but cumbersome solution is to subclass `Collection` classes, to add the notification behavior, and to restrict the usage of collections to those subclasses.

In any case, if we add a new class to the application, we need to enhance that class with such a monitoring functionality: We need to monitor every change of every attribute for every "undoable" class. However, the monitoring is certainly crosscutting!

A software developer, who wants to automatically prepare all domain objects in a package for undo/redo management, can use the following pointcut to annotate all domain classes with an `@Undoable` annotation, e.g., all the classes in package `some.package`:

```
declare @type: ((some.package..*) :
                                 @Undoable;
```

With AOP, it is now easy to monitor field modifications in any class annotated with `@Undoable` by using the `set` pointcut:

```
pointcut fieldModification() :
       set(* *.*) && within(@Undoable *);
```

Moreover, list modifications can also be monitored easily for all `Collection` subclasses:

```
pointcut listModification() :
          call(* Collection+.add(..))
       && within(@Undoable *);
```

The next step consists of extending the `@Undoable` classes with the corresponding behavior (`undo()`,

redo(), ...). This can be done by intertype declarations in the following manner:

```
declare parents : (@Undoable *)
      implements com.siemens.UndoHandler;
declare parents : (@Undoable *)
      extends com.siemens.UndoHandlerImpl;
```

Undoable classes are now implementing an UndoHandler interface by extending a predefined class UndoHandlerImpl. The interface UndoHandler provides the signatures which are implemented by UndoHandlerImpl, and the latter implements the complete Undo/Redo functionality: It adds the methods undo(), redo() and markUnit() to undoable classes. Figure 3 illustrates the semantics of these methods.

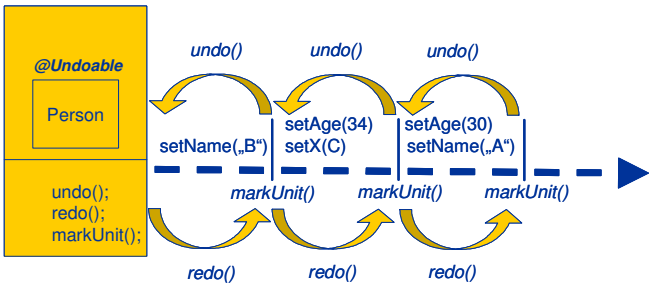

Figure 3 : Undo management

markUnit() can be used to mark several changes on an object graph as one unit of work. markUnit() can for instance be called whenever a save button is pushed. The undo() operation reverts all changes on the object graph until the last unit of work marker (or the initial object state).

The field changes are trapped by the pointcuts fieldModification and listModification, as described before. Every modification is monitored using the command pattern. The UndoHandlerImpl uses this command list to revert or restore changes, by applying the list of commands to the object. Every command object has an undo() and a redo() method which apply/revert the atomic change represented by the command, e.g., restore the old value of a field or revert the field by setting the previous value.

This technical stuff runs in the aspect internally. An application developer just has to use the undo/redo methods that are added by UndoHandlerImpl to any undoable class, e.g., implementing Undo/Redo buttons..

3.5 Description of live demo

For us, the value of the technical solution described in the last two paragraphs is twofold. On the one hand, we can deliver the aspect library, the Aspect Manager and the additional infrastructure for use in projects. On the other hand, the personal data application together with the available AO support is an excellent means to present the benefits of AO technology. Our presentation proceeds along the following lines:

- We demonstrate the problematic application and identify the bad performance because of retrieving addresses unnecessarily from the database. Well, we can certainly hire a database specialist to improve the performance, but it is obvious that we can benefit from caching here. Such a cache can be implemented in a couple of days by software developers, or in a minute using the predefined Caching aspect. We select this aspect with the Aspect Manager and use its functionality just by annotating the relevant methods with @Cached. The audience will see that this small code modification is faster than even the compilation, and performance will be much better. Using the Aspect Manager, just compilation is necessary, nothing else.
- Then, we identify the bad performance because of synchronous database storage. Again, we select an aspect with the Aspect Manager, the Asynchronous aspect, and apply it for asynchronous method invocation.
- Finally, we solve the missing Undo/Redo functionality by importing the Undo/Redo aspect and using the functionality..

This presentation shows how a developer with no AOP programming experience can improve an application radically by using pre-defined aspects in only twenty minutes, which is quite convincing!

4. Conclusions and Future Work

We presented in this paper a holistic way already applied at Siemens of convincing industry of aspect-orientation.

The key to success consists of several building blocks: At first, we provide an Eclipse plug-in that allows the selection of pre-defined aspects that are available at a repository server; we can apply these aspects directly in our Eclipse project without caring about build tools and enabling the weaving process.

Second, we make our pre-defined aspects easily usable without specifying complex pointcuts: The aspects define Java-5 annotations that can immediately be applied.

And finally, we used an existing, ordinary Java application in order to show the potential and benefit of AO: Using aspects that perform caching and asynchronous method execution behavior, we can improve the performance of that application with only a very few code changes. Similarly, we can easily add an Undo/Redo behavior by using aspects.

A demonstration shows in twenty minutes how useful aspect-orientation (AO) is, without claiming for

a better code structure or the avoidance of code scattering and tangling [EFB01].

However, this is only the first step of our dissemination strategy. With our support and the demonstration, we gain partners in Siemens business units who are interested in applying AO in their projects. They can serve as cells of AO knowledge in their business units and one or the other AO-based implementation will emerge. Having such AO-based implementations, we could and should show the benefit of AO, e.g., a smarter implementation, a smaller amount of implementation work, a better flexibility, adaptability, customization, configurability, and so on to convince more managers and developers.

Possible candidates for larger AO implementations are commonly accepted and widely used tools such as EJB containers, Object Request Brokers, Object/Relational (O/R) Mapping Tools such as JDO tools, or database systems, which are implemented in a conventional manner recently. If such tools are implemented with AO, it becomes easier to convince industry. The book of Rashid [Ra04] already addresses Aspect-Oriented Databases. It gives an overview about ongoing research and discusses all facets of AO in the context of databases: AO to implement database systems in a more modularized manner, persistence for aspects, and some ideas on a persistence framework.

We want to continue our work with not implementing just only persistence, but the whole EJB stack [EJB3] including dependency injection, stateful and stateless session beans, remoting, transactional behavior etc.

References

[AJ06] AspectJ: Aspect-oriented Programming in Java, http://www.aspectj.org

[AW06] AspectWerkz home page: http://aspectwerkz.codehaus.org/index.html

[BM+96] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture – A System of Patterns, John Wiley and Sons, 1996

[Bo04] J. Bonér: What are the Key Issues for Commercial AOP Use: How does AspectWerkz address them? In Proc. 3rd Conf. on Aspect-Oriented Software Development, AOSD 2004, Lancaster, ACM Press

[Bo05] R. Bodkin: AOP@Work: Performance monitoring with AspectJ. http://www-128.ibm.com/developerworks/java/library/j-aopwork10/index.html

[Bo06] R. Bodkin: AOP@Work: Next Steps with Aspects. http://www-128.ibm.com/developerworks/java/library/j-aopwork16

[Bu05] B. Burke: Implementing Middleware Using AOP; in Proc. 4th Conf. on Aspect-Oriented Software Development; AOSD 2005, Chicago, ACM Press

[BF06] R. Bodkin, J. Furlong: Gathering Feedback on User Behaviour using AspectJ; in [CVK06]

[CC04] A. Colyer, A. Clement: Large-scale AOSD for Middleware. In Proc. 3rd Conf. on Aspect-Oriented Software Development, AOSD 2004, Lancaster, ACM Press

[CG04] T. Cohen, J. Gil: AspectJ2EE=AOP+J2EE – Towards an Aspect Based, Programmable and Extensible Middleware Framework. In Proc. 18th European Conf. on Object-Oriented Programming, ECOOP 2004, Oslo

[CVK06] M. Chapman, A. Vasseur, G. Kniesel (eds.): Proc. Of Industry Track 3rd Conf. on Aspect-Oriented Software Development, AOSD 2006, Bonn, ACM Press

[Du06] A. Duck: Implementation of AOP in Non-Academic Projects; in [CVK06]

[EJB3] The EJB3 Specification (JSR 220): http://jcp.org/aboutJava/communityprocess/pfd/jsr220/index.html

[GN+06] K. Govindraj, S. Narayanan et al.: On Using AOP for Application Performance Management. In [CVK06]

[JST06] W. Joosen, F. Sanen, E. Truyen, Dissemination of AOSD expertise – support documentation; AOSD-Europe Project Deliverable No.: AOSD-Europe-KUL–8, Mar. 06

[HK02] J. Hannemann and G. Kiczales. Design Pattern Implementation in Java and AspectJ, Proc. of the 17th Annual ACM conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2002, Seattle

[Ho05] U. Hohenstein: Using Aspect-Orientation to Add Persistency to Applications. Proc. of Datenbanksysteme in Business, Technologie und Web (BTW), Karlsruhe 2005

[Ke06] M. Kersten, M. Chapman, A. Clement, A. Colyer: Lessons Learned building tool support for AspectJ, in [CVK06]

[Ki05] G. Kiczales: Adopting AOP; in Proc. 4th Conf. on Aspect-Oriented Software Development; AOSD 2005, Chicago, ACM Press

[La05] R. Laddad: AOP@Work: AOP and Metadata: A Perfect Match. http://www-128.ibm.com/developerworks/java/library/j-aopwork3

[La06] R. Laddad: AOP@Work: Myths about AOP. http://www-128.ibm.com/developerworks/java/library/j-aopwork15

[Le06] N. Lesiecki; Applying AspectJ to J2EE Application Development. IEEE Software, January/February 2006

[Mo91] G. Moore: Crossing the Chasm. HarperBusiness, 1991

[RC03] A: Rashid, R. Chitchyan: Persistence as an Aspect. In M. Aksit (ed.): 2nd Int. Conf. on Aspect-Oriented Software Development, AOSD 2003, Boston, ACM Press

[SB05] S. Soares, P. Borba: Implementing Modular and Reusable Aspect-Oriented Concurrency Control with AspectJ; in WASP05, Uberlândia, Brazil

[Ta05] D. Teare: Quick Start Guide to Enterprise AOP with Aspectwerkz 2.0, 2005, http://dev2dev.bea.com/pub/a/2005/04/enterprise_aop.html

```
public aspect CachingAspect issingleton() {
  pointcut execCachableOperation() : call(@Cached * *(..)) && !within(@Cached *);
  // create a CacheManager using defaults
  private final CacheManager manager = CacheManager.create();
  Object around() : execCachableOperation(){
    //extract the method and the target
    Object[] args = thisJoinPoint.getArgs();
    MethodSignature method = (MethodSignature)thisJoinPoint.getSignature();
    final Object key = MethodBodyCacheEntry.generateKey(method.getMethod(), args);
    final Cache myCache = manager.getCache(CacheProvider.EH-CACHE, "defaultConfig.xml");
    // check if the object is inside the cache
    Element cachedBack=myCache.get(key);
    Object back=null;
    // lets look if the return value is already chached
    if (cachedBack==null){
       Object backFromMethod = proceed();
       Element element = new Element(key, backFromMethod);
       myCache.put(element);
       back=backFromMethod;
    } else {
      back=cachedBack.getObjectValue();
    }
    return back;
  }
}

public aspect AsynchAspect {
  pointcut invokeAsynchronously() : call(@Asynchronous * *(..))
                                 && !within(@Asynchronous *);
  pointcut applicationTerminate() : execution(@Shutdown * *(..));
  void around() : invokeAsynchronous() {
    Object[] args = thisJoinPoint.getArgs();
    MethodSignature m =(MethodSignature)thisJoinPoint.getSignature();
    Object target = thisJoinPoint.getTarget();
    AsynchMethodInvoker.executeAsynchronously(m.getMethod(), target, args);
  }

   after() : applicationTerminate() {
     AsynchMethodInvoker.terminateThreadPool();
  }
}
```

Figure 4: Caching and Asynchronous Method Execution Aspect

```
public aspect RecordCollectionModifications {
  /** Call to an undoable object */
  pointcut undoableObject() : within(@Undoable *);
  pointcut addToList(Object o, Object newValue) :
    args(newValue) && this(o) && call(* java.util.Collection+.add(..));
  pointcut removeFromList(Object o, Object newValue) :
        args(newValue) && this(o) && call(* java.util.Collection+.remove(..));
  pointcut clearList(Object o) : this(o) && call(* java.util.Collection+.remove(..));
  /**
   * Advice to any field modification caused by the execution of a field.
   * Stores the new value and the old value as a Modification on the Command.
  */
  before(Object o, Object newValue) : addToList(o, newValue) && undoableObject() {
    try {
      final Collection coll = (Collection) thisJoinPoint.getTarget();
      final Command cmd = CollectionModificationCommand.addModification
                (coll, newValue, CollectionModificationType.ADD);
      List<Command> list = CommandRepository.getCommandList(o);
      list.add(cmd);
    } catch (Exception e) {
       e.printStackTrace();
    }
  }
  /**
   * Advice to any field modification caused by the execution of a field.
   * Stores the new value and the old value as a Modification on the Command.
   */
  before(Object o, Object newValue) : removeFromList(o, newValue) && undoableObject() {
    try {
      final Collection coll = (Collection) thisJoinPoint.getTarget();
      final Command cmd = CollectionModificationCommand.addModification
                 (coll, newValue, CollectionModificationType.REMOVE);
      List<Command> list = CommandRepository.getCommandList(o);
      list.add(cmd);
    } catch (Exception e) {
       e.printStackTrace();
    }
  }

  /**
   * Advice to any field modification caused by the execution of a field.
   * Stores the new value and the old value as a Modification on the Command.
  */
  before(Object o) : clearList(o) && undoableObject() {
    try {
      final Collection coll = (Collection) thisJoinPoint.getTarget();
      final Command cmd = CollectionModificationCommand.addModification
                        (coll, null, CollectionModificationType.REMOVE);
      List<Command> list = CommandRepository.getCommandList(o);
      list.add(cmd);
    } catch (Exception e) {
       e.printStackTrace();
    }
  }
}

public aspect UndoInterfaceDeclaration {
  declare parents : (@Undoable *) implements com.siemens.ct.undo.UndoHandler;
  declare parents : (@Undoable *) extends
                                  com.siemens.ct.undo.impl.UndoHandlerImpl;
}
```

Figure 5: Undo/Redo Aspect