An Introduction to Spring: For Technical Managers, Developers and Architects

WRITTEN BY RICK HIGHTOWER, CTO OF ARCMIND, INC "KNOW THE NEXT!": TRAINING, CONSULTING AND MENTORING FOR SPRING, HIBERNATE, JSF AND MUCH MORE. (http://www.arc-mind.com)

Contributors: Thomas Bridges (Qualcomm Inc.), Alex Redington, Kris Diefenderfer, Mary Basset, Martha Pena, Andrew Barton (eBlox) and Drew Davidson (Ognl).

Following the latest developer trends, you've likely heard of Spring, IoC (Inversion of Control) containers and AOP (aspect-oriented programming). You may not, however, see where these technologies fit into your development efforts.

Why is it time to consider Spring, AOP and IoC?

- AOP ***reduces cross-cutting concern code*** that would otherwise be repeated throughout your code base.
- AOP ***makes your code base more extensible*** through its generic event-publishing capabilities. Code that is more extensible can be extended without major rewrites of the code base.
- IoC ***gets rid of so-called "wiring code"*** that typically occupies 30% or more of many code bases. It allows for easy assembly of objects with their collaborators.
- IoC ***facilitates test-driven development*** and unit testing.
- A Spring Template ***reduces the possibilities of defects*** that leak resources, and reduces lines of code. Spring takes common, error-prone code and encapsulates the logic for resource management and tricky exception handling so it can be reused, with less redundant code.
- Spring and TDD ***increases productivity.*** Spring is more productive than traditional J2EE development, and allows developers to be more productive with the lightweight frameworks that can run inside or outside of the J2EE container.

- Spring ***simplifies developing***. Spring provides utilities and higher level abstractions for a whole host of projects. These utilities significantly simplify developing with these frameworks.

- Spring ***is easily implemented*** into projects. Because it is not prescriptive, associations may be implemented piece by piece.

KEY POINT: Spring is not prescriptive

One of the biggest benefits of Spring is that it isn't prescriptive. Trying to associate something is easier if done piece by piece, rather than by trying to do it all as one great big horse pill. Using Spring's non-prescriptive elements, one may say, "Let's use this one Spring piece" and then realize that Spring can solve other issues as well.

Spring is therefore easily implemented into your projects, step by step.

Spring: More than IoC and AOP

Spring promotes good programming practices. Spring provides for consistent management of resources and exceptions, while providing great examples of how to effectively and consistently use IoC, AOP and OOP.

The ability to inject collaborating objects, called Inversion of Control (IoC), is also referred to as Dependency Injection (a term coined by Martin Fowler). Thus, Spring is an IoC container. The ability to dynamically add services to objects is called Aspect Oriented Programming (AOP). Spring supports both. Thus, Spring is an IoC/AOP container.

Even if Spring were merely an IoC/AOP container, it would be worthy of consideration; however, Spring goes beyond just being an IoC container or an AOP framework. Spring goes one step further than the other containers which support IoX and AOP by building a framework to simplify J2EE development.

Spring and IoC: Dependency Injection

Injecting Collaborators

With Spring IoC support, one simply injects collaborating objects, called dependencies, using JavaBeans properties and configuration files. Then it's a simple

process to switch out collaborating objects when needed. One may also define the dependencies as “stubs,” created specifically for unit testing.

JNDI Inside Out

In the J2EE model, domain objects are able to look up and inject other objects, such as datasources, which are needed to do their job. As the word inversion implies, IoC is somewhat like JNDI turned inside out. Instead of using a tangle of JNDI lookup, abstract factories, service locators, singletons, and straight construction, with IoC, each object is constructed with its collaborating objects provided to it, using standard JavaBeans setter methods. The container manages the collaborators and removes the burden of doing lookups via ad-hoc methods. This allows programmers/users to vary what gets injected without changing the code.

In the J2EE model, objects depend on the server. The objects either subclass or implement interfaces from J2EE, or are tied to objects only available in the J2EE container. With Spring, objects typically have no dependencies to the Spring container. The objects are just POJOs that use JavaBean constructs.

KEY POINT: IoC gets rid of so-called “wiring code” that typically occupies 30% or more of many code bases. It allows for easy assembly of objects with their collaborators.

Spring and AOP: Dynamic Object Decoration

AOP Mainstream

Spring played a large role in making AOP mainstream and was the first AOP framework to actually build usable services with its AOP implementation

Reducing Code to Create Cross-Cutting Concerns

A key focus of the Spring framework is dealing with cross-cutting concerns. Spring’s AOP support allows developers to dynamically add services to objects, called “aspects.”

AOP enhances object-oriented programming; it does not replace it. AOP further allows developers to create non-domain concerns (called cross-cutting concerns) and mix them into their application code, without the knowledge of the collaborating object. With AOP, common cross-cutting concerns like logging, persistence, and transactions can be factored into aspects and applied to domain objects without complicating the object model of the domain objects.

AOP can drastically reduce the amount of code needed to write for cross-cutting concerns. AOP is a cleaner separation of concerns than traditional domain-model-only programming. Traditional programming of common concerns in the domain model is fragile and can lead to redundant code.

Consider this: If you have 10 domain objects and each has ten methods that need to be decorated with a common concern, then there are 100 places in which to imbed the same code.

If there are three lines of that code, then after insertion there are 300 lines of code, much of it redundant. If a change to the concern implementation is necessary, then 100 changes must be made, instead of one. Note: this refers to the implementation of a common concern; not to the domain object itself.

The core of the Spring Framework helps you configure objects with dependent objects. With Spring, objects do not have to know how to locate their collaborator objects; they need only use them as provided.

Domain objects are objects that deal with your business domain (vertical objects). Domain objects contain logic particular to your business domain. An example of a domain object could be an ***AutomatedTellerMachine*** object or a ***Library*** object. The domain object for an ***AutomatedTellerMachine*** would have the entire business rules particular to ***AutomatedTellerMachine***s.

AOP, on the other hand, implements cross-cutting concerns (concerns that cross domain boundaries). Functionality that is not core to a particular domain, such as that with horizontal objects, is called a cross-cutting concern. For example, both an ***AutomatedTellerMachine*** object and a ***Library*** object may use a transaction manager, but an ***AutomatedTellerMachine*** (ATM) has nothing to do with a Library. With Spring, you can extract the code specific to transaction management (or logging, auditing, security, etc.) out of domain objects and use them with ATM and Library objects as well as other domains.

Let's say you need to implement an auditing cross-cutting concern for your ATM machine and you do it in the traditional approach as demonstrated by figure 1-0-a. The ATM machine has 11 classes that implement a total of 77 methods that must be audited. Each method has the auditing logic and code in it.

Now--based on recent legislation, let's say that the auditing rules change. This now requires a great deal of changes to your code, right? As illustrated in figure 1-0b, you will need to modify 11 objects and 77 different methods.

If you use AOP instead, then you write only one Auditing aspect and apply that auditing Aspect to all 11 classes, as demonstrated by figure 1-0c. If the auditing rules change, you can make the change in one class (the class that implements the Auditing Aspect) instead of making the change 77 times! To target a different country, like Canada for example, you make only one change, instead of 77+ changes to your code base, by simply writing an extra Auditing class and applying it to the country of choice (in this case Canada) instead of the U.S. Auditing class.

FIGURE 1-0A. IMPLEMENTING A CROSS-CUTTING CONCERN THE NORMAL WAY

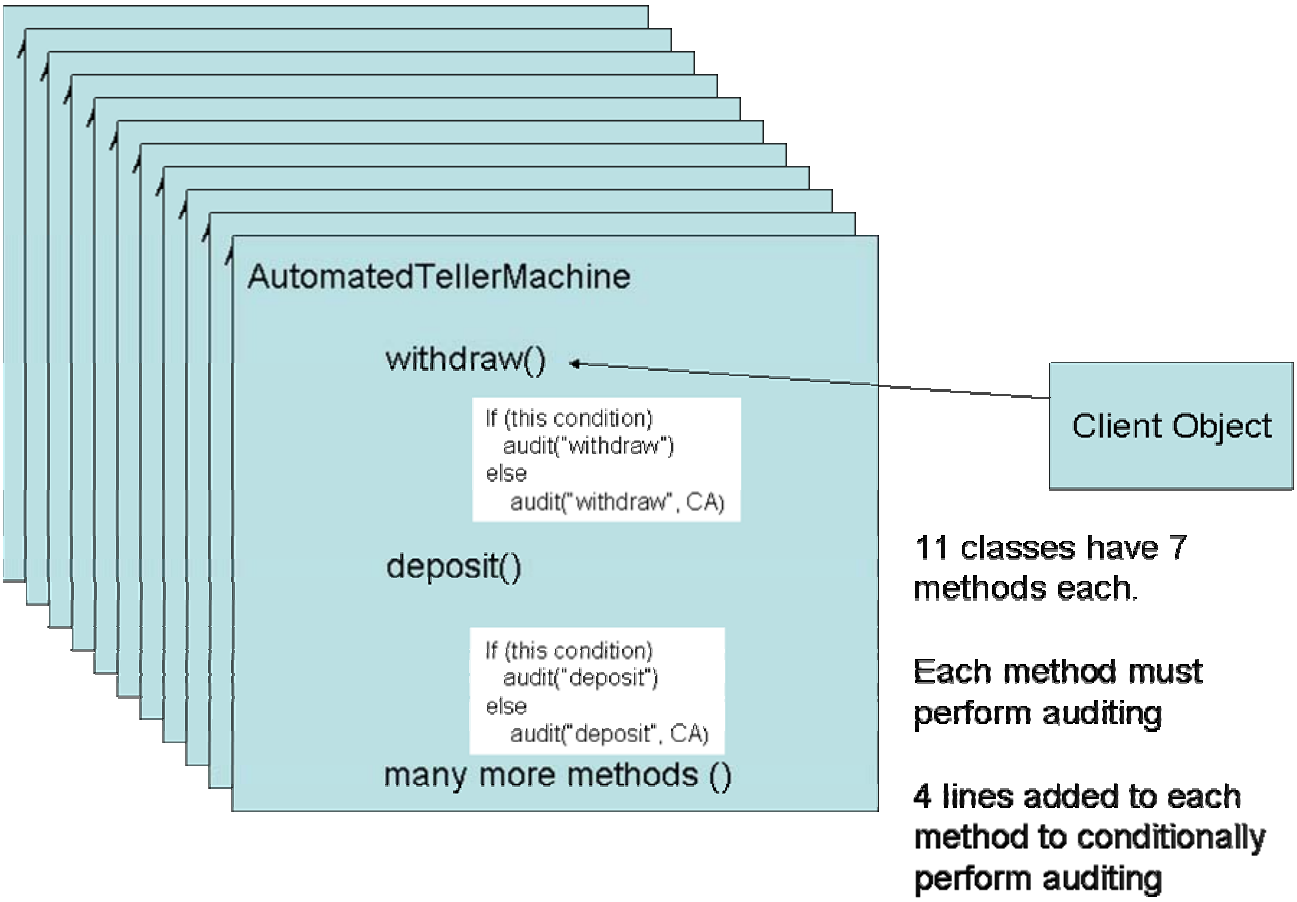


FIGURE 1-0B. IMPLEMENTING A CROSS-CUTTING CONCERN AUDITING RULES CHANGE

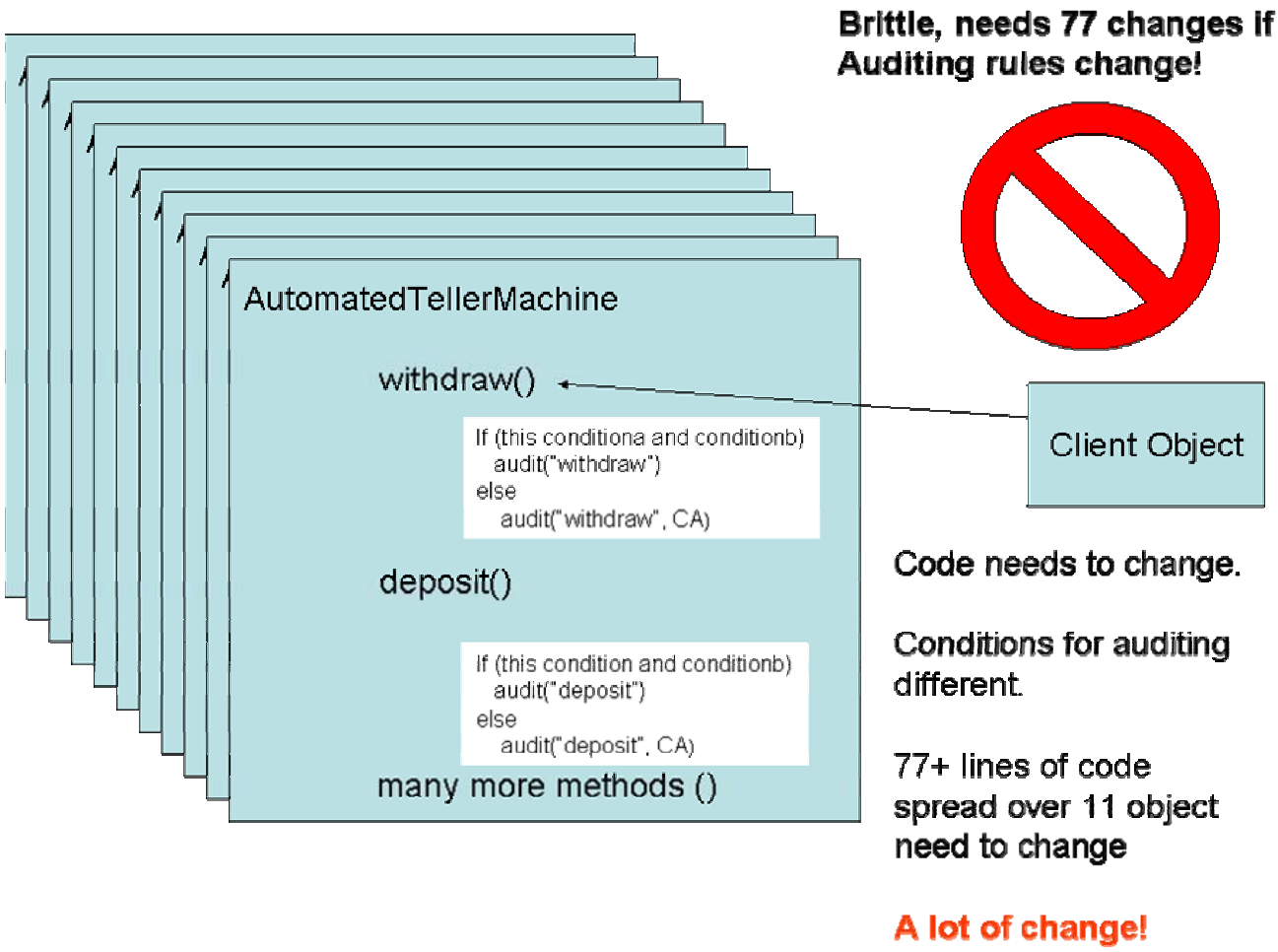
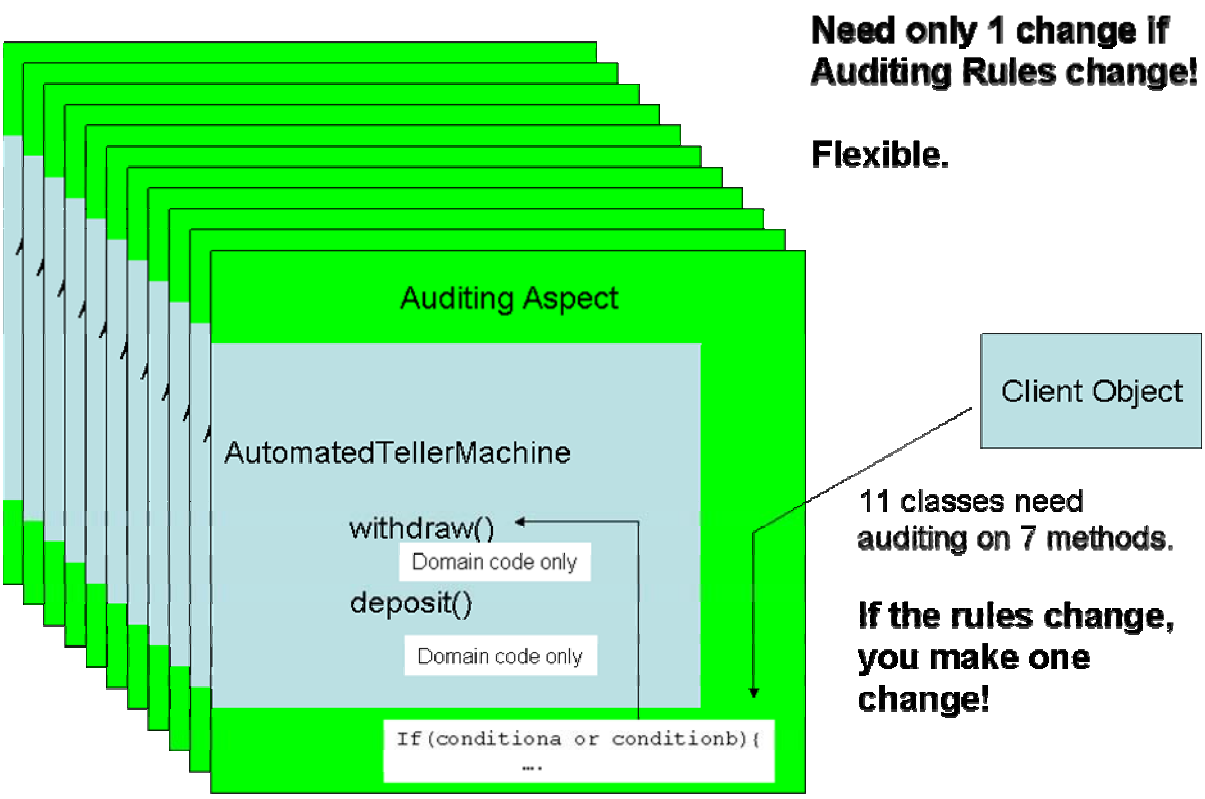


FIGURE 1-0C. IMPLEMENTING A CROSS-CUTTING CONCERN IN AN ASPECT VS. AN OBJECT



AOP as a Generic Event Mechanism

AOP makes OOP more extensible. It may be useful to think of AOP as a **dynamic event listener,** or as a **trigger-like mechanism for Java objects**. With AOP, a programmer can listen to any method invocation and perform actions--*before or after* the method gets invoked.

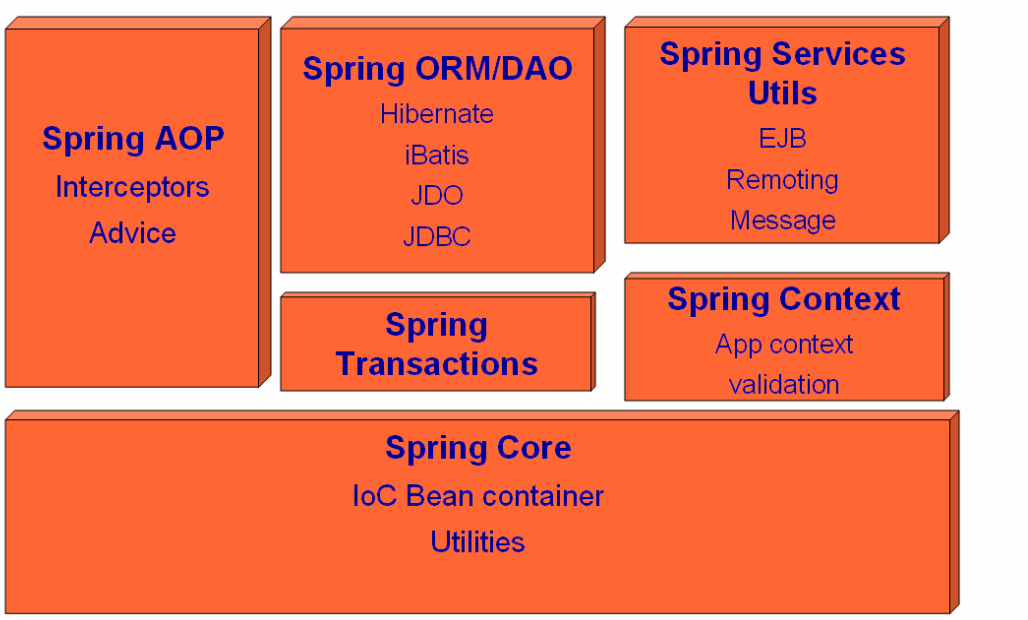
Vendors like SAP now publish ways to use AOP to listen to method invocation. If you need to know when a new employee gets added to the system, you can simply write an aspect. You can extend code in ways never envisioned by the original creators of the code base. In this way, **AOP makes OOP (object-oriented programming) more extensible.**

KEY POINT: AOP makes OOP (object-oriented programming) more extensible.

Spring: Simplifying J2EE Development

Spring uses IoC and AOP to provide a comprehensive library for simplifying J2EE development. Spring makes using other Java APIs and frameworks very easy to incorporate into applications. This comprehensive library is written with aspects and dependency injection. See figure 1.1.

FIGURE 1-1. BLOCK DIAGRAM OF SPRING FRAMEWORK



Spring Templates

Spring makes J2EE development easier with its use of Templates. At first glance, templates appear to be well-written utility classes. Spring templates, however, provide much more than just utility functions. A Spring template is a cross between a utility class and execution environment.

When using a template, you may start by using its utility methods. If the template doesn't have a utility method you need, however, then if using a Spring template, you simply implement a callback object, which executes a method in the environment of the template. The Spring template takes care of exception handling and resource management in a consistent manner. This creates a clean code base that is free of clutter, which is unlittered with try/catch/finally blocks.

Centralizing exception handling helps prevent programming mistakes, thereby improving the quality of your code base and reducing lines of code, making it more productive. **A quality code base is a more productive code base.**

Spring Template Callback Objects

The callback object that the Spring Templates use is another form of Inversion of Control. IoC is a foundation to OO programming. By using the callback object, you

are allowing the Template class to control the flow of the application and allow it to handle exceptions and therefore properly manage resources. This allows you to focus on what you want to do with a particular API (the interesting things) by letting the Template class do the routine things with the API, like resource cleanup and location.

KEY POINT: A quality code base is a more productive code base.

Spring's Template Callback objects reduce your code base and make it less error-prone by allowing the Template to do the cleanup.

Spring and Test-Driven Development

IoC capabilities of dependent object injection turns out to be a great mechanism for testing your code. With Spring, it is easy to inject mock objects (objects for testing) and test your classes in an isolated manner. For example, you can test your business objects without relying on the DAO (Data access objects) talking to the database.

Essentially, Spring has taken back development from the J2EE design pattern hacks that, in the past, were deemed necessary to program J2EE. Spring puts the OO back in J2EE development. Instead of creating code that is tied to a container, Spring frees your code.

Consider the following example of two programmers:

Programmer A is a traditional J2EE developer using EJB. He is dependent on his container to test his code. He must code, deploy, wait for deployment, test, and then repeat the process. He must also reboot his application server every now and then and wait some more.

Programmer B, on the other hand, is a lightweight developer using Spring, IOC, Hibernate etc. This developer can test, even against the db, without a container. His test cycle is code, test, code, test, etc. He gets more done and is likely to do work of higher quality.

KEY POINT: Spring and TDD: More Productive Than Traditional J2EE

Spring goes hand in hand with Test-Driven Development. True developer productivity is driven by the quality of the code base.

Higher Level Abstractions for a Whole Host of Projects

Object Relational Management (ORM)

Spring provides portability through abstraction of common services. For example, Spring provides a common interface for object relational management (ORM) systems like Hibernate, JDO, Cayenne, Spring JDBC and iBatis. It provides a mechanism for building DAO objects that divorces the client code from the underlying implementation. Spring does this by providing a common set of exceptions like the “object not found” exception, and making these exceptions “runtime” exceptions.

Support for industry-standard projects.

Spring provides an easy on-ramp for many industry-standard projects, and also for the de facto industry-standard projects--the projects that people actually use to get their daily work done. For example, Spring provides support for Hibernate, Quartz, Hessian and many other projects.

Simplified J2EE development

Spring simplifies J2EE development, and provides utilities to work with all tiers of an n-tier application. For an MVC application, there are utilities for working with View technologies (Struts, Spring MVC, JSF, Tapestry, Swing Rich Client etc.), Model (EJB, AOP based transaction, AOP based security, etc.). You can use Spring to build Swing and SWT applications just as easily.

Show Us Some Code!

Because the primary purpose of this paper is to give you a bird's eye view of the Spring valley, we will show you some of the sample code without attempting to cover all of the details here. In later papers, however, we will address specific questions in regard to the Spring valley.

If the examples don't make sense yet, realize that this is the first in a series of papers, and subsequent publications will explain them in more detail. For now, I ask that we look at the forest in the valley. Later we will examine the bark on the trees.

IoC for Developers

IoC allows developers to easily assemble objects with their collaborators, eliminating myriads of service locators, abstract factories, and singletons, which are forms of wiring code. This wiring code typically occupies 30% or more of many code bases.

Spring's core functionality serves to provide a central repository for finding objects by name called an ApplicationContext. Your applications will commonly create the

ApplicationContext in a centrally-located place where it is accessible to any object that requires its services (via a static method in a Rich Client application, for example, or the ServletContext in a web application). Since Spring encourages configuration of objects without having to know Spring APIs, you will usually only use the Spring API when you want to use the objects in a ApplicationContext; not in the configured objects themselves.

KEY POINT: You may never need to use the BeanFactory API

The ApplicationContext is a BeanFactory. You could extensibly use Spring and never access the BeanFactory API or ApplicationContext directly. Since Spring integrates with JSF, Tapestry, Struts, WebWork and more, you can write applications which get their dependencies injected and never need to access the BeanFactory or ApplicationContext directly.

Spring can provide services like declarative transactions, declarative security, and more, much like EJB. Unlike EJB, however, beans defined in the Spring container are independent of Spring; you don't have to implement any special interface or subclass any special base class like you do with EJB. Beans can be configured and managed by Spring without depending on Spring; thus, the beans are decoupled from the container. Spring can inject dependencies, also called collaborative objects. Collaborators are objects that beans needs to complete their roles. Spring uses regular JavaBean and Java language constructs to inject dependencies. Beans managed by Spring are plain old Java objects (POJOs).

Decoupling Beans from the Underlying Container

Decoupling beans from the underlying container allows those beans to be unit-tested outside of the container. Also, since Spring is such a lightweight container, it is easily run inside of an IDE, without deploying anything to a container, which also facilitates unit testing and test-driven development.

KEY POINT: Spring can save time in the development and testing process.

One of the biggest slow downs in J2EE development is the "deploy, wait, and test" cycle, with an occasional obligatory reboot of the application server (which requires even more waiting).

Let's show a short example how a bean is configured in the Spring application context. This example demonstrates an ***AutomatedTellerMachine*** (a banking machine). The ***AutomatedTellerMachine*** needs a ***transport*** object to send credit and debit information back to the bank. In other words, it needs the ***transport*** object to fulfill its role. Here is the ***AutomatedTellerMachine*** which we will declare in Spring's application context:

```
package examples;
```

```
    /** @author Richard Hightower from ArcMind, Inc. */
public class AutomatedTellerMachine {
    private ATMTransport transport;

    public void setTransport(ATMTransport transport) {
        this.transport = transport;
    }

    public void withdraw(float amount) {
        transport.connect();
        transport.send(...);
        ...
        transport.disconnect();
        System.out.println("CREDIT");
    }

    public void deposit(float amount) {
        transport.connect();
        transport.send(...);
        ...
        transport.disconnect();
        System.out.println("DEBIT");
    }

   ...
```

Here is the ***AutomatedTellerMachine*** declared in Spring's application context:

```
<beans>
    <bean id="transport" class="examples.SoapSslATMTransport"/>

    <bean id="atm" class="examples.AutomatedTellerMachine" >
        <property name="transport" ref="transport"/>
    </bean>

</beans>
```

Notice that the ***atm*** (bean id="atm"), gets past the ***transport*** object (bean id="transport) by using the property tag. The ***AutomatedTellerMachine*** Java class has a JavaBean property called ***transport***. Having a setter method called ***setTransport*** is similar to the ***AutomatedTellerMachine*** declaring that it needs

the ***transport*** object to fulfill its role (see class diagram figure 1.2). This is demonstrated pictorially in the next few figures.

FIGURE 1-2. UML DIAGRAM OF AUTOMATEDTELLERMACHINE AND COLLABORATORS

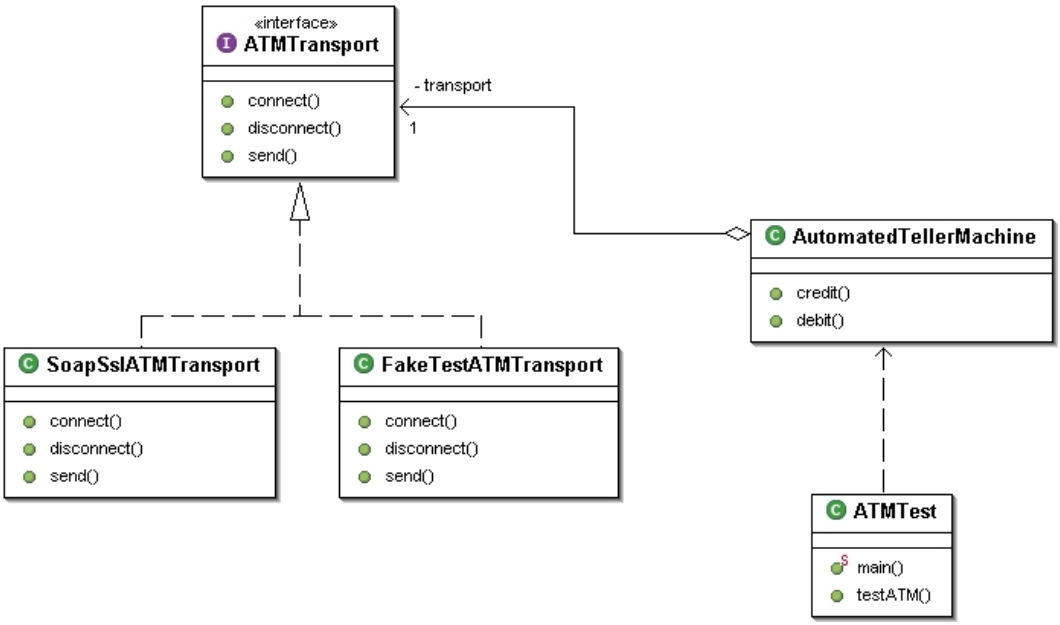


Figure 1-2 is a UML diagram that shows the ***AutomatedTellerMachine*** class which contains a ***transport*** reference (the ***transport*** property) that is of type ***ATMTransport*** (an interface). There are two implementations of the ***ATMTransport*** interface (***SoapSslATMTransport*** and ***FakeTestATMTransport***).

FIGURE 1-3. INJECTION BREAKDOWN (1 OF 2)

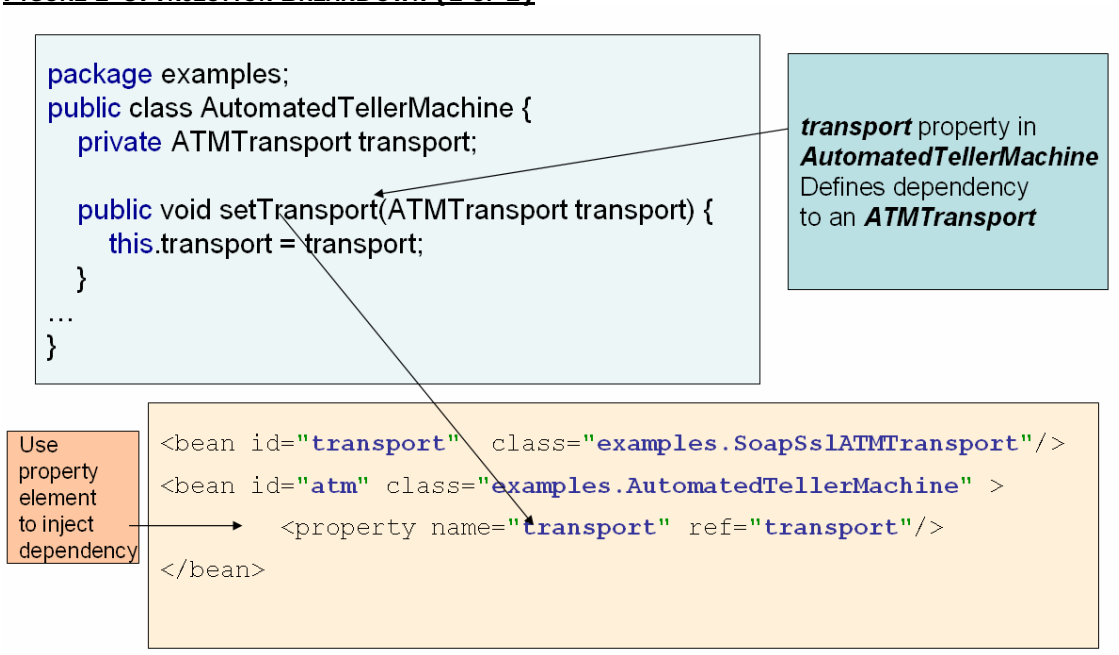


Figure 1-3 shows that the property element of the ***atm*** bean is used to configure the injection of the transport property.

FIGURE 1-4. INJECTION BREAKDOWN (2 OF 2)

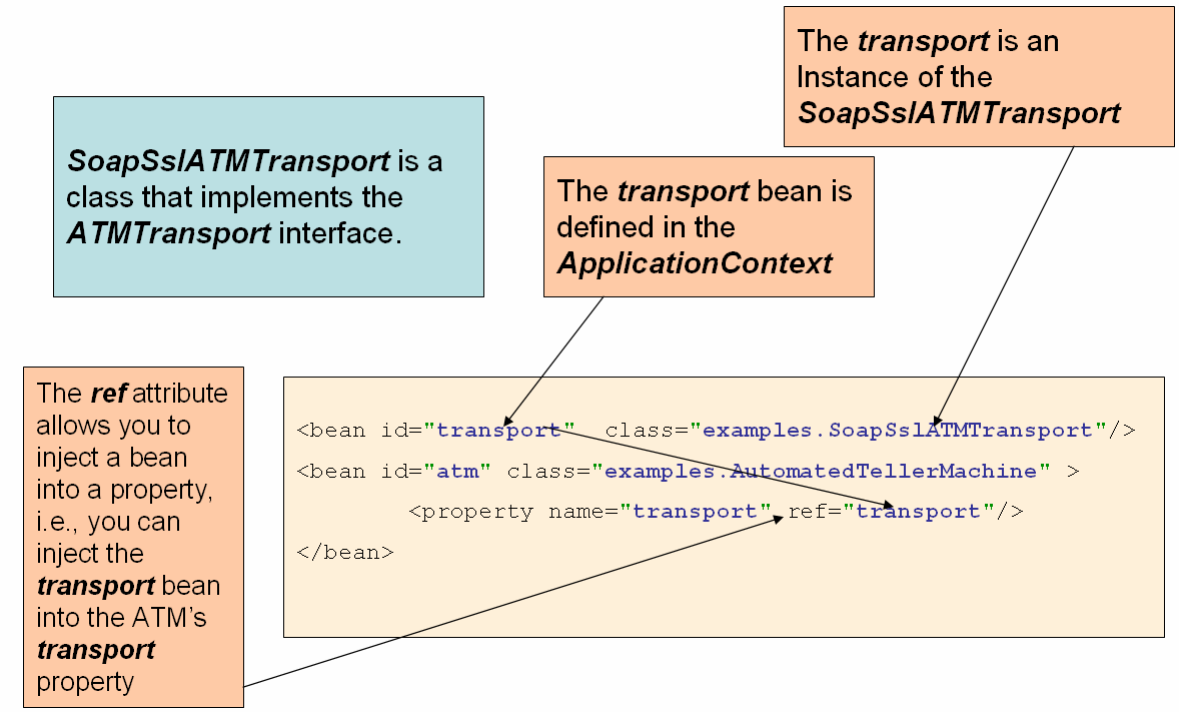


Figure 1-4 shows that the property element of the ***atm*** bean is used to configure the injection of the transport property. An instance of ***SoapSslATMTransport*** will be injected (the setter method will be called, passing an instance of the ***SoapSslATMTransport***).

Notice that the ***AutomatedTellerMachine*** is a domain object that uses the ***transport*** object (a collaborator) to complete its role. Likely, the ***AutomatedTellerMachine*** implements business rules that you would like to test, without requiring it to talk to a real transport. To facilitate testing, you may swap in a fake transport to test the business rules of the ATM machine, demonstrated as follows:

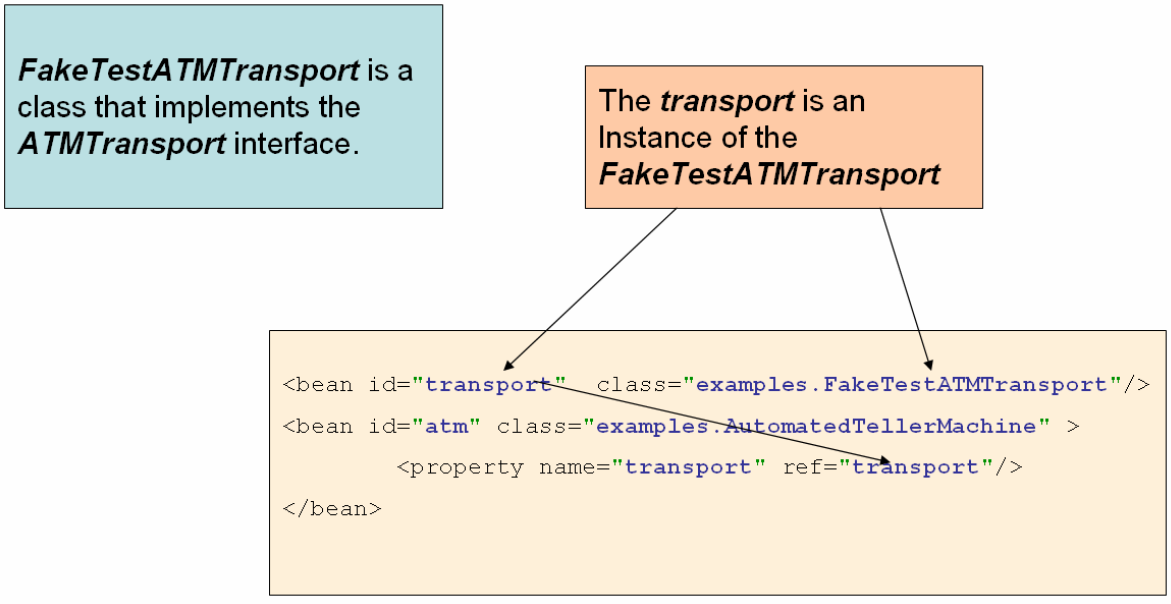
```
<beans>
    <bean id="transport"
              class="examples.FakeTestATMTransport"/>

    <bean id="atm" class="examples.AutomatedTellerMachine" >
        <property name="transport" ref="transport"/>
    </bean>

</beans>
```

Notice here that we swap out the ***SoapSslATMTransport*** with ***FakeTestATMTransport***, allowing you to write tests that focus on the business rules of the ATM, instead of testing the ATM and Transport together (dividing and conquering). This is how Spring supports test-driven development. It allows collaborative objects to be easily swapped out with test stubs. Remember the motto, "The great thing about objects is that they can be replaced." Figure 1-5 shows this concept pictorially.

FIGURE 1-5. INJECTION OF A TEST OBJECT



Client objects that use the ***atm*** object will look up the ***atm*** object in the application context as follows:

```
import org.springframework.context.ApplicationContext;
...
    ApplicationContext context = ...
    AutomatedTellerMachine atm =
            (AutomatedTellerMachine)context.getBean("atm");
    ...
    atm.deposit(100.0f);
    ...
```

Keep in mind that this is just a brief introduction to the IoC container capabilities of Spring. Step-by-step instructions, and more detailed coverage of Spring's IoC support will be provided in the second and third papers in this series.

AOP for Developers

Domain objects are objects that deal with your business domain. Domain objects contain logic particular to your business domain, such as our example dealing with an ATM machine. The domain object would have all of the business rules particular to ATMs. Aspects, on the other hand, implement cross-cutting concerns (concerns that cross domain boundaries). For example, both an ATM and a Library object may use a transaction manager, but an ATM machine has nothing to do with a Library. If you are familiar with Design Patterns, an easy way to think about aspects is to think of them as Dynamic Decorator Patterns.

KEY DEFINITION: Dynamic decorators are often called “AOP proxies” in Spring terminology. The Decorator Design Pattern is a common design pattern that is used quite often to extend functionality of an object without polluting that object with the extension.

> “The decorator pattern allows new/additional behavior to be added to an existing method of an object dynamically. This is done by wrapping the new "decorator" object around the original object, which is typically achieved by passing the original object as a parameter to the constructor of the decorator, with the decorator implementing the new functionality. The interface of the original object needs to be maintained by the decorator.”
>
> “Decorators are alternatives to subclassing. Subclassing adds behavior at compile time whereas decorators provide a new behavior at runtime.”
>
> Wikipedia http://en.wikipedia.org/wiki/Decorator_pattern

Spring’s AOP support allows you to dynamically add services to objects called aspects. This is similar to the Decorator Design Pattern, but does not require you to recompile your code base to apply these services. In this way, it is a Dynamic Decorator Pattern. It allows you to replace objects with objects that enhance the originals with additional behavior.

KEY POINT: Spring increases productivity and reduces the size of your code base by eliminating redundancies.

Spring allows you to create decorator objects “on the fly,” instead of requiring you to write code for each decorator.

Aspects are usually services that can be applied to multiple domains. Aspects are applications of services like transactions, auditing, security, critical exception handling, etc.

AOP is another way to separate areas of concern in your application. Just like exception and catch blocks allow you to separate your "go code" from your error handling code, AOP allows you to separate your services code (transactions, auditing, security) from your domain logic code. This is a boon for reuse and maintainability of your code base.

Returning to our ATM example, let's say that depending on which country our ATM is in (Canada, or the U.S., for example,) you have to perform different levels and styles of auditing; therefore, you want to abstract how auditing is done from the ATM machine. Perhaps some locations do not need auditing at all. Let's say that the auditing can be used with other domain objects in your business as well. Here is how we would write an Aspect to audit our ATM machine:

DEVELOPER NOTE: If a class ends in FactoryBean

If a class ends in the camel case words “FactoryBean” it means that Spring will use this object to create another object. A FactoryBean is a bean that knows how to create another bean. Thus, a ***ProxyFactoryBean*** knows how to create an AOP proxy (something we have been calling a dynamic decorator).

```
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">

<beans>
    <bean id="transport" class="examples.SoapSslATMTransport"/>

    <bean id="atmTarget"
                class="examples.AutomatedTellerMachine" >
        <property name="transport" ref="transport"/>
    </bean>

    <bean id="auditor"
```

```
            class="examples.services.AuditingUSAspect"/>

    <bean id="atm"
   class="org.springframework.aop.framework.ProxyFactoryBean">
          <property name="target" ref="atmTarget"/>
          <property name="interceptorNames">
             <list>
                <value>auditor</value>
             </list>
          </property>
    </bean>

</beans>
```

A common complaint in the industry is that all AOP examples deal with logging and auditing, and that these examples are trivial at best.

In papers prior to this one, we wrote that in dealing with Spring AOP, we developed a ***SecurityManager*** aspect.

For more information on our other Spring papers, contact us, or visit our web site for more details. http://www.arc-mind.com

Notice that instead of getting the ATM object directly, you get the ***atm*** object through the ***ProxyFactoryBean***. When the client code gets the ***atm*** object, it gets the ***atm*** domain object decorated with the auditor aspects behavior. Figures 1-6 through 1-8 explain this pictorially.

FIGURE 1-6. BEFORE AOP ADDS AUDITOR SUPPORT

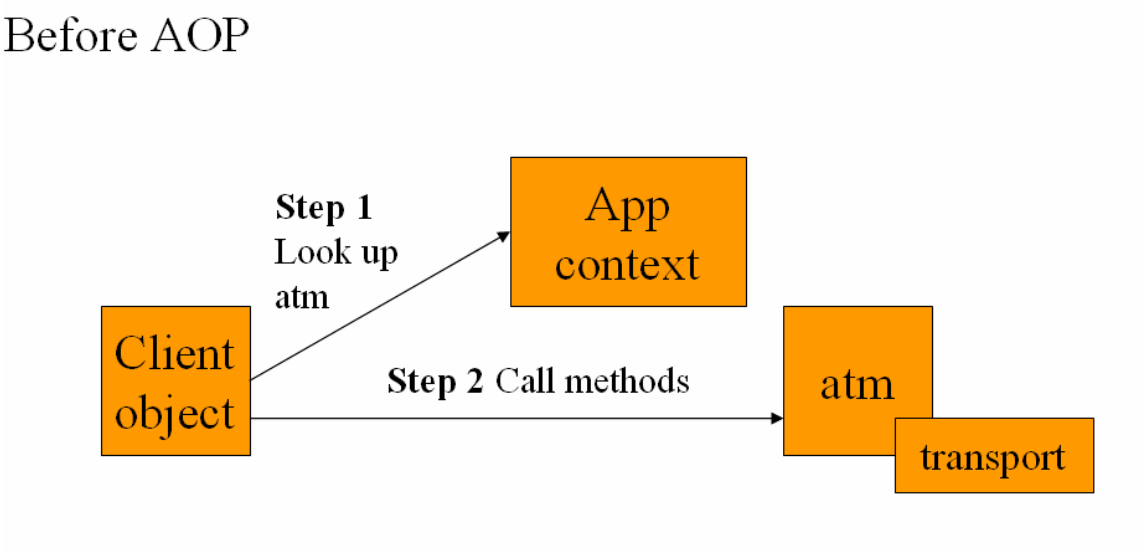


FIGURE 1-7. AFTER AOP ADDS AUDITOR SUPPORT

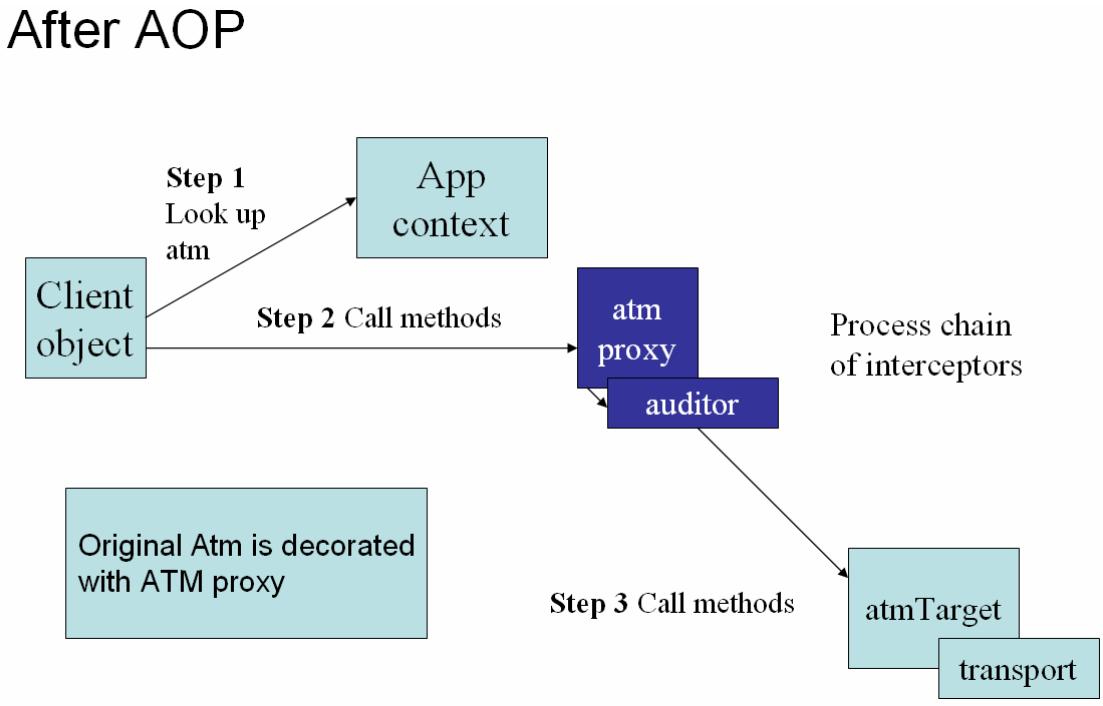
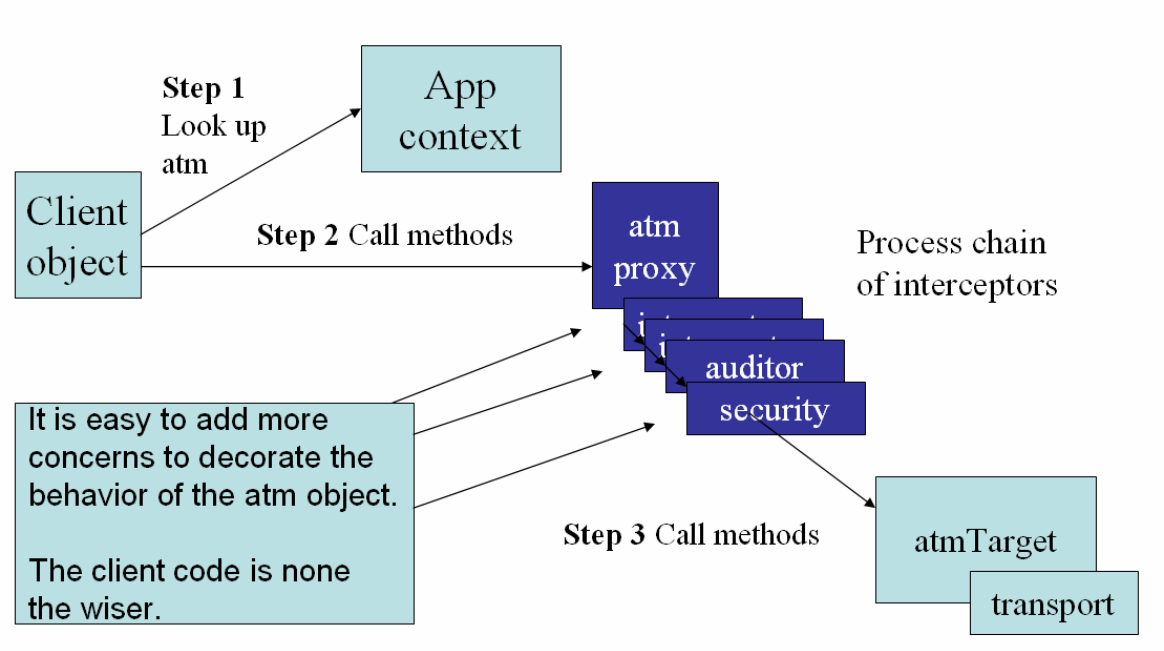


FIGURE 1-8. MORE EASY-TO-ADD CONCERNS



The client code does not know or care that the atm object is being decorated with new behavior by the aspect. The ***ProxyFactoryBean*** associates the aspect (the auditor interceptor in the ***interceptorNames*** list) with the ***AutomatedTellerMachine*** object (the ***atmTarget*** passed to the target property). The ***ProxyFactoryBean***, therefore, decorates the behavior of the ***AutomatedTellerMachine*** with the aspect (the auditor). The auditor aspect is implemented as follows:

```
package examples.services;

import java.util.ArrayList;
import java.util.List;
import java.text.MessageFormat;

import org.aopalliance.intercept.MethodInterceptor;
import org.aopalliance.intercept.MethodInvocation;


/** @author Richard Hightower from ArcMind Inc. */
public class AuditingUSAspect implements MethodInterceptor {

    /**
     * Gets called for every method invocation on target
     * @param methodInvocation
     * @return return code
     */
```

```
    public Object invoke(MethodInvocation methodInvocation)
        throws Throwable {
        /* Get the name of the method being invoked. */
        String operationName =
                 methodInvocation.getMethod().getName();

      /* Grab the arguments passed to the method invocation. */
        Object[] arguments = methodInvocation.getArguments();
        List argList = new ArrayList();

        if (arguments != null) {
            for (int index = 0; index < arguments.length;
                                                 index++) {
                argList.add(arguments[index]);
            }
        }

        /* Invoke the method or next Interceptor in the list */
        Object returnCode = methodInvocation.proceed();

        /* Create the message to log */
        String messageToLog =
            MessageFormat.format(
                 "{0} called with arguments {1} returned {2}",
                new Object[] {operationName,
                              argList, returnCode}
            );

        /* Log the method invocation message,
           return the results */
        ...

        return returnCode;
    }
}
```

Remember--don't sweat the details at this time, as we will cover this more fully in future papers.

Please notice that the auditing aspect has no real knowledge of the underlying domain. You could use this same auditing object with a different domain object.

By modeling a common concern such as an aspect, instead of embedding redundancies in your code base, a savings of 10x, 20x, or even 30x is not unrealistic. Also, what if you need to add new concerns that you have not thought

about before? With AOP, you create an aspect and apply it to your domain objects; you don't change the entire code base.

KEY POINT: If you consider the alternative, you will recognize the value proposition of AOP!

The alternative to our approach is to repeat the code in each method of each domain object. Even considering something less trivial like transaction management and imagining that code repeated in every method, one can easily see that it is error-prone and repetitive, to say the least. AOP is a method by which you can reduce your code base and defects.

Please visit our website for more technical papers like this one. ArcMind (http://www.arc-mind.com).

If our ATM machine is in Canada, and we have to follow the Canadian auditing rules, you only have to change one line in our application context file:

```
<bean id="auditor" class="examples.services.AuditingUSAspect"/>
```

becomes:

```
<bean id="auditor" class="examples.services.AuditingCAAspect"/>
```

Basically, you pass ***AuditingCAAspect*** instead of ***AuditingUSAspect***. The rest of the code remains the same. The client code that uses the atm object does not even know that you changed the auditing implementation. The code that is being decorated does not know that you changed the auditing implementation either. Remember: The great thing about objects is they can be replaced!

Another important benefit of using AOP with Spring is that it is non-destructive. When you write Java classes and compile them, and then deploy the code, your code is unsullied. Classloaders aren't polluted with incompatible versions of the same object. When you want to use AOP, the AOP proxy Java class is generated and has the modifications applied to it, but the original class, as it stands, is never modified. It is only changed when you want it to be, and only within the context of the Spring container.

KEY POINT: Spring is non-destructive.

When you write Java classes and compile them, and then deploy the code, your code is unsullied.

Templates for Developers

Templates simplify the use of some API, usually to an enterprise service. Spring templates help you to avoid common mistakes. They provide a common template for performing a task like sending a message, or querying a database. You focus on sending the message or performing the query and not handling the common invariant behavior. The invariant behavior is handled by the Spring template. If the workflow is fairly complex, you may have to implement a callback object that will be called at the appropriate time (which is another form of Inversion of Control).

Spring templates are similar to the Template Method Design Pattern. Code that uses a Spring template, however, may need to implement callback interfaces. Spring clearly defines the contract for dealing with its templates much more formally than the traditional template method from the classical Template Method Design Pattern.

Spring has template classes for dealing with Hibernate, JDO, JDBC, JMS, JMX, JNDI, iBatis, transactions and more. The concepts of templates are core to Spring. **Essentially, whenever there is an error-prone workflow, Spring creates a template to allocate resources, clean up resources and properly handle exceptions.** This allows you to write code that is not littered in try/catch/finally blocks each time you want to use an enterprise service. Templates are one of the main ways Spring makes J2EE development easier. Spring templates can greatly improve the quality of your code base by reducing defects.

By way of example, let's demonstrate using Spring JDBC template support. Concepts introduced with ***JDBCTemplate*** class will be applicable to other Spring templates. At a basic level, a template can be viewed as a utility class as follows: (read the comments)

```
JdbcTemplate template =
             new JdbcTemplate(dataSource);

//Perform a one line update
template.update(
    "update EMPLOYEE set FIRST_NAME=? where id=?",
    new Object[] {"Rick", new Integer(3)}
);

//Perform a one line query returning an int.
int count = template.queryForInt(
   "select count(*) from EMPLOYEE");
System.out.println(count);

//Perform a one line query returning a name (String)
String name =
    (String) template.queryForObject(
     "select FIRST_NAME from EMPLOYEE where id=3",
        String.class
```

```
            );
        System.out.println(name);

        //Get all of the employees back from the table
        List employees =
            template.queryForList(
             "select id, phone, first_name, last_name " +
             "from EMPLOYEE");

        for (Iterator iter = employees.iterator();
                                    iter.hasNext();) {
            Map currentRow = (Map) iter.next();
            Long id = (Long) currentRow.get("id");
            String phone = (String) currentRow.get("phone");
            String firstName = (String)
                        currentRow.get("first_name");
            String lastName = (String)
                            currentRow.get("last_name");
            String message =
              MessageFormat.format(
              "Employee [id={0}, phone={1}, name={2} {3}]",
                new Object[] {id, phone, firstName, lastName}
               );
            System.out.println(message);
        }
```

As you can see, you can easily work with the Employee table. You can get a count of employees, update employees, query employee data, and even list employees with relative ease. Note that try/catch/finally blocks do not exist, and therefore do not litter the code base, making it easier to read. The Spring JDBC template is handling the errors and cleaning up the resources appropriately. The above usage may remind you of working with a high-level scripting language instead of something you would expect with Java.

But what happens when you want to execute a particular operation not supported by the template? You register a callback object. A callback object runs in the context of the template object. The template object calls the callback object at the appropriate time. It prepares the environment for the callback object, and cleans up after the callback object is done. Here is an example how JDBC allows you to register callback objects with it:

```
/* Get a list of employees */

        //Create callback object
        ResultSetExtractor extractor =
            new ResultSetExtractor() {
                public Object extractData(ResultSet resultSet)
```

```
                    throws SQLException, DataAccessException {
                    List employeeList = new ArrayList();

                    while (resultSet.next()) {
                        Employee employee = new Employee();
                        employeeList.add(employee);

                        employee.setFirstName(
                              resultSet.getString("first_name"));
                        employee.setLastName(
                           resultSet.getString("last_name"));
                        employee.setPhone(
                           resultSet.getString("phone"));
                    }

                    return employeeList;
                }
            };

        /* Execute callback object */
        List employeeList =
            (List) template.query(
       "select id, phone, first_name, last_name from EMPLOYEE",
                extractor
            );

        for (Iterator iter = employeeList.iterator();
                                            iter.hasNext();) {
            Employee employee = (Employee) iter.next();
            System.out.println(employee);
        }
```

The above example creates a special callback object to extract employee objects from the result set from the query. The ***JDBCTemplate*** has many types of callback objects. There are even callback objects to create objects like prepared statements as follows:

```
/* Create a prepared statement and use the same extractor */
PreparedStatementCreator lookUpEmployeeLastNameLikeStatement =
   new PreparedStatementCreator() {
         public PreparedStatement
           createPreparedStatement(Connection connection)
                                        throws SQLException {
             return connection.prepareStatement(
                "select id, phone, first_name, last_name  " +
                " from EMPLOYEE where last_name like 'H%'");
```

```
        }
   };

...


/* Execute callback method */
List employeeList2 = (List) template.query(
               lookUpEmployeeLastNameLikeStatement, extractor);

for (Iterator iter = employeeList.iterator(); iter.hasNext();){
            Employee employee = (Employee) iter.next();
             System.out.println(employee);
}
```

Template objects are often constructed and passed to other objects via the Spring IoC container.

Template objects are proof-positive that Spring is a lot more than a buzzword-compliant framework. Spring uses the best practices of OO, IoC and AOP to simplify your development tasks. It takes design patterns a step further and makes them more applicable to your application development efforts.

KEY POINT: No fluff--just what you need to get started!

If you find the content of this paper interesting, you may really like our custom, lab-centric, JSF, Hibernate and Spring Framework training! Arc-Mind courses are taught by developers with real-world experience. Our courses are designed to get you started quickly. No Fluff!

Visit our website for information on customized, on-site training: ArcMind (http://www.arc-mind.com).

Conclusion and Parting shots

Spring is much more than another buzzword-compliant framework. Spring simplifies J2EE development and supports test-driven development. Spring does this by improving on OO concepts and augmenting them to make them more adaptable.

The Decorator Design Pattern is good; however, Spring makes it more flexible and dynamic through AOP.

The Template Method Design Pattern is a good technique for reuse, but Spring makes it more adaptable via well-defined contracts (callback objects).

Spring simplifies the use of many standard and de facto tools by building the Spring framework on top of its implementation of IoC, AOP, Templates and good OO design principals.

AOP can ***drastically reduce code*** that would otherwise be repeated. AOP is not a replacement of OOP; it is an augmentation of OOP. AOP adds abilities that did not exist before in mainstream programming languages.

AOP allows rank-and-file developers to add services/concerns to any object--not just objects managed by a container that implement a certain interface. This allows you the ability to remove mundane code from your domain objects and easily apply future concerns without the complications of traditional approaches.

IoC allows you to easily assemble objects with their collaborators, getting rid of so-called “wiring code” that typically occupies 30% or more of many code bases. The IoC container allows you to organize applications so that objects are created with their dependencies. This removes the object wiring associated with traditional projects and makes test-driven development and unit testing much more feasible.

Spring Templates encapsulate the logic for resource management and tricky exception handling so the logic can be reused. This encapsulation replaces common and error-prone code, reducing the possibilities of defects that leak resources.

If you are not yet using the Spring framework, it is time to consider doing so. Spring will not only increase your productivity, but save you time and money. And since the Spring framework is non-prescriptive, it is easy to work it into the mix of any project.

And the bottom line is this: Spring saves you time, which saves you money.

Lab Centric Courseware!

- If you find the content in this paper interesting, and would like more information, I encourage you to consider getting customized and detailed instruction in our lab-centric, JSF, Hibernate and Spring Framework training Our courses are taught by developers with real-world experience. Visit our website for information on customized onsite training: ArcMind (http://www.arc-mind.com).

- To receive a copy of other white papers about the Spring framework, please contact us See the page “About ArcMind” for ways to contact us.

- Special thanks to Thomas Bridges, Alex Redington, Kris Diefenderfer, Mary Basset, Martha Pena, Andrew Barton and Drew Davidson for their contributions to this paper.
- Special thanks to Rod Johnson, Juergen Hoeller, Colin Sampaleanu, Rob Harrop and the rest of the Spring core team for creating this amazing framework that took IoC and AOP out of the realms of academic discussion and made them mainstream. You have changed the way software will be developed for some time to come. You have made me a better developer.
- Thanks to Tom Dyer, whose enthusiasm for Spring turned me on to using Spring, and Paul Tabor who let me use Spring for the first time on a project for his company.

http://www.arc-mind.com

Check our website for onsite training and whitepapers:
ArcMind (http://www.arc-mind.com).

"I attended ArcMind's training class on Spring/Hibernate - I can say without exaggeration that it is the ***best technical training*** I have ever attended! … ***I wish all classes were like this!***"

--Deepa
Senior Software Engineer, eCommerce company, Dallas, TX.

"The instructor delivered what was hands-down the ***best technical training*** I've ever received. (Spring, Hibernate, and JSF 5 day QuickStart.)"

--Aaron
Principal Software Engineer, eCommerce company, Los Angeles, CA

Read the next page to learn more about ArcMind, Inc..

http://www.arc-mind.com

Check our website for onsite training and whitepapers:
ArcMind (http://www.arc-mind.com).

About ArcMind

ArcMind is a premier provider of advanced training. ***ArcMind*** is a full-service, software development company that will help you and your company succeed.

ArcMind provides systems integration, consulting and mentoring services. Service to Global 1000 companies with a primary focus on J2EE, JSF, Spring, and Hibernate.

Our niche is the application of Agile practices of continuous integration, unit testing and J2EE development (soon JEE 5). Our focus on Agile Methods and TDD allows our customers to deliver business value to clients in the shortest time possible while reducing risk.

Unlike other consulting firms, our objectives do not require your company to hire more consultants. Although ***ArcMind*** is a consulting firm, our focus is on *building your team* through mentoring and training. Our experienced consultants *mentor* your team until outside resources are no longer needed.

We put members on your team "*who have been there and done that*" with J2EE/Struts, .Net and Agile Methods. We share a unified goal of keeping project control within your team and company. We mentor your team through tangible milestones so project satisfaction is assured.

ArcMind trainers are real world developers and instructors who can help you deliver your next project in a timely fashion. Our trainers are experienced developers who have worked with the technologies and have real-world experience with the techniques that they are training.

Our trainers are available to your teams as mentors and consultants saving you valuable ramp up time because they have "been there and done that before".

Developers need answers – not marketecture. ArcMind, Inc. provides an independent voice devoid of marketecture.

Technology is changing fast. Our *specialty workshops* and comprehensive educational experiences compliment your development team with the skills they need to take on tough development challenges.

Our full time **Courseware Development Team** devotes itself to keeping up-to-date curriculum available for you. We tailor courseware to suit your needs. We will build your curriculum, whether product or technology specific, from the ground up.

Web Site & ArcMind News!

Visit our site at www.arc-mind.com for information about our new Spring, Hibernate and JSF courseware. You are invited to download free technical papers on J2EE development.

This site provides detailed information on our special offers, mentoring and consulting programs, current course lists, public training schedule, individual course details, and licensing information.

Our Java Curriculum

- Java Sever Faces (JSF)
- Java Data Objects
- Hibernate (2 or 3)
- Spring (covers IOP, and AOP)
- Tapestry
- Ajax
- EJB 3

ArcMind
Office 520.290.6855
info@Arc-Mind.com
http://www.Arc-Mind.com

What people are saying about ArcMind Training and consulting:

"ArcMind taught a one week long Spring/Hibernate class for us at Sabre. They did a great job keeping the class entertaining while covering a difficult material. ArcMind's class covers in great details the topic at hand and includes numerous labs to help practice each lesson. Highly recommended!"
– Jacques Morel (Chief Architect, Sabre Systems, Dallas TX)

"We hired ArcMind at the beginning of a major Java development initiative. This was some of the best money we have spent. Their consulting on tool selection, organization of our project and other areas laid a foundation that we continue to benefit from. They worked with us onsite, remotely, and through an ongoing relationship."
–Tim Hoyt (CTO of Picture Marketing, Miami, FL)

"(The instructor's) knowledge of J2EE, Java and Open Source technologies is both deep and wide. His training and mentoring skills are the best that I've come across. His passion and proficiency has been inspiring for me and many others in the development community. ArcMind's input and guidance was crucial when my organization needed to determine a strategy for leveraging new technologies a few years ago."
–Tom (Senior Software Developer, Boston MA)

"I've been using them for a while, so when I was sent to ArcMind's Spring/Hibernate class, I thought it'd be easy. Turns out I learned a lot. The coursework was well organized and clearly presented with lots of labs. More importantly, the instructor shared his extensive knowledge and experience in using these products in real-world projects. Highly recommended, even for experienced Spring/Hibernate users!"
– Rong (Senior Software Engineer at an eCommerce company Dallas, TX)

"The instructor is an expert with in-depth knowledge and understanding of the J2EE concepts and technologies. Additionally his expert advice on the integration of Spring and Hibernate was a great help."
--Rohit (Principal Software Engineer at a Manufacturing company, Toronto Canada)

"The instructor was an excellent teacher, thoughtful architect/programmer and absolutely knows his business inside and out. I would highly recommend ArcMind to anyone!"
--Danilo (Principal Software Engineer, CA)

"Well when it come to Hibernate/Spring, I feel (the ArcMind instructor) is someone who is like 'Been There Done That' kind of a guy. I really enjoyed his sessions."
– **Prasanth** (Senior Software Developer)

More endorsements available upon request.