# Using a Framework to Take the Work out of Creating Web Services

Department of Computer Science
Rutgers, the State University of New Jersey

April 2004

## Introduction

One of the main goals of standards such as SOAP or XML-RPC is to mimic the idea of making a local procedure call remotely. Ideally, one should be able to call a procedure (or method) on a remote object as if he or she is calling that same method on a local object. Unfortunately, current implementations of packages to do SOAP and XML-RPC do not accurately portray the idea of a simple procedure call. Unless the developer writes his own wrapper implementations each time, the calls are hardly transparent.

The example above shows a trend in developing web services. There is no simple way to efficiently handle the many aspects of programming required to create a web service. Wrappers for procedure calls must be written each time. JDBC code is often copied and pasted and just changed slightly. This adds unneeded complexity to already complex web services.

We define frameworks as: "The software environment tailored to the needs of a specific domain. Frameworks include a collection of software components that programmers use to build applications for the domain the framework addresses. Frameworks can contain specialized APIs, services, and tools, which reduce the knowledge a user or programmer needs to accomplish a specific task."[16] Frameworks are designed to speed the development of applications for which the framework was designed. For example, Struts and Spring are both frameworks for developing Model-View-Controller web applications. Both speed the development of MVC applications by creating code for items that are commonly needed, such as controllers and data sources.

It should be obvious by now that something needs to be done if we want web services to become ubiquitous. There must be some way to simplify and automate these difficult and receptive tasks. The most obvious solution would be a framework for developing web services. A good candidate for a web services framework is the Spring Framework (in cooperation with a selected other technologies). Although Spring was not designed from the ground up to be a Web Services framework, many of its concepts and ideas can be applied to a Web Services framework.

## Do We Need a Framework?

Do we need a framework? If we truly believe that web services are a trivial application to develop, then no we do not. Detractors of the development of a framework would argue that a

web service is like any other Java program, and most Java programs are not developed using frameworks. They would say there is nothing considerably harder about creating a web service than a Swing application or a web application. There is no pressing need for a framework, nor does it significantly decrease development time. Developers will still be able to develop web services without a framework.

To support their argument detractors could point to the many applications that have been successfully developed entirely using just a text editor, such as vi or emacs and the core Java API. They can also point out that developers of applications for .NET, Perl, ASP, C, etc. often do not use frameworks either and these languages are often used for web services.

While it is true that one can develop web services without a framework, it may not be the best way. Even the developers of Swing applications and web applications realized this. Frameworks have been developed to assist in their development. Web applications have frameworks such as Struts, Maverick, and Spring to assist in their development and Swing has its own frameworks including Java GUI Builder and SwiX$^{ML}$ that allow the developer to use XML descriptors rather than writing complex Swing code.[14,15] If even these "simple" applications (as opposed to a web service which has to interpolate with various systems) can benefit from a framework, then surely there are areas in which a web services framework would benefit web services development. In addition, there is talk of porting the Spring Framework to .NET in order to enhance the development of web applications using .NET.

In addition, frameworks provide benefits above and beyond the generation of code. They allow the most efficient use of a team. A framework such as Struts or Spring allows for clear separation of tasks. In both applications, once the model of the data is designed, HTML developers can work independently of the Java developers. With support for DAOs in Spring, business logic developers can implement their business logic independent of how far along the SQL developers are. In traditional development, such as the one servlet approach, developers' efficiency is hindered. One cannot work on the HTML portion while another developer is implementing the database code.

Therefore, we believe that the development of a framework is inevitable. For this paper, we will focus on improving web service development in the areas of (a) configuration of the web service, (b) database connections, (c) transactions, (d) procedure calls, and (e) Logging, auditing, etc. These are not all the reasons to choose a web service, but they are often the easiest to measure. Efficiency of the team and shorter timelines are much more difficult to measure unless a project is developed twice (once with the framework and once without with comparable teams).

## Configuration of the Web Services

Whenever developers write applications, they often have programs that are configurable either through command-line arguments, properties files or hard-coded constants. However, this hodgepodge of information scattered across multiple files makes it difficult for developers to find their configuration information, introduce new information, and maintain existing configurations. One must remember which configuration file does what, which classes use what file and where the files are stored. This does not even take into account command-line parameters or deployment files such as those for Apache SOAP.

One of the first things a web services framework needs is a simplification of configuration (preferably one file but also efficiently handling more than one). The most effective use of a configuration file is the use of a concept called Inversion of Control (IoC). Today, it goes by the name of Dependency Injection pattern. The basic idea is to have a separate object that populates the fields of a class with the appropriate implementation. There are three types of dependency injection: construction injection, setter injection and interface injection.[1] Many lightweight containers that exist use IoC, including PicoContainer and Spring.[13] The Spring Framework offers a perfect example of a no-hassle way of configuring an application. First, all classes are treated as JavaBeans (though you can also use construction injection). For developers, this just means exposing their configuration options as setters and getters. The configuration file is read at bootstrap and the options are "wired" to the objects.[3] Here is a simple sample configuration file and an accompanying JavaBean:

```
<bean id="exampleBean" class="examples.ExampleBean">
    <property name="beanOne"><ref bean="anotherExampleBean"/></property>
    <property name="beanTwo"><ref bean="yetAnotherBean"/></property>
    <property name="integerProperty">1</property>
</bean>

<bean id="anotherExampleBean" class="examples.AnotherBean"/>
<bean id="yetAnotherBean" class="examples.YetAnotherBean"/>

public class ExampleBean {

    private AnotherBean beanOne;
    private YetAnotherBean beanTwo;
    private int i;

    public void setBeanOne(AnotherBean beanOne) {
        this.beanOne = beanOne;
    }

    public void setBeanTwo(YetAnotherBean beanTwo) {
        this.beanTwo = beanTwo;
    }

    public void setIntegerProperty(int i) {
        this.i = i;
    }
}
```

The above example shows many benefits. First, we can see that if anotherExampleBean and yetAnotherBean are threadsafe, then we only need a single instance of that bean which can be shared with any Bean that needs it (wired through the configuration file). We can also see that there is no longer any need for set up/configuration code within the ExampleBean. The developer can assume it is passed to the bean and focus on business logic. The configuration will handle all of the set up.[3] This separation will also allow the substitution of different configurations easily for different deployments and will facilitate easier maintenance and testing..

Many other patterns also exist for configuring applications, including the Service Locator pattern. The Service Locator basically means that there is some object that knows how to get a hold of the services that an application needs. Whichever pattern is chosen is irrelevant to the

major idea: the specific pattern used is not as important as actually separating configuration from use. Any Web Services framework should reduce configuration to one file in addition to removing all configuration information from the actual objects themselves.

## Database Connections

One of the aspects of application development that programmers dread is writing code that connects and reads or updates from the database. It is often described as complicated, cumbersome, and tedious.

In general, connecting to a database is a complicated process. One must write code to obtain an instance of the driver for the database, construct a convoluted URL to connect to the database, and wrap the database calls around a plethora of try/catch/finally blocks. All this is done before you even try and execute any SQL statements. To actually obtain a result from the database, you must create a statement, feed it input and execute it. Then to get any results back, a ResultSet must be cycled through.

Not only is that complicated, it's also cumbersome. There is often an exorbitant amount of code to do so little. To read one result from a database, one must create a connection, create a statement, execute the statement, read the result, and close everything. Now, imagine doing that for every SQL statement. With try/catch blocks and connection code everywhere, it becomes very difficult to maintain the program. It will be almost impossible to easily read a class file.

It is also tedious because of the fact that all code to execute SQL statements is almost the exact same thing with minor tweaks. You will end up copying huge amounts of code that do almost the same thing with minor tweaks. Most developers find this task boring. A bored developer is more prone to introducing bugs as one copies code but forgets to change certain items (such as field names). Debugging that code will be a nightmare. It looks like we need an easy and efficient way to write JDBC code.

Finally, a proper framework should attempt to fix the deficiencies of the JDBC specification. JDBC requires you to catch a generic SQLException. In most cases, however, there is nothing you can do with the SQLException as it represents a major problem. In such cases it is better for the SQLException to just propagate (using "throws SQLException" on the method signatures is not necessarily the best way to do that).

The naïve approach would be for a developer to create a simple wrapper function that handles all of the connections and try/catch blocks. In practice, for small programs this will work well. But for large production applications, a function like this will not be efficient enough. In general it will have to return a ResultSet to be generic enough. This however, will do things like leave connections and ResultSets open for too long. In addition, passing ResultSets around is often a recipe for disaster. ResultSets must be closed when finished just like statements and connections. However, this requires the developer to manually remember to close it outside of the wrapper function.

In general it is not good to open a connection to a database every time a query needs to be run against the database. Most production applications use connection pools via DataSources. The

wrapper function does not work particularly well with DataSources (though it can most likely be retrofitted to work).

The Spring Framework makes working with JDBC extremely easy. First, it has native support for the Data Access Object pattern which encapsulates and separates most of your database code from the rest of your application. Second, it converts all checked SQL Exceptions into unchecked exceptions. Unchecked exceptions do not need to be caught and are automatically treated as fatal unless you *explicitly* desire to handle the error. To help with error handling, it converts exceptions to common problems including DataIntegrityViolationException and TypeMismatchDataAccessException among others. Converting exceptions allows you to handle certain problems that you can recover from while allowing other ones to be fatal. Spring also has abstractions for items like SqlQuery and SqlUpdate. SqlQueries are helpful in that instead of creating all of the code to do a loop to get the data back, one creates a class that extends SqlQuery and implements the method mapRow. MapRow is used so that a developer can map an individual row to an Object. Spring then handles turning that into a List so the developer does not need to write tedious, error-prone looping code.[3]

Furthermore, Spring makes connecting to the database effortless. A DataSource is defined in an XML file and passed to the configuration file. Spring then creates a DataSource. A DataSource is used by the developer and given to any DAO object that needs one. Spring then handles obtaining, opening and closing connections when results are needed from a database. The most effort a developer must make is calling getDataSource()[3].

We will now illustrate the ease of use of Spring's JDBC abstraction using an example from a recent Spring presentation.[17] The example assumes we have the following class:

```
package org.buggybean.phillyjug;
import java.math.BigDecimal;

public class Beer {
          private long id;
          private String brand;
          private BigDecimal price;

          public long getId() {return id;}
          public void setId(long id) {this.id = id;}
          public String getBrand() {return brand;}
          public void setBrand(String brand) {this.brand = brand;}
          public BigDecimal getPrice() {return price;}
          public void setPrice(BigDecimal price) {this.price = price;}
}
```

The code to retrieve a beer from the database using traditional JDBC code is as follows:

```
public Beer getBeer(long id) {
          Beer beer = null;
          Connection conn = null;
          PreparedStatement ps = null;
          ResultSet rs = null;

          try {
          Class.forName("org.hsqldb.jdbcDriver");
          conn = DriverManager.getConnection("jdbc:hsqldb:hsql://localhost", "sa", "");
          ps = conn.prepareStatement("select id, brand, price from Beer where id = ?");
```

```
          ps.setLong(1, id);
          rs = ps.executeQuery();
          if (rs.next()) {
                    beer = new Beer();
                    beer.setId(rs.getLong("id"));
                    beer.setBrand(rs.getString("brand"));
                    beer.setPrice(rs.getBigDecimal("price"));
                    }
          }
          catch (ClassNotFoundException e) {
                    logger.error(e.toString());
          }
          catch (SQLException se) {
                    logger.error(se.toString());
          }
          finally {
                    if (rs != null) {
                              try {rs.close();}
                              catch (SQLException ignore){}
                    }
                    if (ps != null) {
                              try {ps.close();}
                              catch (SQLException ignore) {}
                    }
                    if (conn != null) {
                              try {conn.close();}
                              catch (SQLException ignore) {}
                    }
          }
          return beer;
}
```

As we can see, there is a lot of extraneous code that is required by JDBC. This makes the code more difficult to read and manage. Now, see the same example in Spring:

```
private class BeerMappingQUery extends MappingSqlQuery {
          public BeerMappingQuery(DataSource ds) {
                    super(ds, "Select id, brand, price from Beer where id = ?");
                    super.declareParameter(new SqlParameter("id", Types.INTEGER));
                    compile();
          }

          public Object mapRow(ResultSet rs, int rowNumber) throws SQLException {
                    Beer beer = new Beer();
                    beer.setId(rs.getLong("id"));
                    beer.setBrand(rs.getString("brand"));
                    beer.setPrice(rs.getBigDecimal("price"));
                    return beer;
          }
}

public Beer getBeer(long id) {
          BeerMappingQuery beerQuery = new BeerMappingQuery(dataSource());
          return (Beer) beerQuery.findOBject(id);
}
```

Spring's implementation takes up considerably less code and is easier to read, as it removes the JDBC code but retains the logic of what is happening. It is easier to see beerQuery.findObject(id) than it is to see Connection conn, etc. It should also be obvious that it takes less user written code to accomplish something in Spring than it does in JDBC.[17] The table on the right summarizes a few of the the major differences between JDBC and Spring. Transactions will be discussed later. The row for Testing discusses creating unit tests. In general, if your application uses JNDI for connection pools, it is more difficult (but not

impossible) to test your applications. However, Spring provides a simple way of configuring a DataSource to handle your testing.

**JDBC / Spring comparison**

| | JDBC | Spring |
|---|---|---|
| Connections | Need to explicitly open and close connections. Need a separate strategy for making code reusable in a variety of environments. | Uses a DataSource with the framework managing connections. Code following the framework strategy is automatically reusable. |
| Exceptions | Must catch SQLExceptions and interpret database specific SQL error code or SQL state code. | Framework translates exceptions to a common hierarchy based on configurable translation mappings. |
| Testing | Hard to test standalone if code uses JNDI lookup for connection pools. | Can be tested standalone since a DataSource is easily configurable for a variety of environments. |
| Transactions | Programmatic transaction management is possible but makes code less reusable in systems with varying transaction requirements. CMT is available for EJBs. | Programmatic or declarative transaction management is possible. Declarative transaction management works with single data source or JTA without any code changes. |

Though Spring seems to simplify the extraction of data from a database, it still requires a user to write possibly complicated SQL code. Not every developer has the knowledge or ability to write complex join statements. In order to make the creation of web services easier, there should be an abstraction to possibly eliminate the requirement of writing SQL code. Since most developers understand the concepts (and often use them) of Object Oriented Programming, it would be best if the database could somehow be mapped to the domain objects without explicit SQL code written.

A technology, called Hibernate, exists that allows a developer to retrieve and update data in a database without writing a single piece of SQL code. Hibernate describes itself as an "object/relational persistence and query service for Java." In the simplest sense, Hibernate requires you to write a configuration file (usually XML) that maps the objects parameters to the tables and fields in the database. With this mapping, developers write queries in the Hibernate Query Language. At first it may seem weird to have to learn another language, but HQL simplifies many of the SQL paradigms. In addition, by using Hibernate, updates are done by calling a simple save function with passing the object.[2] Finally, Spring has excellent built-in Hibernate support that makes the using Hibernate even easier.[3]

**Spring JDBC Division of Labor**

| Task | Spring | You |
|---|---|---|
| Connection Management | √ | |
| SQL | | √ |
| Statement Management | √ | |
| ResultSet Management | √ | |
| Row Data Retrieval | | √ |
| Parameter Declaration | | √ |
| Parameter Setting | √ | |
| Transaction Management | √ | |

Though use of Hibernate may be considered optional, the benefits of a framework like Spring's JDBC sub-component should not be overlooked. Spring will make development of database-driven applications easier by offering a "JDBC abstraction layer that offers a meaningful exception hierarchy (no more pulling vendor codes out of SQLException), simplifies error handling, and greatly reduces the amount of code you'll need to write. You'll never need to write another finally block to use JDBC again."[3]

## Transaction Management

Another painful aspect of developing web services (and applications in general) is transaction management. Sun has developed a Java Transaction API (JTA). To demarcate a JTA transaction, a developer must invoke the begin, commit and rollback methods. The begin and commit calls denote the update to the database. Rollback is called if the update fails. JTA has some powerful features. It can span updates to multiple databases and is independent of the transaction manager implementation.[4,5] However, the transaction is controlled by the J2EE transaction manager, which forces the developer to use J2EE which adds additional complexity

with minimal benefit. In addition, this again forces us to use try/catch blocks and redundant JDBC code. In order to make transactions a viable option, our framework would need to handle them with minimal effort on the developer's part.

The simplest way to accomplish this is have the developer write no transaction code at all. In theory, we should be able to just say, "hey, this method needs to be a transaction" and leave it at that. We should not be required to explicitly state where a transaction begins and ends, and when to roll it back.

An interesting option of accomplishing this is to demarcate the method itself as a transaction (i.e. everything within the method must complete or the transaction is a failure and must be rolled back). This can be accomplished using the idea of Aspect Oriented Programming, or AOP. The idea of AOP allows the developer to focus on his main concern (such as the updates that need to occur) while the extraneous concerns (called crosscutting concerns) are kept separate and can be easily re-used. A perfect example of a crosscutting concern would be marking a transaction.[6] Spring itself provides excellent AOP Alliance support. One of the things AOP is used for in Spring is to provide *declarative transaction management*. In Spring, to mark something as requiring a transaction, all that needs to be done is modify the configuration file and point a TransactionProxy at the class and method that requires the transaction. Spring will then handle the creation of a transaction advisor and handle the transaction for us.[5] An interesting side-effect of AOP is that if we at some point decided to disable the transactions, we only need to comment out the configuration information. No changes to any code would be required. The following snippet of configuration code shows how easy it is to add transaction management:

```
<bean id="petStore"
    class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">
    <property name="transactionManager"><ref bean="transactionManager"/></property>
    <property name="target"><ref local="petStoreTarget"/></property>
    <property name="transactionAttributes">
        <props>
            <prop key="insert*">PROPAGATION_REQUIRED</prop>
            <prop key="update*">PROPAGATION_REQUIRED</prop>
            <prop key="*">PROPAGATION_REQUIRED,readOnly</prop>
        </props>
    </property>
</bean>
```

From this code we can see that we are saying all insert and updates of the Pet Store have PROPOGATION_REQUIRED. Spring supports Regular Expressions for matching method names (so insert* applies to all methods starting with insert). This also demonstrates how easy it is to remove transactions (just comment out <prop key…..></prop>).

In general, Spring makes it easy to work with Transactions by providing a "generic abstraction layer for transaction management, allowing for pluggable transaction managers, and making it easy to demarcate transactions without dealing with low-level issues. Generic strategies for JTA … are included. In contrast to plain JTA or EJB CMT, Spring's transaction support is not tied to J2EE environments."[3] Spring gives us the benefit of transactions that are normally only available to J2EE without having the additional complexity of actually using J2EE.

## Remote Procedure Calls

One of the benefits of remote procedure calls that have never been fully realized is the seamlessness of going from a local procedure call to a remote procedure call. One such attempt, Java RMI, is not very transparent. For the developer of the web service, he or she must have every class implement a Remote interface. Then, every method must throw a RemoteException. Further complicating the process is compiling Adding another level of complexity, classes must also be compiled into stubs in addition to the normal compilation. It seems that for the developer, RPC is not very transparent. For the client, it is a slightly nicer picture. Other than having to do a look up and catching RemoteExceptions, the client simply calls obj.method.[8]

Using SOAP however, complicates the process immensely. On the developer side, there are minimal changes. A developer writes his or her code as usual. In general, he or she then chooses a SOAP implementation (such as Axis) and then describes how to deploy the service. This however, places a lot of pressure on the user of the service. Current SOAP implementations do not accurately replicate the experience of just calling a method. In many cases, the parameters are placed in a vector, and passed along with a string representation of the parameter name to the service. Further complicating it, we must often explicitly define the serialization of objects (in the deployment descriptors and in the method calls themselves). All of this can be avoided by using an API that is developer provided, but it also requires the developer to develop a unique API for each service, an often tedious and repetitive task.

Obviously what needs to be done is to combine the client ease of use of the Java RMI with the simplification for the developer. Again, we can use Spring as a model for this. Spring provides support for a number of remote protocols such as RMI, JAX-RPC (via Apache Axis implementation), Hessian and Burlap from Caucho.[10] To illustrate the ease of developing web services and exposing them we will be using Hessian as our example.

To expose a business object as a remote service, a HessianServiceExporter is used. The exporter is implemented as a controller and it translates requests into service invocations, entirely independent from the client. The URL to the service is determined by a generic handler mapping (i.e. /example/myservice denotes an entire service rather than a specific method).[10] If a web service client was developed using Spring, the following configuration code would give access to the object (and from within the `beanNeedingAccessToEmployeeService,` the fact that they were remote calls would be transparent.):

```
<!-- Proxy for the Hessian-exported EmployeeService -->
<bean id="hessianProxy"
        class="org.springframework.remoting.caucho.HessianProxyFactoryBean">

        <property name="serviceInterface">
                <value>edu.rutgers.acs.trs.service.EmployeeService</value>
        </property>
        <property name="serviceUrl">
                <value>
                        http://rutgers.edu:8080/trs/caucho/EmployeeService-hessian
                </value>
        </property>
</bean>

<bean id="beanNeedingAccessToEmployeeService" class="edu.rutgers.acs.gas.TrsGateway">
```

```
        <property name="employeeService"><ref bean="hessianProxy"/></property>
</bean>
```

On the client side (without Spring), the user just defines a HessianProxyFactoryBean and specifies a service interface and URL. The proxy implements the service interface and delegates each method call to a remote service. In effect, this allows the client-side to be ignorant it is working with remote interfaces as the interface does not need to extending something like java.rmi.Remote or throw RemoteExceptions.[10] This theory can be applied to Hessian as well as other protocols such as RMI. The below snippet of code from the Hessian web site[9] demonstrates how effortless it is:

```
String url = “http://www.caucho.com/burlap/test/basic”;
HessianProxyFactory factory = new HessianProxyFactory();
BasicAPI basic = (BasicAPI) factory.create(BasicAPI.class, url);

System.out.println(“hello(): “ + basic.hello());
```

One of the other benefits of using Spring as your web service is that because your business object do not implement any protocols, you can expose it via multiple protocols by defining more exporters in your configuration file. The client-side business object can easily be configured for any remote service accessor.[10] Though we have given an example using Hessian, one can see that the same could apply for Axis.

As we can see with minimal effort on both the client and developer side we can have an excellent abstraction of remote method invocation.

## Logging, auditing, etc.

Often, a developer needs to implement both auditing and logging into an application for various purposes including normal performance monitoring, debugging, and security. Sun provides a very basic Logger API (and implementation)[11] and groups like Apache provide even better implementations of Logging (i.e. Log4j)[12].

Often logging statements are placed haphazardly in code to assist in debugging. This often decreases the readability of code (though Apache argues otherwise but does not provide support[12]) and can slow down an application when not in use. Apache has tweaked their Log4j so that the cost of determining if a logging message needs to be sent is five (5) nanoseconds on a AMD Duron 800 Mhz machine[12]. However, it is still better and cleaner to be able to add and remove the logging information at will.

Similarly, auditing information often needs to be placed in an application. Generally, auditing information will be required for logins, access to secure sections and the updating of data in a database. Often though, we do not want to pollute our business logic with auditing logic. What if we could still have the auditing (or logging) done but outside of the method?

Spring and AOP again come to our rescue. Similar to our transaction managing example, it would be nice if our framework could somehow provide pluggable code to allow execution of logging or auditing to be done before or after method invocation.

For example, the following AOP Interceptor would provide much of the same debugging functionality as provided in a normal Logger debugging, but without the needed complexity of adding Logger.write statements:

```
public class DebugInterceptor implements MethodInterceptor {

    public Object invoke(MethodInvocation invocation) throws Throwable {
        System.out.println("Before: invocation=[" + invocation + "]");
        Object rval = invocation.proceed();
        System.out.println("Invocation returned");
        return rval;
    }
}
```

This can be wired in to the application, similar to the configuration information for the transaction manager. It can just as easily be commented out, meaning there would be no performance loss in checking whether it needs to be logged or not.

We can set up a similar system for auditing. Auditing often needs to be done before (and/or after) a method invocation but its often messy to put it along-side business logic. An interceptor similar to the debug can extract the required objects and do any auditing required.

While one can argue that no framework needs this because it does not necessarily make logging and auditing easier, it does make the code cleaner and easier to read which in turn can allow a developer to write less buggy code. It also allows you to turn logging or auditing on or off without touching the binaries and without incurring a performance penalty cost because the code is still within the binaries.

## Benchmarks and Testing

Above, we have discussed various problems and their solutions offered by the Spring Framework. However, at first glance it would appear that these abstractions add signification overhead to the running of the application. However, this is not always the case.

Benchmarks of the Spring Framework indicate that, “the performance of Spring declarative transaction management exceeds that of EJB CMT.”[17] Therefore, not only is it easier to handle transactions in Spring, it is also faster! Within the Spring Framework CVS Tree, there is a benchmark branch that allows a developer to test the Spring implementation of a project against the J2EE implementation. According to a recent presentation by Rod Johnson (father of Spring Framework), in all cases the Spring implementation was faster than the J2EE implementation.[10]

In the specific area of JDBC, tests indicate that there is no performance loss with using the Spring Framework JDBC wrapper versus straight JDBC code.[10] In fact, it would not be hard to speculate that with Spring’s advanced connection and resource management, it may be faster to use their wrapper.

An additional general point is that Spring often uses reflection to simplify the code the user has to write. Rod Johnson, in his book, “expert one-on-one J2EE Design and Development”, argues

that the overhead of reflection is negligable in recent implementations compared to the cost of the business logic itself. The user will see no noticable difference in the performance of the application.[18] Similarly, the use of AOP Intercepters has not introduced any significant overhead to the excution of methods.[10]

Finally, no matter how useful or fast a framework is, if it is buggy, it is useless. Spring Framework has been more rigorously tested than most frameworks or production applications. They have completed unit testing on almost 99% of the framework. They also have rules in place that will not allow a developer to commit new functionality unless it is also committed with the unit complete unit testing.[10] All of this testing allows the developer to be confident the infrastructure he is using can be trusted. Therefore, he can focus on implenting and debugging his business logic with the confidence that his framework will work as advertised.

## Conclusion

There is a reason that an established developer often has his own toolbox of utilities and functions that he uses whenever he develops an application. He recognizes that there are always ways of improving the way something is done and he often writes his own functions and classes to accomplish that and carries them around with him wherever he or she goes.

It should be readily apparent to any developer that there are currently many deficiencies in the way that we write web services. Many of those problems have been outlined above, but that is not all of them. What developers of web services need is their own toolbox to make their lives easier. We believe we have shown that there is sufficient need for a web services framework to be included in that toolbox. We have seen how easy the Spring Framework makes tackling particular tasks. Spring is even robust enough to be used as a web service framework. Many companies including Synapsis Technology, Inc. (for its EMARS program) and the Administrative Computing Services division of Rutgers, the State University of New Jersey (for its Graduate Admissions System and HR Time Reporting System)[17] have implemented production mission-critical applications using Spring. According to Bruce Tate, “Lightweight containers like Spring are spreading rapidly, because they can solve problems that other containers can’t…The most interesting aspects, to me, are the transparency of the objects, clean and pluggable services and the light footprint.”[17]

Spring has the potential to do for Web Services what it did for MVC applications. If not Spring, then another framework will come along. But we can be sure that it will not be long before there will be a developer who attempts to correct the inefficiencies in creating web services. And his solution will most likely look similar to the combination of Spring, Hibernate and AOP.

## References


1. Fowler, Martin. "Inversion of Control Containers and the Dependency Injection pattern." http://www.martinfowler.com/articles/injection.html. 23 Jan 2004.

2. "Hibernate - Object/Relational Mapping and Transparent Object Persistence for Java and SQL Databases." http://www.hibernate.org.

3. "Spring - java/j2ee Application Framework". http://www.springframework.org.

4. "Java Transaction API (JTA)". http://java.sun.com/products/jta/.

5. "Bean-Managed Transactions". http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Transaction4.html#63068.

6. "AOP Alliance." http://aopalliance.sourceforge.net/

7. "Chapter 5. Spring AOP: Aspect Oriented Programming with Spring". http://www.springframework.org/docs/reference/aop.html

8. Wollrath, Ann. "Getting Started Using RMI." http://java.sun.com/j2se/1.4.2/docs/guide/rmi/getstart.doc.html

9. "Caucho Resin : Fast, Developer-Source Application Server." www.caucho.com.

10. Conversations with Dmitriy Kopylenko <dkopylen@rutgers.edu> (Contributor to Spring Framework)

11. "Logger (Java 2 Platform SE v1.4.2)." http://java.sun.com/j2se/1.4.2/docs/api/java/util/logging/Logger.html

12. Gulcu, Ceki. "Log4j project – Introduction." http://logging.apache.org/log4j/docs/

13. "PicoContainer." http://www.picocontainer.org/

14. "SWIXML - Generate javax.swing at runtime based on XML descriptors." http://www.swixml.org/

15. "Java Gui Builder Project Home." http://jgb.sourceforge.net/index.php.

16. "BEA Tuxedo Glossary." http://edocs.bea.com/wle/tuxedo/glossary/glossary.htm

17. "DAO and JDBC support." http://www.springframework.org/presentations/phillyjug/spring-phillyjug.html

18. Johnson, Rod. *expert one-on-one: J2EE Design and Development.*