

Analyzing Spring Framework (revised August 2005)

Sergio Blanco Cuaresma, Computing Engineering student, *URV*

Abstract—This document review the framework concept, trying to build a classification based in the most important features that a framework can present. Additionally, a concrete framework case called Spring is analyzed, giving a complete overview of the main advantages. Spring is a full-stack Java/J2EE application framework, which delivers significant benefits for many projects, reducing development effort and costs while improving test coverage and quality.

Index Terms—Design Pattern, Framework, J2EE, AOP.

• *S.B.C. Author is with the Computing Engineering studies, Universitat Rovira i Virgili, Tarragona, Spain. E-mail: sergio.blanco@estudiants.urv.es.*

Manuscript received 30/08/2005.

1 INTRODUCTION

We can consider a framework as an object oriented tool for improve reusability. A good definition would be a reusable system design, which describes how this system is decomposed in several inter-related objects. Sometimes the system corresponds to a complete applications, but it can also be just a subsystem of a more complex software.

A framework describes the behavior and relationship of the objects with a detailed interface and work flow. System functions and responsibilities are mapped onto this objects, providing an specialized architectures for a given application domain.

The most important part of a framework is how is divided each system component. A framework can reuse implementations, but it is less significant than knowing how are structured its functions.

Normally, a framework is implemented with an object oriented language, each object is described by an abstract class (class that cannot be instantiated, with empty and implemented methods). These abstract classes can be used as templates for object creation and they are used for component designs inside the framework.

Modern object oriented languages like Java or Corba, treat separately the interface from classes. With that languages, a framework is described in interface terms but with an important inconvenient. This kind of languages usually only support inheritance from one class and a framework is also a collaborative interactive object template model, so it is usual to combine abstract classes with simple interfaces in order to define a component.

Frameworks have a lot of resemblance with object oriented programming:

- **Data Abstraction:** Internal data is not accessed directly, instead, a group of public interfaces and signatures are defined.
- **Polymorphisms:** Ability to adopt different types or roles.
- **Inheritance:** Creation of new components/classes with

similar functionality.

The main benefits from working with object oriented frameworks are:

- **Modularity:** frameworks increment modularity by encapsulating implementation details behind stable interfaces.
- **Reusability:** interfaces with generic components improve reusability because they can be applied to create new applications. Productivity, quality and interoperability get also improved.
- **Extensibility:** applications can extend interfaces in order to obtain new concrete services, adapted to each one needs.
- **Inversion of control:** It allows the framework to take the control, instead of the traditional control managed by the main application.
- **Risks minimizations:** costs are drastically reduced and end product quality is improved.

In order to choose and use a framework, it is important to analyze if it would fit in our concrete problem, and understanding of its architecture, identifying hot-spots is a must in order to adapt or extend the framework in a good way.

It is important to remark that frameworks can be object oriented or component oriented. Frameworks based in components present especial features:

- Late composition
- Dynamic interconnection
- Extensibility
- Black box
- ...

In general, a framework offers one of the possible techniques for reusing code. Ideally it must be completely transparent for the developer, providing easily assembled components without the need of knowing its internal implementation. Resulting system

would be efficient, easy to maintain and stable.

Design patterns are complementary to frameworks, they are fine-grain because they offer an abstract solution for a recurrent problem. Design patterns' advantages are:

- Reusability of solutions to common problems
- Already tested solutions
- Flexible, allowing adaptations to specific problems

But they also present some disadvantages:

- Big quantity without a clear classification
- Complexity
- Bad defined composition

The original vision of software reusability was based in components. The ideal situation would be a component accomplishing the exact requirements, without parametrization or specialization. But, in real life, several needs must be covert and adaptations are required.

The more configurable a component is, easier is the adaption job to concrete situations but more complex is learning. And a framework has a big adaptability, so that makes it powerful useful for many applications, but not so easy learning.

Our concrete case of study is Spring, a Java/J2EE framework that offers (among others):

- Light container
- Aspect oriented programming support
- JDBC abstractions
- Source code level metadata
- Web development facilities with MVC

Other interesting aspect offer declarative transactions with JTA to plain Java classes (POJO: Plain Old Java Objects). This allow us to implement transactions in light system based in, for example, Tomcat.

Spring also offers mechanism to make persistence object with just one line of code, using Hibernate templates. Simplifying persistent logic creation.

Inversion of control and different technologies integration are also present in Spring, giving maximum reusability to business objects and data access.

2 FRAMEWORK ANALYSIS

2.1 Classification

Classifications can be done following different arguments, frameworks can be classified in three different ways depending on its scope:

- System infrastructure frameworks: development of portable infrastructures like operating systems, communication frameworks and graphic interfaces are simplified.
- Middleware integration frameworks: commonly used to

integrate distributed components and applications. They are designed in order to increment modularity, reusability and extensibility.

- Enterprise application frameworks: Its main functionalities are related to business activities like telecommunications, economy, invoices, etc... Its costs (development time and money) is usually very high if we compare it with other kind of frameworks.

Spring can be situated in the "Middleware integration frameworks" group. It is oriented to infrastructure communications and distributed components.

Another important frameworks' characteristic is how they can be extended:

- White box
- Black box
- Gray box

Spring can be considered a gray box framework.

2.2 Services

Spring services reach from security to system management or work flow, and persistence is possible with JDO, Hibernate, TopLink, JDB or EJB.

Spring is built in a layer architecture which give a lot of flexibility, the main functionalities are situated in the first levels. The most important services offered by Spring (that can be used form any J2EE server) are:

- Inversion of control: configuration management based in JavaBeans, allowing easy parametrization. Bean factory which can be used in any environment (from J2EE containers to applets).
- JDBC abstraction layer: Exception hierarchy that gives facilities in order to avoid specific error codes and improve error control.
- Transaction management: Generic abstract layer with several interchangeable transactions managers, facilities to define transaction in a easy way. JTA strategies incorporated and unique JDBC DataSource.
- Framework Model-View-Controller: built in Spring's kernel. Allow us to create web forms similar to Struts with different view possibilities (JSPs is the main one).
- Aspect oriented programming support. Any object managed by Spring can be used with AOP.

2.3 Hotspots

Hotspots or hooks are clearly visible in Spring with its interaction with JavaBeans. The top abstract layer is the Bean Factory, which allow us to recover an object by its name.

Developers must define its own JavaBean that are going to be controlled by Spring. JavaBeans persistence is defined using

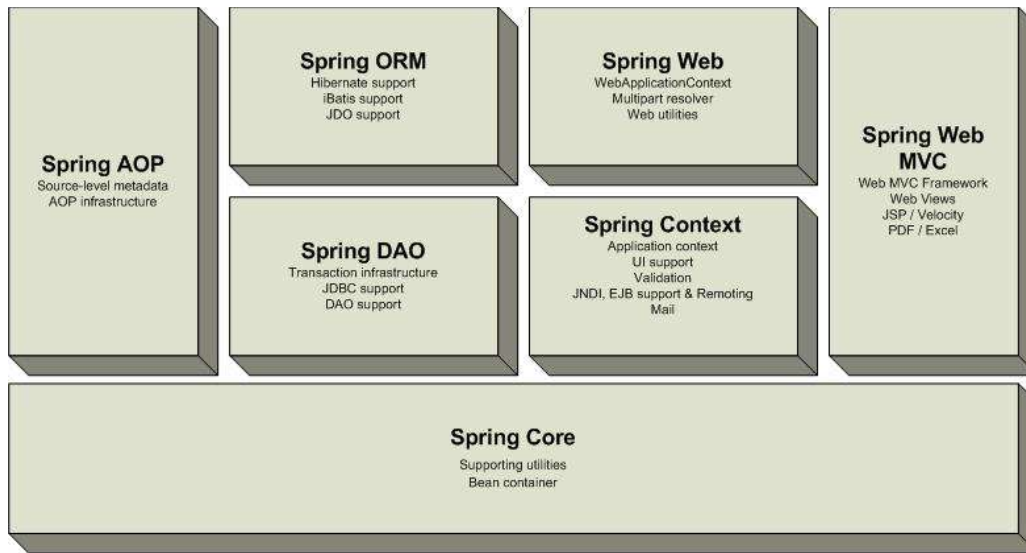


Fig.1 Spring modules

XML files similar to:

```
<beans>

  <bean id="myDataSource"
        class="org.apache.commons.dbcp.BasicDataSo
          urce"
        destroy-method="close">
    <property name="driverClassName" va
      lue="com.mysql.jdbc.Driver" />
    <property name="url" va
      lue="jdbc:mysql://localhost:3306/mydb" />
    <property name="username" value="someone" />
  </bean>

  <bean id="exampleDataAccessObject"
        class="example.ExampleDataAccessObject">
    <property name="dataSource"
      ref="myDataSource" />
  </bean>

  <bean id="exampleBusinessObject"
        class="example.ExampleBusinessObject">
    <property name="dataAccessObject"><ref
      bean="exampleDataAccessObject"/></property>
    <property name="exampleParam">
      <value>10</value></property>
  </bean>

</beans>
```

Additionally, Spring offers method for integration with other frameworks like:

- Struts: using ContextLoaderPlugin and Base classes.
- WebWork: using SpringObjectFactory.
- Tapestry.
- JSF: using DeletegrateVariableResolver or JSF-Spring Li-brary.

2.4 Architecture

Spring is composed of seven different modules as we can see in Fig.1.

Spring Core is the main framework part, it gives us the possi-bility to manage the bean container functionalities. The basic con-cept is the BeanFactory, which eliminates singletons needs and al-low us to separate configuration and dependence specification from the application.

Above Core is found the Context module, it offers an access way to Beans. It adds support for messaging, event propagation and transparent context creation (e.g. servlet container).

The abstract layer above JDBC is provided by the DAO mod-ule, it eliminates the need to treat specific error codes from data base. It also allows us to make declarative transactions.

The ORM module offers integration layers for object-relational APIs like JDO, Hibertante and iBatis. Using ORM it is possible to use this APIs in conjunction with other Spring features like transaction managements.

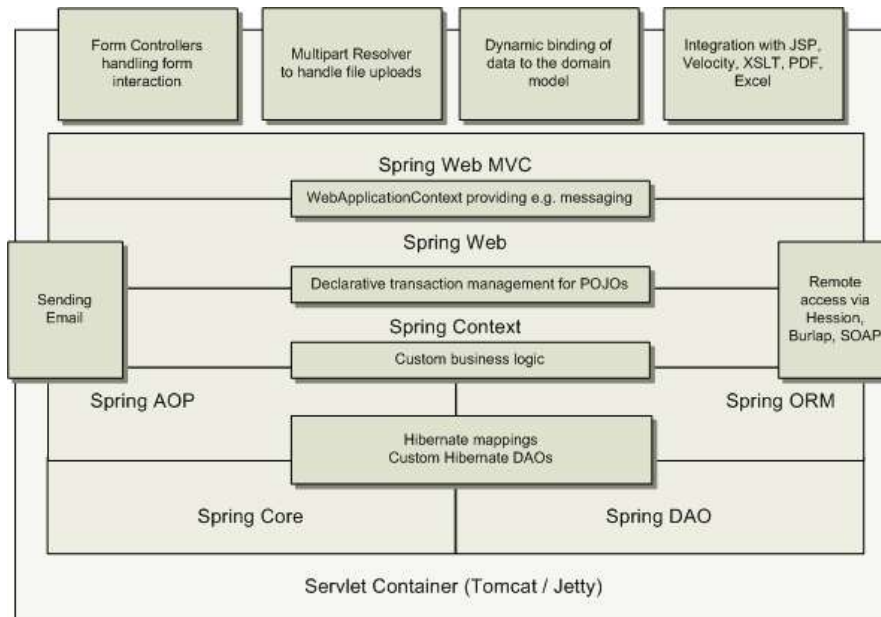


Figure 2. Spring services

Aspect oriented programming is possible thanks to AOP module. It is possible to define method interceptors and pointcuts in order to separate our code from cross-cutting characteristics. It permit us to easily develop security function, log systems, etc... Source code metadata is also offered by this module.

Spring web give us facilities in order to integrate our application in the web with functionalities like multipart, context initialization through servlet listeners and web oriented contexts. It is possible to combine Spring web with WebWork or Struts.

Finally, Spring Web MVC module offers an implementation of Model-View-Controller design pattern for web applications. It allows to separate model domain code from web forms. It facilitate using other Spring functionalities like validation.

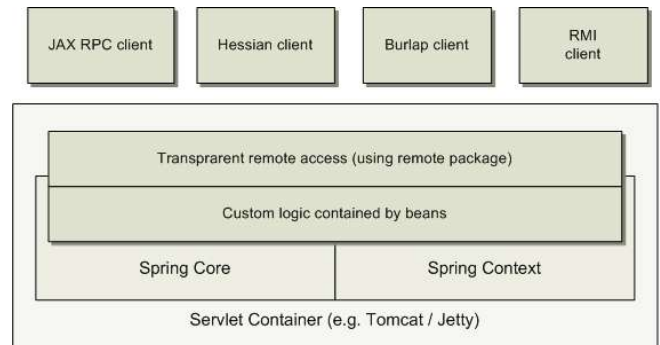
2.4.1 Use cases

Spring can be used in a lot of different scenarios, from applets to complex business applications.

For a typical web application we can use almost all the Spring functionalities. Business logic can be implemented using simple POJOs under the management of Spring Dependency Injection Container. Transactions would use TransactionProxyFactoryBean. Email an validation services are also available in Spring, etc... A complete schema can be seen in fig.2.

If we are not developing a new application and we want to migrate an old one to Spring framework, it is possible to do it progressively. Spring allows us to use just some parts of them in order to incrementally migrate our software to this framework. For example there are front-ends that use WebWork Struts, Tapestry that can be integrated in the middle layer of Spring in order to use its transaction features.

Web services and distributed object can be easily used with Spring Hessian, Burlap, RMOI or JaxRpcProxyFactory.



2.4.2 Design Patterns

It is possible to find several design patterns in Spring framework. The most visible one is the Model-View-Controller pattern, which is heavily used in the Spring MVC module.

In session beans, client code must implement a search for the request object in order to invoke a local or remote method. In this case, several patterns are used in order to avoid code repetition and optimize JNDI object localization:

- Service Locator: service in charge of finding resources ready to use.
- Business Delegate: In combination with other patterns, manage situations like object cache and repeated requests. If we have choose to use complex components like EJBs or JMS, it is possible to simplify its use in the presentation layer.

Finally, another important pattern is the factory one, used in

the Core module eliminating singletons needs and allowing us to separate configuration and dependence specification from the application.

2.4.3 Reflective architecture

A reflective system allows the programmer to create a new object meta-level and be substituted selectively by an existent part of a running system. This ability permits adding object to trace the program execution.

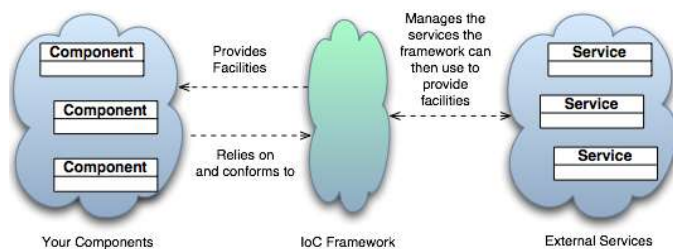
An object oriented reflective architecture makes easy debugging at language level. It adds elements with structural information used as basic interaction element between the different parts.

Spring framework offers reflective architecture based on middleware composed by a configurable component collection. We can get big advantages that make possible to change system elements easily (e.g. security related code or communication protocols).

2.4.4 Life cycle and Inversion of control

Inversion of control permit to simplify source code until the point that it is easier to understand and test. It is based in the Hollywood principle: "Do not call me, I will call you". IoC change the control and execution responsibilities, instead of traditional library calls, the framework is who call the application.

Spring is a container based in the inversion of control, more exactly Dependency Injection (IoC variant).



Dependency Injection eliminates explicit dependence with container APIs. The container predicts that a concrete component needs a concrete object and it provides it on runtime.

The main characteristics of Dependency Injection are:

- Setter Injection: via JavaBean setters.
- Constructor Injection: via constructor arguments.

Spring provides functions for both cases separately or combined, it can be specified in the object configuration.

Alternatively, Spring offers a group of callback events and an API for traditional lookup, but in general it is more recommended Dependency Injection.

The advantages of Spring Dependency Injection are:

- Components express its dependence with other components using JavaBeans set methods or constructor parameters. A JNDI search is the equivalent to this with EJBs.
- Testing is easier.
- Strong type is protected with a good implementation of IoC.
- Dependencies are explicit in the constructor or in the properties of JavaBeans.
- Business objects can be executed in different frameworks which offer Dependency Injection without modifying the code.

Bean factories support two object modes:

- Singleton: just one instance of an object is allowed with a concrete name. It would be located by lookup. Used for stateless objects.
- Prototype or non-singleton: each time an object of this kind is required, it is created with its own independent properties (opposite of singleton).

3 EVALUATION

3.2 Learning curve and development effort

Spring learning curve is moderated in comparison with other frameworks. And thanks to Inversion of Control, application testing is drastically reduced. The amount of needed code is significantly lower and much simpler.

Interaction points are well defined in the framework architecture, so it is possible to get a quick understanding of the system.

Once the developer has managed the learning phase and he knows all the Spring framework features, the development effort is very low.

Aspect oriented programming support is also a great help, it is an easy to understand concept and very useful idea for good software design.

Finally, developers can find several difficulties when they write code for J2EE platforms, but Spring can help this development providing standard application configuration, JDBC abstractions, integration with EJBs without affecting the global design, MVC architecture, etc...

3.3 Documentation

It is very usual to find framework with little documentation and with a non-standard format. Types of documentation that would be interesting to have for any framework:

- Graphic environments: it offers a deep information about

the framework.

- Reusing contracts: it defines cooperation between framework designers and people in charge of extensions and adaptations.
- Design patterns: Useful in order to design the architecture and make the design's documentation.
- Visual tools: visual representations of components, connectors, links, etc...

Spring website (<http://www.springframework.com>) offers a lot of standard documentation with formal semantic and oriented to different kind of users, it is possible to find detailed information about functionalities and architecture of each module, APIs, hotspots and most important developing features.

3.4 Abstractions

Spring offers great DAO abstractions for Hibernate, JDBC, SQLMaps among others. Declaratives transactions are supported, which permits a big facility for EJBs configuration. It simplifies construction and configuration of business objects thanks to Dependency Injection (IoC).

Although Spring has its own MVC framework, it is possible to use abstractions in order to use Struts, WebWork, Tapestry, etc...

Transaction abstractions allow us to have different transaction managers with interoperability capabilities. This transaction support is not linked exclusively with J2EE environments.

JDBC abstractions gives an exception hierarchy that gives facilities in order to avoid specific error codes and improve error control.

3.5 Extensibility

Extensibility is usually understood as facilities in order to expand functionalities. But it is also important to be able to contract functionalities and be extended outside the applications, so we have capacity to substitute pieces if we do not plan to use them. Consequently, we can have a smaller system which is more easy to maintain.

Spring is divided in well-defined separated layers, so it is possible to select which parts we are interested in.

Also, extern application can add new functions extending Spring framework features.

3.6 Validation and defect removal

Spring offers several possibilities in order to validate and debug applications making development easier.

Validation mechanisms are very similar to Struts. Spring permits configure validation logic in the same way it is done for any other business component.

Inversion of control is also a good technique to test and debug applications.

On the other hand, Spring does not force to inherit from a concrete class, so it is completely transparent and software can be tested outside the container.

Abstractions like the possibility to avoid writing code for direct JNDI services access, which normally is very complicated to debug, permits a fantastic improvement in the testing and debugging phase.

3.8 Efficiency

Spring framework biggest efficiency improvement is in the development part, more than in the execution part. Programmers can build application in less time and with better results.

Aspect oriented programing support, also allow to execute cross-cutting actions efficiently like validation, security, authentication, logs control, etc...

Execution costs improvements are not as important as development improvements, but they are situated in a middle position among other kind of J2EE frameworks.

3.9 AOP

One of the seven modules of Spring framework is exclusively targeted to Aspect oriented programming, so this is a very important part of the framework.

The first AOP support goal is provide J2EE services to POJOs. Spring AOP is portable between different application servers and it works in web containers and EJBs. It has been successfully used in WebLogic, Tomcat, Jboss, Resin, Jetty and Orion among others.

Spring AOP supports:

- Interception: it can intercept interfaces or classes invocations and execute code before or after. Fields cannot be intercepted.
- Introduction: an advice can implement additional interfaces to an object.
- Static and dynamic pointcuts: When an execution point is specified for being intercepted, it is possible to work with static pointcuts or dynamic pointcuts that allow access to method's parameters.
- Statefull and stateless interceptors.

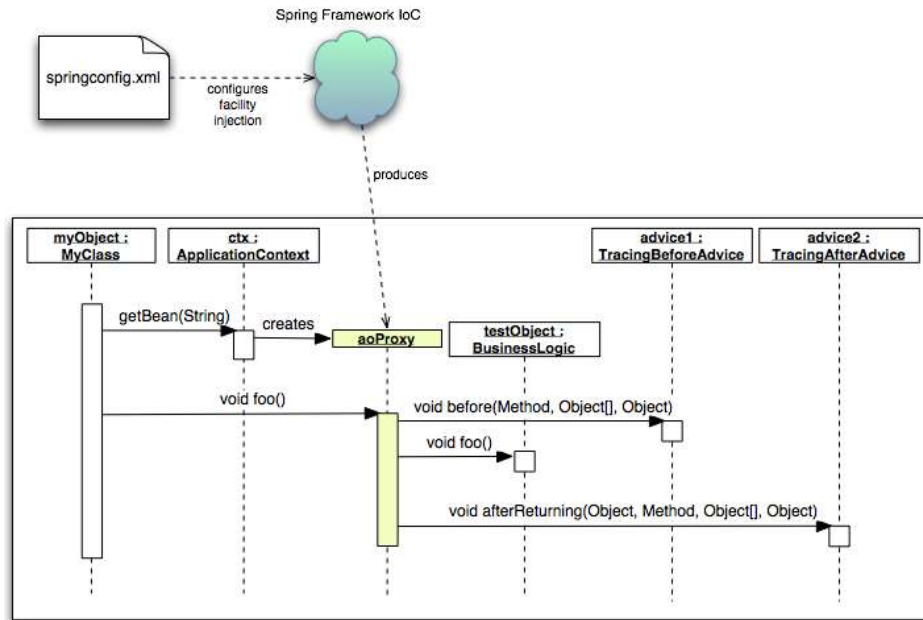


Fig.3 AOP

4 CONCLUSION

Spring is a J2EE framework that offers a light container with AOP support, JDBC abstractions, source code level metadata and MVC framework for web development. It consists in a development environment that foments inversion of control pattern and technologies integration.

The main benefits of using Spring framework are:

- Free Software: Spring is released under the Apache 2.0 License.
- Shorter life cycles: lots of already implemented tasks are built in the framework, so developers do not waste time reinventing the wheel. It has been calculated that saved time is around 40%.
- Lower learning curve: It is not necessary to use every Spring module, it is possible to just choose what the developer needs, so its learning curve is reduced.
- Reduced risk: Core technologies are reused, reducing reinventing the wheel risk.
- Consistent application architecture.
- Higher quality: Spring framework quality is a fact, resulting software is based in a tested framework with lots of features oriented to obtain good quality applications.

ACKNOWLEDGMENT

The authors wish to thank the Free Software community for making real software like Spring framework.

REFERENCES

- [1] Spring web documentation: <http://www.springframework.org/documentation>
- [2] Rod Johnson "Introduction to the Spring Framework", <http://www.theserverside.com/articles/article.tss?l=SpringFramework>
- [3] Rod Johnson, Juergen Hoeller, Alef Arendsen, Colin Sampleanu, Rob

Harrop, Thomas Risberg, Darren Davison, Dmitriy Kopylenko, Mark Pollack, Thierry Templier, Erwin Vervaeke, "Spring Java/J2EE application framework", <http://static.springframework.org/spring/docs/1.2.x/reference/index.html>

- [4] Spring API, <http://static.springframework.org/spring/docs/1.2.x/api/index.html>
- [5] Spring UML diagrams, <http://opensource.objectsbydesign.com/spring/>
- [6] Spring Tag listing, <http://static.springframework.org/spring/docs/1.2.x/taglib/index.html>
- [7] Thomas Risberg, "Developing a Spring Framework MVC application step-by-step", <http://www.springframework.org/docs/MVC-step-by-step/Spring-MVC-step-by-step.html>
- [8] Colin Sampleanu, "Better J2EEing with Spring", <http://www.springframework.org/node/140>

First A. Sergio Blanco Cuaresma 5 years as Computer engineering student at Universitat Rovira i Virgili.