J2EE VERSUS SPRING:
HEAVYWEIGHT VERSUS LIGHTWEIGHT

by
Ir. Jerónimo Benlloch Martí

Supervisors: prof. Dr. Ir. F. De turck, prof. Dr. Ir. B. Dhoedt,
Prof. Dr. Ir. P. Demeester
Coaches: Ir. Bruno Van den Bosche

Final year thesis submitted in partial fulfilment of the requirements for the degree of
Computer Science Engineer

Academic year 2006-2007
Permission to loan

“Permission is hereby granted to place this thesis at anyone's disposal and to copy this thesis completely or partially for personal use. Any other use is subject to copyright restrictions, in particular the liability to mention explicitly this thesis as source reference when referring to results of this thesis.”

Jerónimo Benlloch Martí, June 2007
J2EE VERSUS SPRING:
HEAVYWEIGHT VERSUS LIGHTWEIGHT

by Jerónimo Benlloch Martí

Final year thesis submitted in partial fulfillment of the requirements for the degree of Telecommunications Engineer Academic year 2006/2007

Supervisors: Prof. Dr. Ir. F. De Turck, Prof. Dr. Ir. B. Dhoedt, Prof. Dr. Ir. P. Demeester
Coaches: Ir. Bruno Van Den Bosche

Faculty of Engineering
Ghent University
Department of Information Technology
Chairman: Prof. Dr. Ir. P. LAGASSE

Abstract
Recent years Spring framework has achieved a lot of popularity among web developers. This thesis treats to evaluate whether Spring framework is an authentic alternative to J2EE, the most common framework for developing enterprise applications. In order to do that, comparing Spring and J2EE through learning of both technologies, the research of a J2EE application and latter development of this application under Spring. Finally, a performance evaluation of both applications is made.

Introduction
At the beginning of 21th century, J2EE was the most important platform for developing enterprise applications. Unfortunately, this platform doesn’t provide solutions to reuse code and neither make the application development easy for the developers, two problems that produce a rigid and costly implementation.

Study of J2EE and Spring framework
The strategy tacked to compare Spring Framework and J2EE platform consisted on researching the two technologies, their architectures and components to achieve a wide view
of the technologies. To obtain a knowledge closer to both technologies is necessary to study a more practical method.

A J2EE application was researched and with its structure and restrictions a Spring version was developed. The development of J2EE and Spring project will give us a real experience with these technologies and a more critical view.

The multitier architecture (figure 1), which structures it is the basis of development of the current enterprise applications. The architecture of Spring is similar to J2EE, besides uses the power of J2EE. The differences between both technologies consist on the components which take part of the handling of a request.

![Multitier Architecture](image)

**Figure 1: Multitier architecture**

With the experience acquired and the performance tests of the implemented applications, we will draw conclusions from code level together with performance and development throughput.

**Performance**

The results obtained from performing these Homesuite versions show that the two applications have equal efficiency, figure 2. However, it is verified that Spring application loads less traffic network, that is, it produces less transactions than Homesuite with the same simulated users per second because its application doesn’t always need to download all images from the websites.
Concerning development experience, the Spring features make application development easier than J2EE components. Those features turn out a code tidier and easier than J2EE plataform version. All of that entails that the required time to implement a Spring application is reduced and the throughput of web development increases with regard to J2EE application.
## Contents

### CHAPTER 1 INTRODUCTION AND OBJECTIVES

1.1 INTRODUCTION

### CHAPTER 2 STUDY OF J2EE AND SPRING FRAMEWORK

2.1 JAVA 2 ENTERPRISE EDITION

2.1.1 Enterprise JavaBeans

2.1.2 Java Servlet Technology

2.1.3 JavaServer Page Technology

2.2 SPRING FRAMEWORK

2.2.1 Core container – Inversion of Control

2.2.2 Context Container

2.2.3 Data Access Object Container (DAO)

2.2.4 Object/Relational Mapping Container

2.2.5 Aspect-Oriented Programming

2.2.6 Java Enterprise Edition Container

2.2.7 Model-And-View Framework

2.2.8 Web Container

### CHAPTER 3 CASE OF STUDY: HOMESUITE APPLICATION

3.1 FIRST STEPS

3.2 HOMESUITE APPLICATION

3.2.1 Introduction

3.2.2 Homesuite architecture

3.2.3 Components

3.2.4 Configuration

3.2.5 Studying Homesuite behaviour

3.2.6 Sequence Diagrams

3.2.7 Summary

3.3 HOMESUITE VERSION WITH SPRING FRAMEWORK

3.3.1 Problems joining frameworks

3.3.2 Structure of the application

3.3.3 Implementation

3.3.3.1 Injection of control

3.3.3.2 DataAccessLayer

3.3.3.3 Service Layer

3.3.3.4 Presentation Layer
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3.3.5</td>
<td>Security in Homesuite</td>
<td>43</td>
</tr>
<tr>
<td>3.4</td>
<td>SUMMARY</td>
<td>43</td>
</tr>
<tr>
<td>4.1</td>
<td>TESTING</td>
<td>45</td>
</tr>
<tr>
<td>4.2</td>
<td>HOMESUITE’S PERFORMANCE</td>
<td>46</td>
</tr>
<tr>
<td>4.3</td>
<td>ANALYSIS</td>
<td>47</td>
</tr>
<tr>
<td>4.4</td>
<td>TEST CONFIGURATION AND METHODOLOGY</td>
<td>47</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Load profile</td>
<td>48</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Action List</td>
<td>49</td>
</tr>
<tr>
<td>4.4.3</td>
<td>User profile</td>
<td>50</td>
</tr>
<tr>
<td>4.4.4</td>
<td>Ports and associations</td>
<td>50</td>
</tr>
<tr>
<td>4.5</td>
<td>RESULTS AND EVALUATION</td>
<td>50</td>
</tr>
<tr>
<td>5.1</td>
<td>CONCLUSIONS</td>
<td>55</td>
</tr>
<tr>
<td></td>
<td>REFERENCES</td>
<td>58</td>
</tr>
</tbody>
</table>
## Abbreviation Table

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARP</td>
<td>Address Resolution Protocol</td>
</tr>
<tr>
<td>AOP</td>
<td>Aspect-Oriented Programming</td>
</tr>
<tr>
<td>DAO</td>
<td>Data Access Objects</td>
</tr>
<tr>
<td>DI</td>
<td>Dependency Injection</td>
</tr>
<tr>
<td>DLL</td>
<td>Dynamic Linking Library</td>
</tr>
<tr>
<td>EAR</td>
<td>Enterprise Archive</td>
</tr>
<tr>
<td>EJB</td>
<td>Enterprise JavaBeans</td>
</tr>
<tr>
<td>EIS</td>
<td>Enterprise Information System</td>
</tr>
<tr>
<td>HTML</td>
<td>HyperText Markup Language</td>
</tr>
<tr>
<td>HTTP</td>
<td>HyperText Transfer Protocol</td>
</tr>
<tr>
<td>HTTPS</td>
<td>HyperText Transfer Protocol Secure</td>
</tr>
<tr>
<td>IoC</td>
<td>Inversion of Control</td>
</tr>
<tr>
<td>J2EE</td>
<td>Java 2 Platform Enterprise Edition</td>
</tr>
<tr>
<td>JEE</td>
<td>Java Enterprise Edition</td>
</tr>
<tr>
<td>JDO</td>
<td>Java Data Objects</td>
</tr>
<tr>
<td>JCA</td>
<td>Java Connector Architector</td>
</tr>
<tr>
<td>JMS</td>
<td>Java Message Service</td>
</tr>
<tr>
<td>JSP</td>
<td>Java Server Page</td>
</tr>
<tr>
<td>JSTL</td>
<td>Java Server Pages Tag Library</td>
</tr>
<tr>
<td>MVC</td>
<td>Model and View Controller</td>
</tr>
<tr>
<td>ORM</td>
<td>Object-Relational Mapping</td>
</tr>
<tr>
<td>POJO</td>
<td>Plain Old Java Object</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>SSL</td>
<td>Secure Socket Layer</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
Chapter 1
Introduction and objectives

In this chapter, an overview of the thesis’ origin will be given, seeking to give the reader a broad perspective of the causes that moved Spring Framework to proposing it and of its primary objectives.

1.1 Introduction

In the past, increasing interest in providing programmers and designers with software developments, together with the development more and more of complex applications with all kind of services, has caused the search for new ways to carry out these goals. Software framework not only eliminates tedious low level programming but also, it allows developers to spend more time concentrating on the business-specific problem at hand. Johnson and Foote, who have researched about object-oriented programming and software architecture, especially related with frameworks, expressed the concept of framework as a set of abstract classes and the way their instances collaborate for a specific type of software like graphical editors, web application, compilers, multimedia or communicating between different systems.

A lot of frameworks emerged and each one was a new complex tool with a new methodology built to solve complex problems. But this supposed that the building of these alternatives had a main problem. Developers thought that simplicity was contrary to the concept of power. It unleashes the explosion of complexity in J2EE, EJB, and XML. Nowadays, some targets in the Java community are working to swing the pendulum from monolithic frameworks back towards cleaner, simpler ones. A number of new projects are helping to buck the trend toward complexity, choosing instead to build lighter, focused frameworks that can power simpler applications.

J2EE has emerged as standard platform to develop Java applications that resides in the server. This platform develops running distributed multitier architecture Java applications. A central theme of the J2EE specification was to enable the easy development of distributed web applications. In practice, the development of J2EE applications hasn’t been without difficulties and challenges. In many cases, these have led to open source developers providing innovative solutions. A lot of this is a reaction to the heavyweight complexity in the
mainstream J2EE world, but much of it is also exploring alternatives and coming up with creative ideas.

Some of alternatives help to reduce the complexity problem, but a common issue to deal with is how to wire together different elements: how do developers fit together this web controller architecture with that database interface backing when they were built by different teams with little knowledge of each other. A number of frameworks have taken a stab at this problem, and several are branching out to provide a general capability to assemble components from different layers. These are often referred to as lightweight containers, examples include PicoContainer, Spring, Struts or Hibernate.

PicoContainer is only a container that provides inversion of control (IoC), Struts is a example of Model-And-View framework that we will study later, and Hibernate is object-relational mapper with provides a framework that maps a object-relational domain model to a traditional relational database.

Open source communities such as Spring, JBoss and Hibernate have played an innovative role that has contributed to ongoing discussion within Enterprise Java community and the evolution of Java EE specification.

This doesn’t suppose that J2EE and its EJBs are bad implementations and must disappear, because, we must remember that the EJB component model was a welcome breath of fresh air. The fact is, there is a tremendous investment out there in "corporateville" in all things J2EE. It may feel righteous to declare that we must throw away all previous work and retool, but that kind of thinking isn't grounded in good business acumen. EJB continues to evolve and with it idioms, practices, and frameworks spring up that complement the J2EE API.

Spring is a lightweight alternative framework that replaces the complexity of Enterprise JavaBean model, which J2EE contains in its specification. Spring combines JavaBeans, an Inversion of Control (IoC) container and Aspect Oriented Programming (AOP) to address areas not well served by other frameworks, allowing for easy adoption, delivering easy of use, being non-invasive and making testing easier.

This thesis doesn’t try to teach how you must use Spring framework because there are a lot of books that explain it. The thesis will try to extract the main conclusion of comparing Spring behaviour and, the advantages and disadvantages it provides us regarding the web application technology most extended, J2EE. Finally, you decide if it's worth changing your framework to Spring.
The project was divided into three parts: a conceptual research of J2EE specification and Spring framework, where we will know architecture and components of both technologies. A second part is dedicated to study a J2EE application Homesuite, which is a portal which helps user to manage their information himself. We will also develop a version of Homesuite under Spring to have a real experience with this framework. Finally, the third part is a testing of both applications to evaluate the throughput and efficiency of each one.

In the first part, we will give the J2EE’s and Spring’s fundamentals that will be throughout the book.

- In the first chapter, we’ll do a going over of J2EE specification, its components (JSPs, sevlets, EJBs) and J2EE’s behaviour.
- The second chapter will be dedicated to understanding Spring framework. We’ll explore its components and we learn about .

The second part is the implementation of a web application under Spring framework.

- In chapter one, we will get to know Homesuite, a web application done with J2EE specification and uses components like EJBs. Furthermore, we will study some functions like log on, register, insert contact and send an email which later we’ll implement in Spring.
- Chapter two is the development of Homesuite application under Spring. A real experience using Spring. This chapter is divided into different sections, each of them focusing on an application tier: persistence-tier, service-tier and web layer.

Finally, we’ll do a few comparison of both technologies, based on acquired experience during this phase: suitable technology for which application domain, code of Homesuite, development experience and components of Spring and J2EE.

Part three tests both versions of Homesuite using Spirent Communication Avalanche 2500. In this part, we will evaluate the performance of the two versions of Homesuite (Spring and J2EE) which use HTTPS connection, and the results of Spring version of Homesuite with the protocol HTTP. With these performances we will try to clarify whether the power and simplicity that we have been talking about at the beginning of this article, is really possible in an identical application by looking at the results of the testing.

Finally, conclusions will be drawn from the experience gained during the research of Spring and J2EE.
Chapter 2

Study of J2EE and Spring Framework

The first websites were collections of web pages linked among themselves using HTML. Nowadays, the websites include multimedia, e-mail and other kinds of complex web applications such as voIP and e-commerce. This improvement is because of web server technologies improvement. The development of Server applications and technologies of web content development have provided the necessary architecture and components to build easily enterprise applications and that these applications have more capabilities.

In this chapter, we will study two technologies of enterprise applications development: J2EE and Spring framework. The first one is a standard used by most of the Java community developers in the last few years. And the second one is a new framework with MVC architecture which comes to make easy the developer’s work and substitute the deficiencies of J2EE has.

2.1 Java 2 Enterprise Edition

The J2EE specification (Java 2 Platform Enterprise Edition) is a platform to develop, deploy and carry out enterprise applications on the top of the Java programming language that is based on a distributed multitiered application model that enables the developer to divide the application across multiple tiers. J2EE applications are generally three-tiered applications which mean that applications are distributed over three locations: client machines, Java Server machines and data server machines.

Client machines are entrusted with the presentation and interaction with the users. They allow the connection with the second tier process. J2EE server has these processes, which manage the business logic of the application and access to third level services. Data server machines contain the data that is needed for the application. Their services are protected and must be accessed through second tier components.

The J2EE standard is based on multitiered programming, this model has three types of application components:

- Enterprise beans that are executed by EJB container.
- Servlets and JSP pages that are executed by Web container.
• Application clients that are executed by client machines.

There is a container for each kind of application component. The components are assembled in a J2EE application, are verified to be well formed and in compliance with the J2EE specification and are managed by J2EE server that provides support of runtime and contains EJB and Web containers.

J2EE containers provide application components with APIs that are used to access to services. Containers are also capable of controlling security, resources, state and transaction management and problems with the names.

Once the implementation of J2EE components is finished, there is a process of assembly, which provides the final application. This process lies in encapsulating J2EE components to modules and these are encapsulated on the application. Each application module contains a J2EE deployment descriptor. The deployment descriptor is a XML file that describes the deployment settings of an application, a module, or a component (servlets, EJBs). Deployment descriptor information is declarative, it can be changed without the need of modifying the source code. In an application that uses enterprise beans, we need two types (J2EE and runtime) of deployment descriptors, because we need to define enterprise beans in files of both types.

From a request is received by web container, until a response is returned to the client, there are involved many J2EE components. The process starts when client sends a request. The Web container receives it and must map the request to the component that can handle it. If the request is another web page, the container normally calls a corresponding JSP page, but if the request has an action/function then it is usually a servlet which solves the request. Thus, the container creates an instance of the servlet and afterwards, invokes the service method that resolves the request. Sometimes service methods can make use of EJBs to resolve business logic, these have support to implement business logic more easily than servlets, besides leaving an application better structured. A JSP page can call Enterprise beans using custom tag libraries; The components that take part of J2EE application are explained below:
2.1.1 Enterprise JavaBeans

An EJB is a component which implements the business logic of an application. The business logic is the code that fulfills the purpose of certain application function. An enterprise bean can be used alone or with other enterprise beans, and its code can be recycled. Its container provides transaction management and security authorization, which makes developing enterprise applications easier.

The use of enterprise beans should be considered when there is the need to distribute an application’s components across multiple machines. That way, the application will be able to support the growing number of users and the variety of them. Furthermore, it will be able to use transactions that manage the concurrent access of shared objects.

There are three kinds of enterprise beans: session beans, entity beans, and message-driven beans.

Session beans

A session bean represents a single client inside the J2EE server. It performs work for its client, executing a business task inside the server. A session bean is not persistent and when the client has what he wants the session bean terminates. There are two types of session beans:
A stateful session bean instance is used by a client during its cycle of life, in which the client carries out a group of method calling related with the time of this bean. The state is kept across multiple method calls by the same caller/user.

A stateless session bean instance is used by multiple clients during its cycle of life. That is, in contrast to stateful session bean, the state is not kept across multiple sequential calls by the same user. Therefore, these session beans should be used for business logic processes that can be finished with a method. A stateless session bean can implement a web service, but other types of enterprise beans cannot.

The stateful beans must be used when necessary, because using stateless beans improve the scalability, maintenance and debugging of an application.

Entity beans
An entity bean represents a business object in a persistent storage mechanism. Unlike a session bean, its state is saved in the storage mechanism therefore it is persistent. Moreover, it allows shared access when a transaction is executed and can participate in relationship with other entity beans.

Message-Driven beans
A message-driven bean is an enterprise bean that allows J2EE applications to process messages asynchronously.

EJB and JMS (Java Message Service) containers work together to process messages. When a JMS message arrives from another application component, the EJB container sends it with the onMessage() method to a message-driven bean instance that processes the message. In several aspects, message-driven beans and stateless session beans are similar.

Resource adapter JCA (Java Connector Architecture) also needs EJB container to interact with an enterprise information system (EIS). When a message from an EIS arrives to the resource adapter and this is sent to message-driven bean, where the message is processed. EJB container also provides some services like transaction support for this type of enterprise bean.

2.1.2 Java Servlet Technology

A Servlet is a component of the J2EE server, and it extends the capabilities of servers that host applications which are accessed by way of a request-response programming model: they
can support dynamic content of web pages, access to database, provide multiple clients services at the same time, invoke other Web resources such as EJB.

The interaction between client/server based on web use the protocol HTTP. When a HTTP client request is received by the Web container, this starts the required servlet. The servlet processes the request and generates a response to the Web container, which directs the response to the client. The response contains the response state and meta-information describing this response.

The life cycle of the servlet is controlled by the container in which the servlet has been deployed. When a request is mapped to a servlet, the container creates a servlet instance and calls the init() method, where every configuration parameters like database connections, files and resources are initialized. No methods can be called in that method if init() doesn’t initialize the servlet.

If a client has sent a request to the Web container then, the servlet invokes the service method, which is entrusted to process the request information and to send a response to the Web container. Sometimes, the HTTP mapped request is a method such as GET, POST, HEAD, OPTIONS, PUT, TRACE, DELETE, CONNECT, etc. in these cases that servlet invokes a doMethod, where Method takes the value of those request methods. Finally, if the container needs to remove the servlet, it finalizes the servlet by calling the servlet's destroy method.

2.1.3 JavaServer Page Technology

A JSP page is a text-based document that allows creating web content with static and dynamic components. Any text-based format such as HTML, WML, and XML, would implement static content while JSP elements construct dynamic content.

Each function that a client can do in an application has its corresponding JSP page. JavaServer Pages (JSPs) technology is an extension of Java servlet technology and combines HTML and Java code into a single file. Nowadays, many Web development methodologies and architectures prefer to separate HTML from Java code. Thus, HTML would be used to develop presentations aspects and Java implementation to processing logic.

Java code is contained within a JavaBean. The business logic placed in JavaBeans can be used by different JSPs. For invoking these operations on JavaBeans components, JSP technology provides a mechanism named custom tags which are usually distributed in the
form of tag library. Some companies provide custom tag libraries to their customers for free or for individual purchase, but can also be found online.

When a JSP page containing a custom tag is translated into a servlet, the tag is converted to operations on an object called a tag handler. The Web container then invokes those operations when the JSP page's servlet is executed. Some of the wide variety of operations that custom tags have, include implicit objects, process forms, access database or perform flow control.

Some of these custom tags have bugs or do not work correctly. The purpose of the birth of Java Standard Tag Libraries was to standardize some custom tags whose functionality was found in many JSP applications. The tags provided in JSTL can be used within any JSP-compliant container. The advantage of using JSTL over a vendor-specific tag library is, obviously, that you won't be bound to a specific vendor's container.

JSTL has tags for four different areas: core, those tags as iterators, conditional processing and expression support. XML processing which provides tags that allow manipulating XML. Internationalization tags that give support I180N, this referred to the tags that can handle multiple languages and other conventions without the need for redesign them. And SQL tags which are entrusted with database access from within JSPs.

2.2 Spring Framework

Spring is a framework, created to make up for the deficiencies that J2EE specification has. Spring’s main aim is to help to reduce the complexity of J2EE development and to promote good programming practice. It is based on enabling a POJO-based programming model to be applicable in a wide range of environments.

Spring promotes loose coupling through a technique known as Inversion of Control (IoC). It is entrusted with configuring and managing Java objects (beans). The container is configured by loading XML files that contain Bean definitions which provide all information that is required to create objects. Spring is modular and is divided into independent packages, which can function independently. The developers of an application can be implement just a few Spring packages and leave out most of the packages in Spring. These packages, which make existing technologies easier to use, are: core Container, DAO, ORM, AOP, JEE, WEB.
2.2.1 Core container – Inversion of Control

It is the heart of any Spring framework, uses JavaBeans and this package provides most of the basic functionality to manipulate JavaBeans and provides the basic infrastructure for the other Spring framework classes. This container also provides the basis of the Dependency injection (DI) pattern that Spring is based on.

The concept of Dependency Injection also referred to as Inversion of Control. While in traditional container architectures, each object is responsible for obtaining its own references to the objects it collaborates with. In IoC, objects are given their dependencies at runtime by some external entity that coordinates each object in the system. That is, dependencies are injected into objects. So, the container figures out that the component needs a certain object, and provides it at runtime.

There are two implementations of IoC: Setter injection is realized by calling setter methods on the beans after invoking a no-argument constructor or no-argument static factory or no-argument static factory method to instantiate the bean. Constructor injection is realized by invoking the constructor with the arguments which represent an object which bean has dependencies.

In this module we can find the BeanFactory, an implementation of a factory design pattern that applies IoC to separate the application’s configuration and the dependency specifications from the application code. It is really a class whose responsibility is to create and give beans out. It holds BeanDefinitions of multiple beans within itself and when a client asks for a bean then it instantiates the bean.

2.2.2 Context Container

Provides an alternative way to access objects seems to a JNDI register. JNDI is entrusted with organize and look for the objects or name of the objects in a distributed application. This module extends the concept of BeanFactory, adding to publish events to beans that are registered as listeners (Event propagation), internationalization support to resolve text messages and, a generic way to load file resources (resource-loading).

2.2.3 Data Access Object Container (DAO)

Spring provides all the JDBC related support required by an application. With DAO objects read and write data to the database. In the data access process, some steps are always required.
On the one hand, we have fixed steps like obtaining a connection to our data store or cleaning up resources when done. On the other hand, variable steps that manage queries and update the data. Spring separates these parts and provides templates and callback classes that manage each one of them. The only thing that developers must do is to implement the data access logic.

This helps to have the database code simpler. Also, it prevents problems that result from a failure to close database resource.

Springs provides an exception hierarchy which separates the data access ties from the rest of the application. This hierarchy descends from DataAccessException. Thus, we will not need implementation-specific exceptions or worry about proprietary error codes.

2.2.4 Object/Relational Mapping Container

This module provides an integration tier to object/relational mapping tools, such as Hibernate, JDO, Oracle TopLink, iBATIS SQL Maps and JPA: in terms of resource management, DAO implementation support, and transaction strategies. These tools are used when the structure of the object of the business tier does not correspond with the relational database columns where the object is connected. The mapper deals with the differences of both structures. The birth of Object/Relational provides these features that JDBC technology lacked.

2.2.5 Aspect-Oriented Programming

This module serves to develop application based in aspect-oriented programming compatible with AOP Alliance (is a project which tries to facilitate and standarize the use of aspect-oriented programming). Spring’s AOP support is to provide J2EE services to POJOs. The concept of AOP is not necessary to change the functionality of an object to be able to apply it how and where the developer wants.

The Spring AOP module also introduces metadata programming to Spring. Thus, we can to add annotations that instruct Spring on where and how apply aspects.

2.2.6 Java Enterprise Edition Container

There are several enterprise services that Spring doesn’t support directly. Spring makes use of other APIs to provide the service, then places them under an abstraction layer so that they are easier to use: Some of these enterprise services include: Java Naming and Directory Interface (JNDI), E-mail, Scheduling, Java Message Service (JMS).
This module also has the possibility to access EJBs, implement EJBs and functionality within them. Spring provides support for them in two ways:

- Declaring EJBs as beans within the Spring configuration file. This makes it possible to wire EJBs references into the properties of your other beans as though the EJB was just another POJO.
- Writing EJBs as a façade to Spring-configurated beans.

### 2.2.7 Model-And-View Framework

Spring’s MVC package is found on Spring’s WEB module. It contains a framework that implements the presentation-tier of a web application. It is based on Model-View-Controller (MVC) pattern. This architecture allows separating the business code and the web forms. The model, representation of the information with which the system carries out some functionality, the view is the user’s interface and the controller handles the request from users, and depending on the request invokes changes in the model and normally in the view.

Spring’s Web MVC framework is designed around a DispatcherServlet. It receives the request from the client and handles it. Furthermore, it is responsible for managing the handling of the request. Configured via IoC, it delegates the control to the beans, which build the response.

The Spring DispatcherServlet is formed of beans which it uses to process requests and render the appropriate views. The figure shows that process:

![DispatcherServlet’s behaviour](Figure 2.2: DispatcherServlet’s behaviour)
First of all, the client sends a request that is handled by the DispatcherServlet (1), with a Handler Mapping bean the DispatcherServlet knows what controller will be responsible for handling the request (2). The Controller performs the business logic of the function that the client need (3). Afterwards, when the controller has finished its functions, it returns a ModelAndView to the DispatcherServlet (4). If the ModelAndView contains a logical name of a View, DispatcherServlet looks up the view object in a ViewResolver (5). Then, DispatcherServlet dispatches the request to this View (6) object which will be responsible for rendering a response back to the client.

To configure the DispatcherServlet and use components which take part of MVC framework, it is necessary to know the task they carry out.

*HandlerMapping* maps requests to appropriate controller using URL patterns. This bean relates controllers with a URL patterns. There are three implementations:

*Controller* handles all incoming HTTP requests from the browser. It is the manager of obtaining a ModelAndView object based on the outcome of performing some business functionality, which controller has implemented. The model is a map, and a view is the page JSP/HTML where model will be used.

Spring provides a rich controller hierarchy, table 2.1, which helps developers to find a controller for his needs.

<table>
<thead>
<tr>
<th>Controller type</th>
<th>Classes</th>
<th>When do you use it?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple</td>
<td>Controller (interface) \nAbstractController</td>
<td>a simple controller that doesn’t have almost functionality</td>
</tr>
<tr>
<td>Throwaway</td>
<td>ThrowawayController</td>
<td>developer needs a single way to handle request as commands</td>
</tr>
<tr>
<td>Multi-Action</td>
<td>MultiActionController</td>
<td>an application need to perform multiple actions, and each action has different method</td>
</tr>
<tr>
<td>Command</td>
<td>BaseCommandController \nAbstractCommandController</td>
<td>developer needs to take one or more parameters from the request and bind them in an object</td>
</tr>
<tr>
<td>Form</td>
<td>AbstractFormController \nSimpleFormController</td>
<td>it is necessary to show a form to the user and also process the data entered into the form</td>
</tr>
<tr>
<td>Wizard</td>
<td>AbstractWizardFormController</td>
<td>developers must process forms that span multiple pages</td>
</tr>
</tbody>
</table>

Table 2.1: Types of Controllers
**ViewResolver**, takes the view name returned in the ModelAndView and maps it to a view. It has a names relation between the view name which was returned by the controller and their corresponding JSPs. Thus, the developer does not need write all path of the JSP and only write a name that identifies it. Spring provides four implementations for this bean, but for views which are rendered by JSP is better use InternalResourceViewResolver.

**View** is a bean that renders results to the user. Normally the type of view that is used is JSP but Spring gives support to others such as Velocity and FreeMarker templates or PDF. This bean takes the ModelAndView object and inserts the data that model provides in View. Spring provides a library tag for JSP, which helps to implement these web pages.

### 2.2.8 Web Container

This module provides the basic properties to integrate some web components to Spring, such as multipart functionality, context initialization with servlet listeners and a context oriented web. When Spring is used with WebWork or Struts, allows the integration with those environments.
Chapter 3

Case of study: Homesuite application

The compiled information about J2EE and Spring framework have helped to understand the behaviour of both technologies. In addition to showing the manner of implementing them. Due to time constraints it was impossible to research in depth the studied technologies. With the purpose of tackling both technologies in more detail to try to reach the goals of those such as, the ease to develop a web application under Spring, need to be studied over a more practical method which gives developer a knowledge closer to both technologies.

This chapter experiments with both technologies, using the previous research. The development of J2EE and Spring project will give us a real experience with these technologies and a more critical view. It will consist of the use the most way of the modules or components that those provide. Several ways were studied:

a) Building two applications, first for J2EE platform and the other for Spring framework. With that solution we’ll explore the two technologies. It was a superficial study and with a few conclusions.

b) Selecting a good J2EE finished project and developing it under Spring. This is more interesting because we would work with more components from the Spring framework and we would know J2EE components through a sample.

The application that we decided to use as of sample was Homesuite. It had some properties: a complex application, the use of EJBs and it covers the majority of capabilities that J2EE provides. The problem: it was built under Pillar framework, a technology created by the same developer, and that doesn’t have a specification where we can collect information.

Before beginning with the implementation under Spring we must study the requirements and the structure of the application to develop. In this case, we already have advanced work because we have a sample application. Even so, Homesuite is a complex application that needs to be studied in depth to know its requirements, components, functions and the big structure that form it. Although this part is not really practical, it gives us more specific knowledge about J2EE components.
The second part of this chapter illustrates the development of the new application, in which we’ll try to follow the structure of Homesuite and will apply Spring features. As a result of this implementation there should be an application more structured and tidier code than J2EE.

### 3.1 First Steps

For the proper implementation of any application, developers should have experience with the prospective technology. It was our first practical contact with both technologies and we decided to devote a time-period to develop an easy application for the two technologies, where we could use some of the existing components of both technologies. This web application was to compute Fibonacci formula. It would not access to the database but J2EE would use of EJB and servlets while in Spring, we would know its MVC framework.

In one hand the implementation under J2EE was followed as: implementing a main JSP wherever was required by inserting a number to compute the Fibonacci computation. A servlet that handled the request of the client, created an instance of session bean that would be entrusted of business logic. Finally, in the servlet code, we implemented the JSP outcome page. In J2EE implementation, we used XDoclet, which helped to create XML files of web application automatically (EJBs/Servlet) because the writing of these I was very tiresome.

In the other hand, Dispatcher Servlet of Spring handled the request, called to handler mapping which mapped the controller. This controller was entrusted in the business logic and returned a ModelAndView object. Afterwards, ViewResolver bean of DispatcherServlet looked up the view of the ModelAndView instance that corresponded with JSP page.

Thanks to this exercise, we knew the behaviour of both technologies and their components besides of distinguishing the first differences between the technologies and obtaining preliminary conclusions:

- Although one of the features of spring is the **inversion of control**, this application is not a good sample to know and understand the behaviour of IoC. In the DispatcherServlet configuration a bean has been injected in a controller with setter injection. Thus, this allows controller that does not have to look for the bean resources. Figure 3.1 and 3.2 shows how we do inversion of control.
Chapter 3: Case of study: Homesuite application

However, in the J2EE, servlet needs to look up the enterprise bean context in the initialization method. This feature will be important when controller might access to service and persistence layer.

- **Spring takes off functionality to its servlet** (DispatcherServlet). This is entrusted with handling the request to its beans that will resolve the request. Controller performs the business logic of this servlet. In J2EE, the servlet resolves itself the request. It is the one in the web container with other servlets.

- **Spring maps the controllers to a URL pattern** using handler mapping bean. Furthermore, developers must indicate URLs that are handled by the DispatcherServlet. In J2EE, the mapped component is logically servlet.

- **Controllers do not need a constructor or init() method like servlet.**

- **Spring MVC framework.** The idea of this model of MVC framework is to separate business logic and user interface to have a tidier and clearer code. Thus, controller creates a ModelAndView object that redirects to JSP page (view) and sends data (model), in a line. In J2EE, if we want to show a JSP page from a servlet, we implement the JSP code into the servlet. But this solution is chaotic, when we implement complex application. It is desirable to separate business logic and user interface to have a tidier and clearer code. J2EE proposes an instance of RequestDispatcher, which is a servlet to redirect to JSP page. The figure 3.3 shows how RequestDispatcher redirects JSP.

```java
public void setFibonacci(fibonacci fibonacci) {
    this.fib = fibonacci;
}
```

Figure 3.2: Fibonacci setter method in the implementation of nameController
The deployment descriptor describes web container of J2EE, its elements and the way to access their. For J2EE are described elements like EJBs and Servlet. In Spring, it only describes DispatcherServlet, because this component handles the rest of components (beans) of the web container. Spring uses XML named Fibonacci-servlet.xml file to configure its beans (controller, viewResolver, handler Mapping).

Enterprise beans needs four files: the business interface, the home interface, the bean implementation, and the deployment descriptor. Spring defines the implementation as a POJO and wire in any additional services needs through injection or AOP.

Enterprise beans are more rigid. To use services provided by EJB container, developer must be use javax.ejb interfaces. It is difficult to use a component outside of an EJB container. Spring use POJOs do not need to inherit from framework classes or implement framework interfaces. Making that the code can be also reused out of Spring.

EJBs are configured in the EJB runtime deployment descriptors.

Entity EJBs fall short. This bean uses a Value Object pattern to pass data to and from the EJB container. That is, developer writes the persistence property twice: once in the entity bean and once in his value object. With the use of Spring with a ORM framework, the entity object is not coupled with their persistence mechanism. [3]

At first glance, Spring web application programming is simpler and more logical than J2EE. The use of a ModelAndView framework which separates the view and the business logic become that the code is more structured and easier to understand. Its components are more independent and reusable than J2EE.
3.2 Homesuite Application

3.2.1 Introduction

HomeSuite is a portal that contains a collection of web-based applications which that manages, controls and retrieves user's personal information. Developed using J2EE features, it uses different databases which can automatically be synchronized, furthermore users can share data. Some of the services that Homesuite offers are:

- E-mail: Pop as many e-mail address as you has used. Unlimited number of filters, folders, and disk space.
- Contacts: is like an address book. It also maintain a call-history with contacts for audit purposes
- Calendar: events, reminders that you have to do.
- Diary: Setup multiple diaries for recording your thoughts and actions.
- Bookmarks: are pointers, whose purpose is to easily catalog and access web pages that the web browser user has visited or plans to visit.

Homesuite is a similar portal to Yahoo! Mail. Its complex structure offers us a great variety of different components. Furthermore, a module of the application consists of Pillar framework components. Pillar allows you to map url's to controller objects which are like Struts' Action objects. It also allows you to retrieve resources easily like strings out of your properties files. Although the study of that framework is out of reach, we’ll study the Pillar’s components that Homesuite uses, but this framework will not be part of this research.

Struts is an open source MVC implementation that provides base controller functionality that you can extend and enhance in applications. The base controller is implemented as a Java servlet, and its configuration is controlled by an XML file called struts-config.xml. When a Struts-based JSP Web application is created correctly, most changes in flow control are made in the struts-config.xml file rather than in the code itself.

In this chapter, the Homesuite portal will be explored; architecture, modules, functions and components that make up it. Via sequence diagrams we’ll study the behaviour of objects that are called by an application function.
3.2.2 Homesuite architecture

Homesuite is the result of the use of some projects of Michael Remijan, each one entrusted with a different aspect for the construction of an application. While the module PillarFramework gives developers the necessary components to build web layer of a portal, D.S.R. (Data Storage and Retrieval) is designed to provide the business logic using EJB in front of the database. The rest of the modules, Utilities and LibExt support components which are needed for Pillar and D.S.R modules. The following figure 3.4 shows this architecture under what homesuite is built.

Figure 3.4: Dependencies between modules which support HomeSuite

/LibExt, contains all of the 3rd party libraries which the entire code is dependent on.
/Utilities, provides the following components:
- JStandard: This project is designed to provide additional functionality and facades over the classes available in J2SE
- JSwing: This project is a subset of jStandard. jSwing focuses on developing some easy to use Swing components as well as tools for working with standard Swing components.
- JEnterprise: This project is a subset of jStandard. JEnterprise is designed to provide functionality and facades over the standard J2EE classes and to make a set of tools that make creating J2EE applications easier.
• QCron: This is a scheduler that is similar to cron (instructions are carried out periodically) but its configuration is much different and simpler.

• Cache: A very simple cache with some moderately advanced features such as:
  - Set maximum size of cache.
  - Specifying how long objects live in cache.
  - Scheduled purging of stale objects.
  - Automatic purging of stale object if memory becomes low.

• Pool

/PillarFramework: This is a J2EE webapp servlet/jsp framework.

/D.S.R (Data Storage & Retrieval): EJB Tier. All application access the database they through the business logic in this project. This module creates business processes in the form of Enterprise Java Beans (EJB).

/HomeSuite: is the web application is being studied.

3.2.3 Components

Org.moss.dsr.ejb
It contains session and entity beans of the web-application. They are called in the service layer and are commissioned to manipulate database.

Org.moss.dsr.bean
Data transfer objects between session beans and the web application. These beans manage the filter properties of the email interface.

Org.moss.dsr.command
Created by the web application and sent to the Session beans to tell the Session bean what to do. These components are commissioned for checking the persistence of datastore. A command is the instance that we want to manipulate in the database.
**Org.moss.homesuite.service**

It provides the classes that manage the service layer of an application function. Web application never knows which Enterprise Beans are being used. Services are components that work to communicating controllers with EJB. They receive checked information from controllers and create appropriate commands for managing datastore.

**Org.moss.homesuite.bean**

They are objects that contain information that will show in JSP pages. These are created by services and get information of the instance that has been managed in the database. Their data can be stored to the session of the web application, but normally their data has been manipulated correctly in the database. Data transfer objects from org.moss.dsr.bean are convert into these beans which the web application then uses. This isolates the web application from changes in the objects in org.moss.dsr.bean. In that way, it is protected from incorrect changes in the database, and ensures the coordination between belonging data to user’s session with the data that are managed in the database.

**Org.moss.homesuite.controller**

Handler of requests from the user, its work begins when the users submit a form. It receives the requests, checks the request information and attempts to process the request. Controllers are the heart of the web layer. Their service() method, which is automatically called when the controller cannot figure out what operations to perform, connects with the service layer. It returns to NavigateTo object that provides all necessary information to JSP pages to link with the next page that will be showed.

NavigateTo is a class that contains the information needed to construct a URL to navigate to. It notifies to JSP pages the state of the request which was handled by the controller.

**Org.moss.homesuite.form**

Its components store data of the requests. With them, the controller checks and validates the request received.

**JSP Pages**

The web content is made under JSP pages. Homesuite includes different taglibs: struts tags, pillar tags, and its own tags.
The Struts framework provides some valuable custom tag libraries. The bean tag library has been used to make data available to further tag processing. Most of the tags of this library are used to define and create beans to hold the data, but there are also a couple of tags to insert data into the beans. `<bean:write>` is the most used, it inserts the value of the specified bean property into the current JSP page being rendered.

The logic tag library has custom tags for iteration and tags for if-then-else structures. The HTML tag library features many useful custom tags, including custom tags for input form tags and form item tags used in Struts' form handling and validation.

The Pillar framework provides its own tag library, though it is not too large, it contains tags entrusted with presentation task of some sections of an application such as error messages, titles and dates.

We must differentiate `<pillar:url>` tag from the rest because this tag maps all appropriate requests to the Pillar controller. This links with the controller that handles an action of the JSP page, besides of attaching with other theme files of Homesuite. It is used by users to access and manage users account information (email, newsletters, bookmarks, diaries) and manage of this. Homesuite.properties is a file where you can find the relation between url aliases(tags) with controllers or JSP pages. The purpose of this properties file is to provide localization to the web container components. Some of URL Aliases are related to controllers, this indicates what tag (url alias) is handled by controller.

Finally, there are Homesuite custom tags which have been created to access Homesuite beans and can manage the data that will be sent to the controller.

### 3.2.4 Configuration

The `web.xml` file is read when the JSP container starts. This file defines the configuration of the Homesuite application, the servlets and each controller that is under their control. Furthermore, it also contains the properties file with the needed information to map the requests with the controllers.

### 3.2.5 Studying Homesuite behaviour

From the time that the request is received by Pillar until the time that a response is returned to the client, many pieces of the Homesuite application are involved. Most of the functions Homesuite provides, have a similar steps. It starts when a user submits a form or clicks on hyperlink, then the request is sent:
1. The request is looked up in a properties file, where it checks which controller has assigned the URL pattern that the request carries.

2. Then, Pillar servlet passes the control to corresponding controller object. The controller figures out what to do and uses a service object that returns a NavigateTo object to do it.

3. The service creates a command according to the data received and that later they will be managed in the database.

4. The service gets a session bean and passes it the command to perform some business method.

5. The session Beans are the client interface to the entity beans. The session bean uses an entity bean to insert, modify or delete a data from the database.

6. After the entity bean has managed database, the session bean converts the entity bean into a POJO from org.moss.dsr.bean and returns this POJO to the Service if everything has gone well, to the service layer.

7. The service converts received POJO and converts it into another POJO from org.moss.homesuite.bean.

8. The service passes this POJO back to the controller.

9. The controller puts the POJO into the request or the user’s session and forwards it on to a JSP page.

Service objects handle all function-related matters. They communicate web layer with EJB layer. Session beans retrieve and checks entity beans when someone wants to modify the database. That means, the sessions beans access to entity beans. On the back end of the application, the entity beans are the persistent items that could be stored in the database.

### 3.2.6 Sequence Diagrams

The previous section, explained in general terms how Homesuite works, and the components and modules that it uses. In the next step, some implemented functions of Homesuite are analyzed in depth and with sequence diagrams we will explain the process that follows each function and the interaction between most of the components of the application.

A sequence diagram is an artefact for dynamic modelling, which focuses on identifying the behaviour within a system. This kind of diagrams is used primarily to show the interactions between objects in the sequential order that those interactions occur.
The four functions that we have chosen to analyze are: Log on, register, insert contact and send an email.

**Register**
If a user wants to have an Homesuite account, he has to register. To register, Homesuite needs certain user data (email, password and display name) which is stored in the database. If some of these data are wrong, then the application shows a message with the restrictions that haven’t been complied with. The next figure is a sequence diagram of the function register:

![Sequence diagram of the Homesuite function “register”](image)

Figure 3.5: Sequence diagram of the Homesuite function “register”

A user who wants to register in Homesuite, has to the go to main page of Homesuite and clicks on then component of website named “Become a member”. Then, Register.jsp page is opened.

On that page, the user inserts all necessary information to create an account (email, password, display name). When the user fills in these parameters and clicks Register, this request is managed by the RegisterController that checks if all data complies with the restrictions. If something was wrong, it returns to the JSP page messages with the errors. Otherwise, it tries to register this user in the database. It uses an AccountServices object, which receives this information and passes it to a command, creates AccountLocalSession which tries to give this instance to the entity bean (Account). The entity bean is stored in the database.
**Log on**

This function makes it possible for a user to access his/her account. If he inserts his login and password correctly, he can see every field that Homesuite has prepared for him. If he does not have an account or, if his login or password was wrong the application shows a message indicating what happened. The figure shows the sequence diagram corresponding to Log on.

![Sequence Diagram of Log on Function](image)

**Figure 3.6: sequence diagram of the Homesuite function “log on”**

There is a Homesuite user who wants to enter his account. After he has accessed the main page of Homesuite, he writes his email and password in section Members and submits it with the button “Login”.

Left.jsp page controls that event. This passes the control to LoginController that manages the request. If you don’t specify any function of the controller, it automatically calls the function service. Then, the controller checks if the email and password from the user input are correct. That is, if they carry out restrictions like login with more than 5 letters, and a password well-formed. If this doesn’t occur then the controller returns an explanation about the errors that happened. Otherwise, it will look for the user’s mail in the database. For that purpose, LoginController passes the user data to an AccountServices object. That service drives this data to the correct session bean (AccountLocalSession) which interacts with entity bean (Account). Finally, the entity bean checks if the data of the user is in the database.

If the data is found, then the EJB tier returns this data in a way of entity bean to the service tier. The service object converts this user information in a bean and, returns it to the controller. The controller receives the exception or the bean and sends a message to left.jsp about the result of the query. Finally, if everything is correct, the servlet opens the homesite of user.
**Insert Contact**

In a diary we have a section named “Contact” where we note down telephone numbers, emails and other properties of a person whose data we want to record. This function tries to replace the typical diary section contact. The difference is that in an application we have to store more fields that we’ll be able to use in others functions of the application.

![Sequence Diagram of the Homesuite function “insert contact”](image)

**Figure 3.7: sequence diagram of the Homesuite function “insert contact”**

A login user wants to add a new contact to his contact list. He submits in the Contact tab. Then, the navigator shows a main page of contacts with a list of contacts that the user has already added. For adding a contact he must click on the link “Entry”. Finally, the user will have to fill in the form that will be opened.

When the user clicked the save button, the control of the action passes to the servlet that looks up the controller that will handle this action. This controller is the ContactController, that handles the request and checks that all the data is correct. Afterwards, the application can save this data, by means of ContactServices, which calls Session Bean (ContactLocalSession) which checks the persistence after inserting the user’s data in the database. Finally, If everything is correct the browser shows a informative page about if the data is inserted correctly or not.

**Send an email**

This application also provides the possibility to sending emails to some contacts on your contact list. This function is not available in most other portals. In this case, you send an email to a category of your contacts. It is a newsletter.
In this activity, the user wants to send an email to some contacts of his list. First of all, user has to go to the Inbox folder where he will click in the folder named “Sent”. Then the browser shows the homepage of Newsletter.

This application is implemented to send newsletters. The user can send an issue from one of the newsletter that already existed, or create another newsletter and send an issue for this newsletter.

In this case, user has created a newsletter, the sequence of sentence of this part is like activity “insert contact”, only change the components that process this activity change: NewsletterIssueController, newsletterIssueServices, newsletterIssueLocalSession, newsletterIssue.

When the newsletter is saved, user must click the “send an issue” link and the browser shows a page like any send e-mail webpage, user must fill in all fields. Afterwards, the user clicks “next” and the browser shows a confirmation page before sending an email. Finally, the user sends the email and newsletterIssueController handles this function (send issue), which uses EmailServices, and it is this object which is in charge of sending the issue to all subscribers.
3.2.7 Sumary

Homesuite is a J2EE application which is implemented with the support of Data Storage & Retrieval (D.S.R) and Pillar framework. While D.S.R provides the business logic of the application as through EJBs, Pillar framework helps to build the presentation tier.

The heart of the Pillar framework is the controller that handles the received request from the JSP page, and figures out what to do and uses an object from service layer to do it. This object is entrusted with getting a Session bean and gives it a command with data that session bean needs to perform some business logic. The Session bean uses an Entity bean for managing the database. Finally, the Session bean converts an entity bean into a POJO that will be passed until the controller, which puts the POJO in the request or user Session and sends it to the JSP page, which interprets the request and links it with the corresponding page. The NavigateTo object is the request that will be sent to the JSP page, and also informs about whether if the process that the controller has done, has gone well or not.

A sequence diagram is a technique that shows what the behaviour of components of an application function is. It has proved that the previous schema is repeated in every function that we have studied in this chapter. Homesuite uses components from each package in the same order. Only name components change depending on the activity.

In the next chapter, we’ll build the Homesuite application under Spring Framework. This has two purposes, learning how a web application is built on Spring and comparing simpler or tidier that can become to implement an application under this lightweight technology.

3.3 Homesuite version with Spring framework

In the preview study, we learned about the basic operating of some components which allow to develop an application more easily. But when we talk about the Homesuite application, we don’t only have components that are part of the J2EE specification like EJB’s, servlets, etc. It was implemented using a framework (Pillar), which is a custom solution developed by the Homesuite programmer. This makes the understanding of the application more difficult, because other specific components and elements that we are involved that cannot be integrated with Spring. On the other hand, studying Pillar, its components and it involvement in Homesuite, can help us to better understand the goals that the frameworks have inside J2EE.
In this chapter, we explain how we have implemented Homesuite under the Spring Framework. This doesn’t consist of comparing components of both frameworks. We only show what procedure we have followed for developing this variant of Homesuite. We try to give our experience programming with Spring framework. We replaced some components in the application with the components and modules that Spring gives us to implement the application.

Given that Homesuite is a big and complex program and the period that we have to implement it is two months, we decided to develop some of the modules of Homesuite that identify the usual uses-cases. We will implement the components that we studied in the preceding section (log on, insert contact, register) in addition to log out and modify account.

### 3.3.1 Problems joining frameworks

First of all, we tried to join the Homesuite application with the Spring framework, with which we wanted to modify the application as little as possible. Spring has a service that helps to join applications with EJBs, so we only changed the MVC tier of the application, and the xml files that are used to configure it, so everything would work fine. After doing this and tried out the new application, we never got to show it. The problem was that the application is done under Pillar framework and Homesuite is being used in most of its components and we didn’t know the way to connect those under Spring. We had to change these elements or remove them because Spring doesn’t need them. The solution was to start to develop the application from zero. The MVC of Homesuite was already changed and lacked implement the part of EJBs and intermediate components between MVC and EJBs.

As we said, Spring has the chance to configure the application so that we can use EJBs. This was very interesting for our project because that made our implementation easier and it implied that Spring is not only a new technology, with different methods, components and modules. In addition it also embraces other technologies like EJBs. This chance was ruled out because, in the EJBs tier, Homesuite also used classes implemented under Pillar framework, that we didn’t know at that moment, and it supposed another study about Pillar Framework, that it’s really unnecessary because the framework was developed by Michael Remijan to use by him. Spring comes with a family of data access frameworks that integrate with a variety of data access technologies. We will persist our data via direct JDBC, Java Data Objects (JDO), or an object-relational mapping (ORM) tool like Hibernate. Spring removes the tedium of data access from our persistence code. This DAO support allows one to switch between the
aforementioned persistence technologies fairly easily and it also allows one to code without worrying about catching exceptions that are specific to each technology.

### 3.3.2 Structure of the application

We talked before about the changes that we have to do in the application, these also affect the structure of the application, with the disappearance of EJBs we have to decide what technology for database access objects to use. JDBC does not require learning another framework’s query language to master. We can more finely tune the performance of your data access when you use JDBC than practically any other technology.

Then, we have the DAO support of the new application, Homesuite uses service objects to connect Controllers with EJBs, we will follow the same path and will seize its code. Finally, the other change is in the MVC principle. Spring provides different kinds of controllers that are more suitable for our solution. As we will see, we will soon use the Controller that are the simplest, and SimpleFormController which adapts to the type of controller that Homesuite needs. The controller receives a form that it will try to interpret and shows a web page with the solution that can be another form.

Finally, the changes in the JSP pages are important as well. Michael Remijan created his own tags for the application and the code of these tags contains Pillar components which we aren’t using. To replace them we use the tags that Spring and J2EE provides us. The new architecture that we use to implement Homesuite is showed by figure 3.9:

![Figure 3.9: Architecture of the Homesuite under Spring](image)

### 3.3.3 Implementation

To explain the implementation, we’ll give you an example of how we have done one of the functions that we have implemented of Homesuite. In this case to modify an account. The rest
of the functions are similar and the explanation can take us to repeat to great extent in the next paragraphs. First, we’ll start by introducing IoC and how Spring uses it to develop Homesuite. Later we’ll go into the explanation of the middle tier of our application. Finally, we’ll move on to explore the presentation layer of the new version: the web.

3.3.3.1 Injection of control

Traditionally, each object is responsible for obtaining its own references to the objects it collaborates with (its dependencies). This implies a lot of code that is hard to test. Applying IoC, objects define their dependencies (i.e. the other objects they work with). Then, it is the job of the container to actually inject those dependencies when it creates the bean. So, IoC means inversion of responsibility with regard to how an object obtains references to collaborating objects.

In the previous chapter, we thoroughly analyzed how the application worked and what objects it used to make each function work. If we are building the function modify account, and following that structure we can declare that AccountController will need dependencies of accountService objects, and accountService from accountDao objects.

To get the account that we want to modify, we define a setting method for that service in accountController, and later we add other setting method for accountDao that gives service the account. This way of IoC is named setter injection, figure 3.9:

```java
47<br:bean id="accountController" class="org.moss.homesuite.controller.AccountController">
48<br:property name="validator" ref bean="accountValidator"/>
49<br:property name="accountService" ref bean="accountService"/>
50<br:property name="formView" value="member/account/edit.jsp"/>
51<br:property name="successView" value="member/account/editThanks.jsp"/>
52</bean>

public class AccountController extends SimpleFormController
47 {
48    AccountServices service = null;
49
50    public void setAccountService(AccountServices account) {
51        this.service = account;
52    }
```

Figure 3.10: Injecting service bean to the controller

The second step is to set up associations between application components. Figure shows the wiring between AccountController - AccountServices, and AccountServices with AccountDao, figure 3.10:
We have created some basic classes that are normal JavaBean with private attributes and public getter/setter-methods and a constructor method. We have used them to store the result of the query that we make to the database. These objects will be used later to manipulate the forms and bean objects that needed the query.

These objects are equivalent to the tables of the database, included all attributes of them. Figure 3.11 shows an example:

```java
47<bean id="accountController" class="org.moss.homesuite.controller.AccountController">
48  <property name="validator">ref bean="accountValidator"</property>
49  <property name="accountService">ref bean="accountService"</property>
50  <property name="accountView">value="/member/account.edit.jsp"/value</property>
51  <property name="successView">value="/member/account.editThanks.jsp"/value</property>
52</bean>
....
91  <bean id="accountService" class="org.moss.homesuite.service.AccountServiceImpl">
92  <property name="accountDao">
93    <ref bean="accountDao"/>
94  </property>
95  </bean>
....
115  <bean id="accountDao" class="org.moss.homesuite.dao.AccountDaoImpl">
116  <property name="jdbcTemplate">
117    <ref bean="jdbcTemplate"/>
118  </property>
119  <property name="diaryDao">
120    <ref bean="diaryDao"/>
121  </property>
122</bean>
```

Figure 3.11: Dependencies between beans

### 3.3.3.2 DataAccessLayer

**DAO support with JDBC**

We have created some basic classes that are normal JavaBean with private attributes and public getter/setter-methods and a constructor method. We have used them to store the result of the query that we make to the database. These objects will be used later to manipulate the forms and bean objects that needed the query.

These objects are equivalent to the tables of the database, included all attributes of them. Figure 3.11 shows an example:

```java
public class Account implements Serializable {
    private String _id;
    private Integer _revision;
    private Date _createdOn;
    private Date _updatedOn;
    private String _displayName;
    private String _accountName;
    private String _password;
    private boolean _hasChanges;

    public Account() {
        super();
        // TODO Auto-generated constructor stub
    }

    public String getDisplayName() {
        return _displayName;
    }

    ...
}
```
DAOs exist to provide a means to read and write data to the database. They should expose this functionality through an interface by which the rest of the application will access them. The data access tier is accessed in a persistence technology-agnostic manner. That is, the data access interface does not expose what technology it is using to access the data. Instead, only the relevant data access methods are exposed. This allows a flexible application design. If the implementation detail of the data access tier was allowed to leak into others parts of the application, the entire application becomes coupled with the data access, leading to a rigid application design.

```
public interface AccountDao {

    public abstract Account insert(AccountInsert cmd);
    public abstract Account FindByAccountName(AccountGet get);
    public abstract Account update(AccountUpdate cmd);
    public abstract Account findByPrimaryKey(String id);
}
```

Figure 3.13: DAO interface classe

In the implementation class of AccountDao, we will use a Spring template to access the database. A spring template keeps us away from dealing with JDBC datasources and connections.

**Datasource**

In order to execute any JDBC operation on a database, we need a Connection. In Spring’s DAO frameworks, Connection objects are obtained through a DataSource. This is configured as a bean in the application context, given a service to a prepared template. We will use the DriverManagerDataSource implementation for the connection but there are several additional implementations like JNDI. The DriverManagerDataSource works the same way as Homesuite used to work when it obtains a JDBC connection. Figure 3.13 shows how DataSource is configured:
JDBCTemplate

We always need to obtain a connection to our data store and clean up resources when we are done. These are the fixed steps in a data access process. But each data access implementation we write is slightly different. We query for different objects and update the data in different ways. These are the variable steps in a data access process.

Spring separates the fixed and variant parts of the data access process into two distinct classes: templates and callbacks. Templates manage the fixed part of the process while callbacks are where you fill in the implementation details.

The JdbcTemplate is a template class that Spring’s data access incorporates. We only need one JdbcTemplate instance for each DataSource in our application. To make use of the JdbcTemplate, each of our DAO classes needs to be configured with a JdbcTemplate instance adding a private attribute with the type JdbcTemplate and generate the setter method.

JdbcTemplate executes SQL queries, update statements or stored procedure calls, imitating iteration over ResultSets and extraction of the returned parameter values. For example, it provides an execute (String sql, Object [ ] params) method that creates a PreparedStatementCreator and PreparedStatementSetter. We just supply the SQL and the parameters. Also, we can make the JdbcTemplate method accept the JDBC types of our parameters.

When we queried the database we had to iterate through the ResultSet. Spring recognizes that this is a step that is always required for queries, so it handles that for us. Instead, we simply need to tell Spring what to do with each row in the ResultSet. Spring provides an implementation of this class that does exactly what we need: RowMapper.

The RowMapper interface is responsible for mapping a ResultSet row to an object. To map a row of Account object, we would create a RowMapper like the figure 3.14:
Chapter 3: Case of study: Homesuite application

3.3.3.3 Service Layer

In the new Homesuite, service objects and commands fulfil the same function as in Homesuite. The service objects communicate MVC tier with data access tier. The difference being that service objects of Homesuite called to session beans, while the new service objects call to DAO objects. Commands are the same as in Homesuite, and they are entrusted of giving dates, format for DAO objects could work with them.

3.3.3.4 Presentation Layer

Spring’s web module provides a clear separation of roles: controller, validator, command object, form object, model object, DispatcherServlet, handler mapping, view resolver, etc. Each role can be fulfilled by a specialized object. Our intention in this point of the implementation, is to complete the web layer of the new Homesuite version, taking the components that Spring gives us. Selecting the correct controller and working with validators can help us to make an application whose components will be more independent and tidier. In the JSP pages, we have changed most of the tags that Homesuite had. Some of them had been created under Pillar framework avoiding to know components and features that Spring provides us.

DispatcherServlet

Unlike we explained in the second chapter, Spring MVC provides a servlet which delegates responsibility for a request to other components of the application, like controllers or handler

```java
@public class accountRowMapper implements RowMapper {
    @public Object mapRow(ResultSet rs, int rownum) throws SQLException {
        // I use JDK 5 so I do not have to wrap int with an Integer object
        account account = new Account();
        account.setId(rs.getString("id"));
        account.setAccountName(rs.getString("accountName"));
        account.setDisplayName(rs.getString("displayName"));
        account.setPassword(rs.getString("password"));
        account.setRevision(new Integer(rs.getInt("revision")));
        return account;
    }
}
```

Figure 3.15: implementing accountRowMapper

We now have a reusable class that can take a ResultSet row and create a Account object.
mappings, to perform the actual processing. The dispatcherServlet is this servlet and it is configured in the web application’s web.xml file as the figure 3.15:

```
17  <servlet>
18      <servlet-name>homesuite</servlet-name>
19      <servlet-class>
20          org.springframework.web.servlet.DispatcherServlet
21      </servlet-class>
22      <load-on-startup>2</load-on-startup>
23  </servlet>
24
25  <servlet-mapping>
26      <servlet-name>homesuite</servlet-name>
27      <url-pattern>*.htm</url-pattern>
28  </servlet-mapping>
29
```

Figure 3.16: DispatcherServlet configuration

Our application context will be loaded from an XML file whose name is based in the written noun on <servlet-name>. The DispatcherServlet will try to load the application context from the file named homesuite-servlet.xml. <servlet-mapping> tag indicates which URLs will be handled by the DispatcherServlet. In our case we will use “.htm” because the content produced by our application is HTML.

Mapping request to controllers

When a request comes in, the DispatcherServlet will hand it over to the handler mapping to let it inspect and associate the request with a specific controller. There are several handler mapping implementations and we selected SimpleUrlHandlerMapping. It lets you map URL patterns, associating each of them to a controller. The figure 3.16 shows the HandlerMapping:

```
61  <bean id="urlMapping" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
62      <property name="mappings">
63          <map>
64              <entry key="/register.htm">registerController</entry>
65              <entry key="/login.htm">loginController</entry>
66              <entry key="/account_edit.htm">accountController</entry>
67              <entry key="/logout.htm">logoutController</entry>
68              <entry key="/member/contact/contact_save.htm">contactController</entry>
69              <entry key="/contactView.htm">contactViewController</entry>
70          </map>
71      </property>
```
Controller
Spring has implemented the notion of a controller in a very abstract way enabling a wide variety of different kinds of controllers to be created. Spring contains form-specific controllers, command-based controllers, and controllers that execute wizard-style logic, to name but a few.

Pillar framework doesn’t have that rich hierarchy that Spring provides. Homesuite is a web application whose JSP pages are forms that users must fill out. When the user submits the form, the data that he enters is sent to the server for processing. With the class SimpleFormController, once the processing is completed, the application can present a successful page or return the form page with the errors that user must correct. On other occasions, we don’t need to manage database or perhaps we read it, for these case we use Controller interface.

a) SimpleFormController
First of all, we implement the function formBackingObject to data from the given request. We’ll work with this data later to check with the new data entered. Figure 3.17 shows the implementation of this method for the AccountController

```java
protected Object formBackingObject(HttpServletRequest request) throws Exception {
    AccountBean bean = (AccountBean) WebUtils.getSessionAttribute(request, "userSession");
    if (bean != null) {
        return new AccountForm(this.service.findByEmail(bean.getEmail()));
    } else {
        return new AccountForm();
    }
}
```

Figure 3.18: formBackingObject method of AccountController

OnSubmit () is the method that handles the form submission by passing the form object to the service layer. That method, figure 3.18, returns a ModelAndView object where we send to DispatcherServlet what page and model (data) will be shown.
If we look at AccountController, we have tried to continue the original structure of Homesuite. That is, we use the same method names that Michael Remijan uses in his controllers, and they are called in the same order, service(), update() and attemptAccountUpdate(). So, it’s easier to compare both applications in later comparisons so we call in onSubmit the method service().

SimpleFormController is designed to keep view details out of the controllers’ Java code as much as possible. Instead of hard-coding a ModelAndView object, we configure our controller in the context configuration file as follows:

```java
private final ModelAndView attemptAccountUpdate(HttpServletRequest request, HttpServletResponse response, HttpSession session)
    throws ServletException, IOException, Exception {
    try {
        account = service.update(account, form);
    }
    catch (DatabaseException e) {
        request.getSession().setAttribute("userService", account);
        ModelAndView = super.onSubmit(request, response, form, errors);
        // Finally, cleanup and return
        service = null;
        account = null;
        return ModelAndView;
    }
```

Figure 3.19: attemptAccountUpdate() method of AccountController

In the figure 3.19, the validator property wires with a validator bean, we will explain that later. AccountService is the injected service reference. The formView property is the logical name of a view to display when the controller receives an HTTP GET request or when any errors are encountered. Likewise, the successView is the logical name of a view to display when the form has been submitted successfully.

We have implemented validators when controllers needed to check if data is valid before inserting or modifying them into the Database (LoginController, RegisterController,
AccountController). These components examine the fields of the object passed into the validate() method and reject any invalid values via the Errors object.

This method is called by the controller inside the method onBindAndValidate(). When we try to modify an account, if all of the required properties are set and the name, email, new password and current password are valid, then AccountController’s doSubmit() is called and the account is changed. However, if AccountValidator rejects any of the fields, then the user will be returned to the form view to correct the errors.

b) Controller interface

Sometimes, we must read information of the database or we don’t need to work with database layers, we use the controller interface. It is simple and abstract, defines a single method that is responsible for handling a request and returning an appropriate model and view. LogoutController, figure 3.20, is one of these basic controllers that we have been making.

```java
public class LogoutController implements Controller
{
    ...

    public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response) throws Exception {
        request.getSession().removeAttribute("userSession");
        return new ModelAndView("index.jsp”);
    }
}
```

Figure 3.21: LogoutController implementation

On that, we simply remove the content of the session attribute “userSession” and show the JSP page “index.jsp”, the main page of the Homesuite website.

Creation of JSPs

As soon as we have handled requests and forwarded them to JSPs, we must to implement these JSPs to access the model in order to display it. We will make use of JSP Standard Tag Library (JSTL) and a set of data binding-aware tags, which Spring provides. With each tag we’ll get our command objects, all of their properties, and any error messages associated with these properties as well. We haven’t worked with the tag libraries that Homesuite used because, JSTL and Spring libraries, have tags that can replace the others.

We were building the JSPs through the Homesuite JSPs, although we had to change large part of them, the files helped us like a schema which we could follow.
First of all, we must bind the `<input>`, figure 3.21, form elements to command objects, that are used together with the form. We work with the `<spring-bind>` tag, which is used to access command objects and any error messages associated with them. This tag has only one attribute – `path-` that indicates the bean or bean property being used. For example, following with the implementation of modify account, we have the JSP “account.edit.jsp” where we have to access to the properties of the AccountForm object, we do it by setting the path attribute to `AccountForm.property_name`. This is made available through a org.springframework.web.servlet.support.BindStatus object that is placed in the page scope with the name `status`. The `${status.errorMessage}` and `${status.value}` are special variables declared by the framework that can be used to display error messages and the current value of the field. Furthermore with `${status.expression}` we set the name of our form input tag.

```html
<td><input type="text" name="account.property_name" value="${status.value}" /></td>
</tr>
</table>
</body>
</html>
```

Figure 3.22: JSP code to take data

The previous figure is shown as we access the email property of a accountForm object, with that, Spring will automatically be able to map the form input field with it current value to our accountForm object when the form is submitted.

```html
<spring:hasBindErrors name="accountForm">
  <spring:bind path="accountForm.*">
    <div class="errorBox">Please correct the following error then try logging in again:</div>
    <ul class="errorList">
      <li class="errorItem">${error}</li>
    </ul>
  </spring:bind>
</spring:hasBindErrors>
```

Figure3.23: code which shows errors
When users do something wrong we have the errorMessages property, figure 3.22. This property binds to the command object and displays all error messages associated with this object. The following figure shows us how this was done.

### 3.3.3.5 Security in Homesuite

Homesuite uses HTTPS protocol for the communication between client and server. It is used when the connection between client and server needs to be secure. The information is encrypted and sent to the other side. Spring provides support for this protocol with the package of Acegi Security System. This is a security framework that uses servlet filters to intercept servlet request and perform authentication and enforce security.

In this case, we needed that our application used HTTPS protocol, for this Acegi provides Channel-processing filter, figure 3.23, which ensures that requests are transmitted over a secure channel.

First of all we configured this filter in the web application’s adding a FilterToBeanProxy configuration in the web application’s web xml. Thus, the filter can be used like a bean in the Spring configuration file. Here, we configured the filter, indicating what pages would be encrypted, in our case all pages. The figure shows this configuration.

```xml
<bean id="channelProcessingFilter" class="org.acegisecurity.securechannel.ChannelProcessingFilter">
  <property name="filterInvocationDefinitionSource">
    <value>
      CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON\n      \#SECRET\n      \#SECURITY_CHANNEL
    </value>
  </property>
</bean>
```

Figure 3.24: channelProcessingFilter configuration

Finally, the bean ChannelDecisionManager provides the channel processor which is entrusted with redirects the request a secure form if it have been required.

### 3.4 Sumary

In this chapter we have research the web application Homesuite. Between the properties of this J2EE application. It is noteworthy, its presentation tier implementation under Pillar Framework and the use of EJB container in the persistence tier.
The development of the Homesuite under Spring shows us that Spring framework can adapt to any kind of web application easily. The Spring modules gives us wide variety of components that make easy to implement this application. Concepts like Inversion of control, POJO and ModelAndView help to convert the code of Homesuite into tidier and easier to understand code.

Some of the changes that Homesuite have undergone with its implementation under Spring affect web application’s web.xml and web container like we explained in the section first steps, and persistence layer where EJB were substituted EJBs by DAO components. These components are POJO whose code can be reused easier than EJB.
Chapter 4

Homesuite’s Performance

Nowadays, the commercialization of an application depends on the result of a performance test. Analyzing performance and ensuring the application functions are carried out correctly, allows us to understand the application better, including both its properties and limitations. The quality of the test is important to check that an application passes every restriction and measures the efficiency and throughput of the product.

In the past, for performance tests it was necessary to build a test infrastructures what helped to ensure tight security and optimum application performance, but it was not always efficient and besides, they were expensive and cumbersome.

Avalanche is a load testing appliance that provides a scenario to carry out the test close to the reality. It can simulate the user behaviour, that is, the system can interact with websites using dynamic and interactive content, HTML links and sending requests that include filled-in forms, or values captured from a previous response such as order numbers, session IDs or transactions IDs. Avalanche evaluates different types of browser such as SSL versions, authentication and browser client headers.

Avalanche provides two types of test, those that are entrusted with Infrastructure/ Device testing such as Routers, gateways, etc. and those entrusted with performance server testing. The server testing is used to measure the efficiency and throughput of web applications and other systems under protocols (HTTP/HTTPS, POP3, FTP, DNS). It supports all type of traffic, even the combination of video, voice and data networks.

Avalanche has a wide variety of load variables such as user sessions, new user session/second, transactions, transactions/second, connections or connections/second. One load profile can be used for different groups of simulated users. Each group of simulated groups can perform different actions and work with different network characteristic.

4.1 Testing

Homesuite is a web application that offers multiple services like email, bookmark, contacts management and diary, to every registered user in Homesuite. This portal was created by Michael Remijan and is structured in three tier. The web layer is implemented under Pillar
Framework while the business logic is made up by enterprise services (EJB). All within the J2EE platform context.

The new version of Homesuite under Spring Framework tries to take advantage of its features such as Inversion of Control, DAO support and its MVC framework. So, the development process of Spring version is different from the original version and possibly performance evaluations, too. The new version of Homesuite has two variants: using SSL protocol (HTTPS), and without SSL protocol.

In this chapter we tested Spring version of Homesuite. First, without SSL encrypted protocol to know the number of SimUsers/second that can use the system without problems in the received transactions.

We studied the throughput of this version, which indicates the maximum number of transactions/second that the system can support without HTTP errors.

The second one evaluated the throughput of both Homesuite versions using Avalanche Commander and with SSL protocol. Thus, we would compare Spring and J2EE at performance level and, would know which technology, Spring or J2EE, has more efficient components.

In both tests lie in simulating the behaviour of multiple users that access to the portal and do the action sequence login, access to contacts and logout.

4.2 Homesuite’s performance

Avalanche Commander was configured to simulate the access of a number of registered users to the web application, at the same time. The appliance supports HTTPS protocol, so we would not have problems with SSL-encrypted traffic used by Homesuite. The purpose of this test scenario is to show and validate the Spring version of Homesuite. Besides of facilitating the programming of that application to the developers, it also obtains good results in the performance evaluations.

For the evaluation of Spring version of Homesuite without SSL protocol, we performed multiple test by changing the number of simulated users to trace/differentiate the with or without HTTP errors.

For that evaluation of both technologies with SSL protocol, each application, Spring version and original version, has to do two tests. These evaluate the resulting throughput (transactions/second) when we apply a scenario to an application, in which each certain time
(11) the number of simulated users is increased by one. The first test increases this number until 11 simulated users, and the second increases it to 6.

4.3 Analisys

J2EE was created to develop distributed applications. Those applications typically have a web front end and a relational database backend. The increasing development of new web application has made it the most important web application platform.

However, that caused applications to be more complex and widened the range of enterprise use cases, pointing out the deficiencies and complexities that J2EE specification had. This favoured the creation of new frameworks that make application development much easier and faster. One of these frameworks is Spring also named lightweight framework as a variant of the very rigid structure of EJB 2.1.

The core of Spring is the inversion of control and the use of POJO. The first feature allows to coordinate the system objects and the second one is used as service components. At development level, Spring application looks better than J2EE application but, will a Spring application get in performance evaluations better or similar results than a J2EE application? Or are those results worse?

4.4 Test configuration and methodology

To test the performance of the two applications, we connected layer 4-7 of the Spirent Communication Avalanche 2500 to asklepios134.test server, which has the following properties: AMD Athlon XP 1600 (1.4GHz) 256MB RAM Debian Linux with kernel 2.6.11 and 100MBit, is the machine where deployed applications reside. These layers generate realistic traffic. Figure 4.1 shows the connection between these machines.

Lastly, a PC running Windows XP and Avalanche Commander Software which simulates and captures the performance of the Homesuite (original and new versions).
4.4.1 Load profile

To simulate the application first, we specified the load profile of the performance, in our case simulated users per second that indicates that an effective user is opening several web pages. After which, we decided on the traffic of simulated users that would access the Homesuite. On the one hand, it was the evaluation of Spring version of Homesuite, the number of simulated users were changed until finding the test without HTTP errors. The evaluation of Spring and J2EE versions have two tests each one, with 11 and 6 Simulated Users/second respectively.

For each test we have kept the general load graph in 5 stages. To continue with the definitions of each stage, we will be able to give an example of the performed tests:

1. Delay: Time for the setup to exchange ARP – message. Steady time: 30 seconds
2. Ramp up: Gradually increase load. With a Ramp time of 30 seconds
3. Stair step: increasing a SimUser each 11 seconds, this process is repeated 10 times.
4. Steady state: keeping the state of 11 SimUsers during a steady time of 60 seconds
5. Ramp down: Process the data that have not finished before the test ends. Steady Time: 60 seconds

The following figure 4.1 shows the load graphs used to simulate the two applications.
The figure 4.2 shows that the second load graph is similar to the first one, the only difference being that stair step stage changes where the number of repetitions is 5.

**4.4.2 Action List**

Later, we created two action lists, one for Homesuite and another for the new version of Homesuite. The action list is the set of actions that each simulated user will do. This list contains the tidy sequence of HTTP headers of each application opened page. We have to differentiate between actions preceded with 1 or 2. The first one indicates that the user is clicking a link, while the second one indicates the images automatically download from the web page.

The main actions were access to Homesuite portal, login, access to contacts and logout. One of them, login users have to insert their email account and password.
In contrast to Homesuite original application, Spring version doesn’t need that the images are downloaded from the web page each time that an user clicks in a link have been. Once the images are downloaded in the cache, are used these. As a result Spring version carries out less actions in each action list.

### 4.4.3 User profile

Then we configured the user profile. There are three important fields:

- User think time, time between two “1-command” actions, in our case 4 seconds.
- Abort, number of connections (%) that are aborted, 0% in our case.
- Time before abort, time taken before abort, 0 also in our case.

Furthermore, there must be an associated UserForm database, because the login action demands an email account and a password. UseForm contains a list of user’s email and password which are registered in the database.

### 4.4.4 Ports and associations

We created a subnet profile with the IP address 192.168.01-192.168.0.127 (asklepios134.test) and network 192.168.0.0.

Finally, we associated the different profiles as it is showed by figure 4.4:

![Figure 4.4: associations between profiles that perform the application](image)

### 4.5 Results and Evaluation

The tests with Spring version of Homesuite without HTTPS protocol, indicates the maximum number of simulated users per second that this system can be supported without HTTP errors is 22. Furthermore, when the performance has used more than 22 SimUsers/second, the throughput exceeded 600 transactions/second and the system returns some HTTP errors. So, the maximum throughput that can be about 600 transactions/second, figure 4.5.
In order to ease the interpretation of the following graphs, it is worth mentioning that time axis is scaled (1:4), which means that time 10 in the graph is equivalent to time 40 seconds.

![Performance with HTTP](image)

**Figure 4.5: Throughput of Spring version with 22 SimUsers/second**

With regard to performance of Spring and original version of Homesuite, we took the application throughput until the second 124, figures 4.6 and 4.7, because after that moment extreme changes in the traffic of the server hindered the rest of the test of 11 simulated users. As it can see in the following graphs, the results are similar, what they point out which the two applications has equal efficiency. The figures 4.8 and 4.9 shows the same performance but for 6 SimUsers/second

Another important result that was produced, is Spring version carries out less transactions/second in the same time as original version. That is because each time which a user interacts with the application Spring version has to do less actions than original version, then the Spring version produces less traffic in the network.
Figure 4.6: Performance of original Homesuite for 11 SimUsers/second

Figure 4.7: Performance of Spring Homesuite for 11 SimUsers/second
Finally, the results of the performance using HTTP and HTTPS are not comparable. Although both of them uses similar protocols, it is important to remember that HTTPS is an encrypted protocol where the server must establish a secure connection with a client and passes on encrypted information between them. Thus, HTTPS protocol generates more network traffic than HTTP protocol and the server has to attend more transactions per second for the same kind of service.
Chapter 5
Conclusions

In the nineties, the content of the first websites were simple and easy. With the increasing
demand of applications with more complicated services, the development was more costly
and difficult to maintain. The code turned complex and difficult to understand. With the birth
of J2EE, it revolutionalized the way of developing applications because it introduced concepts
like distributed multitier applications, enterprise beans, JSP and servlet in the web application
environments.

The distributed multitier application was quickly adopted by developers and in a few
years, J2EE standard was the technology most extended in the enterprise applications world.
J2EE provides three-tier applications because its components are in three locations: client
machine, J2EE server machine and DataServer machine. The application client is executed in
the client machine. In the web machine, there are two containers: the Web container and the
EJB container.

The Web container provides JSP where the web content or user’s interface is created.
Servlets are entrusted with giving service to web content, like interacting with the enterprise
beans or accessing the database.

The J2EE container provides the enterprise beans that implement the business logic of the
application. There are three kinds of enterprise JavaBeans: session bean which executes a
business task demanded by the client, entity bean which is entrusted with managing the
database and, message-driven bean which processes messages asynchronously.

The database is localized in the DataServer machine where the data is stored.

In the last few years, there has been an increasing popularity of the frameworks. These
normally use the power of J2EE and substitute the deficiencies that J2EE has in the
implementation. Frameworks like Spring have made it easier for developers to build
applications. Spring is a lightweight framework whose features are based on five concepts:
Inversion of Control, POJO (Plain Old Java Objects), AOP (aspect-oriented programming),
MVC (ModelAndView) and integration.

- **Inversion of Control** is when the container figures out that the component needs a
certain object and provides it, without the need of the component to look up references
of the object.
• **POJO** helps in the code reuse.

• **AOP** is used when the functionality of an object is applied in another place and with other properties.

• It provides a **MVC framework** which separates the business logic (model) from the web content (view). The DispatcherServlet is the heart of this container and it is entrusted with managing the handling of the requests which the client sends to it. An important component of this framework is the controller which performs business functionality and interacts with the persistence tier of the application.

• **Integration**: Spring provides support to integrate other persistence tier tools like Object/Relational Mapper (Hibernate, Apache OJB and iBATIS), persistence API JDO and enterprise beans or, it can work with other frameworks like Struts, Tapestry, WebWork and JavaServer Faces.

The problem encountered with the Spring framework like the rest of the frameworks, is the cost of required time to learn a new way to develop web applications. For software enterprise, it is a daring bet, changing the framework that it uses because it changes the work methods that can become obsolete in few months besides spending time and money.

A practical example of the web application development for J2EE and Spring technologies is Homesuite. It is a portal which offers some user services like email, contacts manager, calendar, diary and bookmark.

Homesuite was developed under J2EE technology. It is structured in three layers: web layer which is build with Pillar Framework, service layer and the persistence layer which is formed by enterprise beans. Pillar framework has similar features to Struts. It emphasizes on the use of a properties file where the relationship between web components of application (JSP and controllers) and request URL’s pattern is found. This way, servlet gives the request to the corresponding web component.

Implemented functions Homesuite application like register, log on, insert contact and send an email, are implemented by the same procedure. The controller receives the request and calls service instance which is entrusted with providing the data to session bean which will modify the database. Session bean uses an entity bean to manage the database. Later, the session bean converts the entity bean in a special POJO which is sent to the service instance. This converts this POJO into another one that can be used by the controller to send with the request.
For its part, Spring version of Homesuite maintains the same structure as J2EE Homesuite, but takes advantages of some Spring features like Inversion of Control, POJO and MVC framework.

Thus, the configuration of the application and of the components is adapted to the concept of inversion of control. In the presentation layer, in addition to tags libraries of JSP, Pillar controllers are replaced by Spring controllers. The enterprise beans are changed by DAO components (Spring JDBC support) in the persistence layer and the service components are adapted to the new components.

Spring version of Homesuite was implemented with the idea of using the highest number of its features and enlarge view of J2EE and Spring theoretically were given. The support that Spring provides, makes easy the applications development because it eliminates the implementation of fixed (EJB context, database connection). The inversion of control also contributes to the code of Spring version turns out tidier and easier to understand than J2EE version. The use of POJO allows developers to reuse code. All of that entails that the dedicated time to implement a Spring application is reduced and the throughput with regard to J2EE application.

The results which are obtained from performing these Homesuite versions show that the two applications have equal efficiency. However, it is verified which Spring application loads less traffic network, that is, it produces less transactions than Homesuite with the same simulated users, per second because its application doesn’t always need to download all images from the websites.
References

[1] J2EE 1.4 Tutorial
   Eric Armstrong, Jennifer Bal, Stephanie Bodoff, Debbie Bode Carson, Ian Evans, Dale Green, Kim Haase, Eric Jendrock, December 2005
   mei 1997

[2] Spring, java/j2ee application framework
   Rod Johnson, Juergen Hoeller, Alef Arendsen, Colin Sampaleanu, Rob Harrop, Thomas Risberg, Darren Davison, Dmitriy Kopylenko, Mark Pollack, ThierryTemplier, Erwin Vervaet, Portia Tung, Ben Hale, Adrian Colyer, John Lewis, Costin Leau, Rick Evans

[3] Spring in Action
   Manning
   Craig Walls, Ryan Breidenbach

Diverse internet websites
