Masaryk University
Faculty of Informatics

Online Shop Web Tier in Java EE

Bachelor Thesis

Martin Janík

Brno, May 2007

Declaration

Hereby I declare, that this paper is my original authorial work, which I have worked out by my own. All sources, references and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Advisor: Ing. Petr Adámek

Acknowledgement

I am very grateful to my advisor, Ing. Petr Adámek, for his support and guidance throughout my work on this thesis. I do appreciate his willingness to help me solve all the problems I had struggling with the Java Enterprise Edition platform and the Sun Java System Application Server I used for development and deployment.

Many thanks also belong to my family and friends who supported me while working on this thesis. It would have been very difficult without their support.

Abstract

This thesis is mainly focused on the Java Enterprise Edition 5 platform and the web application frameworks based on this platform. Brief introduction to the Java EE 5 platform helps understand the concepts used by the modern Java web application frameworks and other powerful tools used by Java web application developers. Some of the most widely used request-based and component-based Java web application frameworks are described in detail, as well as some other tools and approaches to Java web application development. Based on the specified requirements, a conceptual design of an online shop web tier in Java EE is proposed. Finally, a reference implementation of the online shop web tier following the proposed conceptual design is provided.

Keywords

Java Enterprise Edition, Sun Java System Application Server, Java Servlet API, JavaServer Pages, JavaServer Faces, Enterprise Java Beans, Java Persistence API, Java Web Application Frameworks, Model-View-Controller, Representational State Transfer, Online Shop Web Tier

Contents

1 **Introduction** . . . 1
1.1 *PatroNG Project* . . . 1
1.2 *Thesis Structure* . . . 1
2 **Java Enterprise Edition 5** . . . 3
2.1 *Web Applications* . . . 4
2.1.1 Java Servlet Technology . . . 5
2.1.1.1 Servlet Life Cycle . . . 5
2.1.1.2 Sharing Information . . . 6
2.1.1.3 Filtering Requests and Responses . . . 6
2.1.2 JavaServer Pages Technology . . . 7
2.1.2.1 JSP Page Life Cycle . . . 7
2.1.2.2 Unified Expression Language . . . 7
2.1.2.3 JavaServer Pages Standard Tag Library . . . 8
2.1.2.4 Custom Tags in JSP Pages . . . 8
2.1.2.5 Scripting in JSP Pages . . . 9
2.1.3 JavaServer Faces Technology . . . 9
2.1.3.1 User Interface Component Model . . . 9
2.1.3.2 Navigation Model . . . 10
2.1.3.3 JSF Page Life Cycle . . . 11
2.1.3.4 Custom JSF Components . . . 11
2.1.4 Internationalization and Localization . . . 12
2.1.4.1 Localization in Java . . . 12
2.1.4.2 Date and Number Formatting . . . 12
2.1.4.3 Character Sets and Encodings . . . 13
2.2 *Web Services* . . . 13
2.3 *Enterprise JavaBeans* . . . 13
2.3.1 Session Beans . . . 14
2.3.2 Message-Driven Beans . . . 14
2.3.3 Entity Beans . . . 14
2.4 *Java Persistence API* . . . 14
2.4.1 Persistence in the Web Tier . . . 15
2.4.2 Persistence in the EJB Tier . . . 15
2.4.3 Java Persistence Query Language . . . 15
2.5 *Services* . . . 16
3 **Java Web Application Frameworks** . . . 17
3.1 *Model-View-Controller* . . . 17
3.2 *Representational State Transfer* . . . 18
3.3 *Request-Based Frameworks* . . . 18
3.3.1 Apache Struts . . . 19
3.3.2 WebWork . . . 19

3.3.3 Apache Struts 2 20
3.3.4 Stripes 20
3.3.5 Spring MVC 21
3.4 *Component-Based Frameworks* 21
3.4.1 JavaServer Faces 21
3.4.2 Tapestry 22
3.5 *Related Technologies* 22
3.5.1 UrlRewriteFilter 22
3.6 *Evaluation* 23
4 **Online Shop Web Tier** 24
4.1 *Analysis* 24
4.1.1 Customer's Perspective 25
4.1.2 Web Designer's Perspective 25
4.1.3 Programmer's Perspective 25
4.2 *Specification of Requirements* 25
4.2.1 Customer's Perspective 25
4.2.2 Web Developer's Perspective 26
4.3 *Conceptual Design* 27
4.3.1 Presentation Technologies 27
4.3.2 Web Application Framework 28
4.3.3 View Technologies 28
4.3.4 Client-Side Technologies 28
4.3.5 Shopping Cart 29
4.3.6 Internationalization 29
4.3.7 Security 29
5 **Reference Implementation** 30
5.1 *Customer Catalogue* 30
5.2 *Product Catalogue* 30
5.3 *Shopping Cart* 30
5.4 *Web Administration* 31
5.5 *Web Tier* 31
6 **Conclusion** 32
Bibliography 33
A **Contents of Attached CD** 34

Chapter 1

Introduction

Online shopping has become very popular in the last few years. There are hundreds of online shops available all around the Internet. Customers can purchase music, books, movies, games, electronics, perfumes, jewellery, clothing and many other products while sitting comfortably at their home PC. They do not need to spend the whole afternoon running around the city, they can simply access many of these shops online. When they do not like the product portfolio of a particular online shop or the products are too expensive, there are plenty other online shops available, just one click away.

Companies running such online shops try really hard to provide the best services for all their customers. Often a single company sells many different kinds of products worldwide. This requires robust, scalable and secure applications for the company itself to manage all the business processes. The online shop is only one of them. In addition, companies usually need these enterprise applications quickly and require ongoing support and maintenance. Therefore several platforms for developing such applications have emerged, Java Enterprise Edition 5 being one of them.

1.1 PatroNG Project

The PatroNG project is devoted to the development of a new generation online shop based on the latest Java technologies. The main objective of this project is to study technologies provided by the Java Enterprise Edition 5 Platform and use them to reimplement a particular online shop. Since the company running this online shop sells products of almost every kind, the new implementation should provide a sophisticated infrastructure and a framework for creating additional online shops.

The project is being solved by several students studying at the Faculty of Informatics at Masaryk University. Some of them, including me, focus on the technologies and approaches that should be used for the development, others are interested in usability, search engines, data import/export and other topics related to the online shops.

1.2 Thesis Structure

This thesis is divided into four main chapters, not including introduction and conclusion. All chapters expect general knowledge of web-related technologies, such as HTTP, HTML, XML, CSS, JavaScript and AJAX.

The chapter called *Java Enterprise Edition 5* provides a brief introduction into the Java Enterprise Edition 5 platform and its main features. Due to the objective of this thesis, web tier related technologies are covered in the greatest detail.

In the next chapter titled *Java Web Application Frameworks*, basic concepts of web application frameworks based on the Java EE platform are introduced. Some of the most widely used request-based and component-based Java web application frameworks are presented and briefly compared. Some other tools and approaches to the Java web application development are described as well.

The following chapter titled *Online Shop Web Tier Concept* includes a brief analysis of an online shop. This analysis is then used to specify the requirements and later propose a conceptual design of the online shop web tier in Java EE.

Finally, in the last chapter called *Reference Implementation*, a simple implementation of the online shop web tier is presented. The implementation follows the conceptual design proposed earlier.

Chapter 2

Java Enterprise Edition 5

The Java Enterprise Edition (Java EE) is a platform for developing portable, scalable, transactional and secure server-side applications in Java programming language. Based on the Java Standard Edition (Java SE), Java EE provides a powerful set of APIs for developing web applications, web services and distributed multi-tier applications. The main aim of the Java EE 5 platform is to provide developers with an infrastructure reducing resources and effort required to develop such enterprise applications.

The Java EE 5 platform takes advantage of the Java annotations, introduced in the Java SE, providing a simplified programming model. This approach eliminates the need for annoying deployment descriptors, although they can still be used, by allowing the developer to place the configuration data directly into the affected Java source code. This way all the information needed by the Java EE server to successfully load and configure a component can be found in one place, making it easy to keep them synchronized.

The Inversion of Control (IoC) principle, also known as Dependency Injection, is supported in the Java EE 5 platform. Java EE components can use annotations to request resources, such as database connections, that are automatically injected by the Java EE container at deployment or at runtime. This approach effectively hides the lookup and creation of required resources from the developer, reducing the amount of boilerplate code written. This approach can be used in EJB containers, web containers and application clients.

The Enterprise JavaBeans (EJB) 3.0 component architecture has been introduced in the Java EE 5 platform. This version of the EJB delivers significant simplification in development, mainly by using annotations instead of deployment descriptors, Plain Old Java Object (POJO) entities instead of Entity EJBs, and the Dependency Injection. EJB components are used primarily to encapsulate the business logic of distributed multi-tier applications and can be accessed remotely from other enterprise applications.

New to the Java EE 5 platform is the Java Persistence API, a part of the EJB 3.0 specification. This API provides an Object-Relational Mapping (ORM) for managing the relational data within applications built on the Java EE or SE platforms. The main idea behind this concept is to provide a powerful but yet easy-to-use ORM tool. Like EJB 3.0, the Java Persistence API uses the Java annotations for configuring entities and their relationships.

The Java Enterprise Edition 5 platform is further described in the following sections, and perfectly covered in [1]. This chapter assumes a good knowledge of the Java programming language and corresponding programming experience. The reader should also be familiar with relational databases and related topics.

2.1 Web Applications

In the Java EE platform, a web application runs in a web container or an application server. There are two types of web applications – presentation-oriented and service-oriented.

Presentation-oriented web applications present static content or dynamic content generated in response to a client's request. Server's response may represent an HTML web page, an XML document or an image that is displayed in the client's web browser. Web designers can use HTML web pages to create a user interface, web forms for processing user input data, and hyperlinks for navigation between the pages. Such user interface can be then enhanced by attaching a graphic layout or by employing client-side scripting.

Service-oriented web applications, on the other hand, provide a web service for the client, often represented by a presentation-oriented web application. Web services are briefly covered in the next section of this chapter.

In the Java EE platform, a web application is composed of web components implemented as Java servlets. Every HTTP request sent by the client is translated into a request object implementing the `HTTPServletRequest` interface. The request object is then passed to the web component responsible for handling the request. The web component can process the request and generate a response object implementing the `HTTPServletResponse` interface, or simply forward the request to another web component. This process continues until one of the web components processes the request, generating a response object. The response object is then translated into an HTTP response and sent back to the client.

Java servlets can be represented as Java classes or as text-based JavaServer Pages (JSP) executed as servlets at runtime. While the servlet classes provide a low-level approach to handling HTTP requests, JSP pages provide a more natural approach to creating textual content. Additionally, it is possible to use predefined and custom JSP tags providing application-specific functionality in JSP pages. Therefore the latter approach is more suitable for web designers with little or no programming experience.

The JavaServer Faces (JSF) Technology has been developed, based on the three mentioned technologies. JSF provide a mechanism for creating rich web components that are rendered to the client. Each of these technologies is described in a separate section.

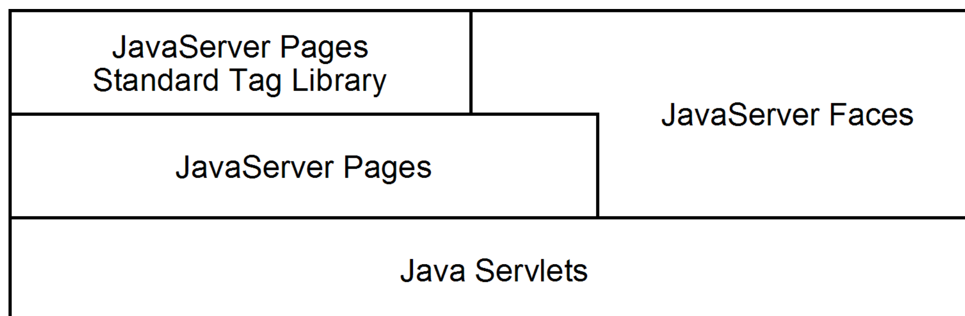

Figure 2.1: Java Web Application Technologies

2.1.1 Java Servlet Technology

A *servlet* is a Java class capable of responding to any type of request via the request-response programming model. Servlets are mainly hosted by web servers to handle HTTP requests.

Interfaces and classes required for writing servlets are located in `javax.servlet` and `javax.servlet.http` packages. A servlet is any class implementing the `Servlet` interface, usually extending either the `HttpServlet` or `GenericServlet` class.

We will focus on servlets responding only to HTTP requests.

2.1.1.1 Servlet Life Cycle

At the deployment time, each servlet is mapped to a certain URL pattern specified in the application's deployment descriptor, called `web.xml`. When an incoming HTTP request matches any of these patterns, a corresponding servlet is executed to respond to that request. If no instance of the servlet exists in the web container, the corresponding class is loaded and an instance is created and initialized before the request is processed.

From the developer's perspective, it is possible to register listener classes for particular servlet life-cycle events. If an event is fired, the corresponding method of the listener class is invoked. You can register following listeners:

- `ServletContextListener` for `ServletContextEvent` fired during initialization and destruction of the web context.
- `ServletContextAttributeListener` for `ServletContextAttributeEvent` fired when a web context attribute is added, removed or replaced.
- `HttpSessionListener` for `HttpSessionEvent` fired when a session is created and destroyed.
- `HttpSessionActivationListener` for `HttpSessionEvent` fired during activation and passivation of the session.
- `HttpSessionAttributeListener` for `HttpSessionBindingEvent` fired when a session attribute is added, removed or replaced.
- `ServletRequestListener` for `ServletRequestEvent` fired when a servlet request is being processed.
- `ServletRequestAttributeListener` for `ServletRequestAttributeEvent` fired when a request attribute is added, removed or replaced.

All the listener interfaces and events are located in either `javax.servlet`, or `javax.servlet.http` packages. To define a listener class, you simply implement any of the listener interfaces and register the class in the application's deployment descriptor. The rest is taken care of by the web container.

2.1.1.2 Sharing Information

A web application is usually composed of multiple web components that need to cooperate, sharing business logic objects and other pieces of information required to fulfil their tasks. These can be stored as attributes in four scope objects:

- *Web context* represented by the `javax.servlet.ServletContext` class.
- *Session* represented by the `javax.servlet.http.HttpSession` class.
- *Request* represented by a subclass of the `javax.servlet.ServletRequest` class.
- *Page* represented by the `javax.servlet.jsp.JspContext` class.

Each scope object provides `getAttribute` and `setAttribute` methods that can be used to get and set attributes of given names. Since the attributes can be accessed concurrently, the access must be synchronized to prevent inconsistent states. This can be achieved by using the standard Java synchronization techniques.

2.1.1.3 Filtering Requests and Responses

A *filter* is a Java class that can inspect and modify an HTTP request before it is processed by a particular web component. After the web component generates an HTTP response, the corresponding filter can also inspect and modify this response. In general, filters can affect the way in which the HTTP requests are handled.

A web component can be filtered by a chain of filters. Besides, a filter can be mapped to multiple web components or URL patterns. Before a request for a specific URL pattern is processed by a particular web component, all filters in the filter chain are invoked in the order they were specified in the application's deployment descriptor. After a response is generated, filtering of the request-response pair is completed in the filter chain reverse order, allowing the filters to modify the response.

Each filter must implement the `javax.servlet.Filter` interface. This interface provides `init` and `destroy` methods called by the container during initialization and destruction of the filter. The most important is the `doFilter` method that receives the request, response and filter chain objects. This is the method used for request/response inspection and modification, as well as for chaining the filters in the filter chain. The chain can also be broken if an error occurs or if the business logic determines so. If this happens, a filter has to generate an HTTP response itself.

Filters are usually used for authentication, data conversions, encryption, XML transformations and various other tasks. They should never depend on the web component they do the filtering for so that they could be used with more than one component. With filters, you can simply customize requests or responses through wrapped requests or responses, following the *Wrapper* design pattern.

2.1.2 JavaServer Pages Technology

A *JSP Page* is a text document that provides a more natural approach to processing a request and generating an appropriate response to it. A JSP page typically contains both some static textual content and JSP elements that provide dynamic capabilities of a Java servlet. The recommended file extension for a JSP page is `.jsp`.

If the standard JSP syntax does not meet developer's requirements for any reason, there is a possibility to use the XML syntax. A JSP page with the XML syntax is called a *JSP document* and should have the `.jspx` file extension.

If a JSP page is too complex, it can be split into smaller JSP pages or JSP fragments and later assembled, using the include directive, into the resulting JSP page. The recommended file extension for a JSP fragment is `.jspf`.

2.1.2.1 JSP Page Life Cycle

The life cycle of a JSP page is very similar to that of a servlet. This is because JSP pages are translated and compiled into servlet classes before they are used to process requests. This happens at runtime, or at deployment time if required. When a web container detects a change in requested JSP page, it is automatically recompiled. This feature is very useful during development.

Translation of a JSP page is not a simple process. The web container converts the static textual content into Java code that writes the content into the response stream. JSP elements, such as directives or custom JSP tags, are treated by the web container in many different ways. For details about translation of JSP pages see [1].

If an error occurs during translation or compilation of the JSP page, a corresponding exception is thrown. If an error occurs during execution of the JSP page, a `JasperException` is thrown indicating the exact line in the JSP page that caused the error. The JSP technology also provides a mechanisms for displaying an error page when an exception is thrown.

Since each JSP page is actually a servlet, it is possible to access all four scope objects available to servlets.

2.1.2.2 Unified Expression Language

The new *Unified Expression Language* is a mechanism for both accessing and modifying the data stored in the JavaBean components, as well as for invoking public and static methods, and for performing arithmetic operations.

When used in JSP pages, only the read-only expressions are allowed, due to the JSP page life cycle. On the other hand, JavaServer Faces technology introduces a more complex life cycle that requires deferred evaluation of expressions, the ability to store data in JavaBean components and to invoke methods. The JSF technology as well as its life cycle are covered later in this chapter.

Because the Unified Expression Language has a lot of specific features, it is not possible to cover all of them in this brief introduction. You can learn more in [1].

2.1.2.3 JavaServer Pages Standard Tag Library

The *JavaServer Pages Standard Tag Library* (JSTL) provides a set of JSP tags encapsulating the standard functionality that simplifies the development and maintenance of JSP pages. This core functionality can easily be extended by adding another JSP tag library implemented by a specific vendor, or by implementing a custom JSP tag library.

The JSTL provides various functionality divided into five independent tag libraries. Each tag library is assigned a URI, giving it a namespace that uniquely identifies the library. The tag libraries and their corresponding URIs are as follows:

- The *Core* tag library identified by the `http://java.sun.com/jsp/jstl/core` URI, providing core functionality such as variable support, flow control, URL management and more.
- The *XML* tag library identified by the `http://java.sun.com/jsp/jstl/xml` URI, providing support for XML processing, XML transformations and flow control.
- The *Internationalization* tag library with the `http://java.sun.com/jsp/jstl/fmt` URI, providing functionality for manipulating locales, and for message, number and date formatting.
- The *SQL* tag library with the `http://java.sun.com/jsp/jstl/sql` URI, providing functionality for managing the data source, querying and updating data, and handling transactions.
- The *Functions* tag library identified by the `http://java.sun.com/jsp/jstl/functions` URI, providing functionality for determining collection length and for string manipulation.

Before a tag library can be used in any JSP page, it has to be referenced using the `taglib` directive at the top of the JSP page. This can be achieved either by specifying the URI of the tag library, or by providing the path to the *tag library descriptor* (TLD), an XML document containing necessary information about the library.

JSP tags can cooperate with their environment and the other JSP tags. This collaboration is done via the Unified Expression Language variables, or by nesting JSP tags into each other. This makes JSP tags a powerful tool for creating JSP pages.

For a complete list of the JSTL tags and their attributes see [4].

2.1.2.4 Custom Tags in JSP Pages

The JavaServer Pages technology provides a mechanism for creating custom JSP tags that may be used to encapsulate application-specific functionality. It is also possible to create JSP tags that encapsulate commonly used JSP fragments. Either way enables the developer to reuse the tags in multiple web applications and thus help increase productivity.

Related tags are organized into a tag library, as we saw with JSTL, that can be distributed in a Java archive. A tag library also contains a tag library descriptor, an XML document with the `.tld` file extension. A TLD is used by the web container to validate JSP tags used in JSP pages. It can also be used by the JSP page development tools for tags code completion.

The easiest way to create custom JSP tags is with tag files. A *tag file* is actually a JSP fragment with the `.tag` extension that allows you to implement the tag using the JSP syntax. This approach is suitable for web designers with little or no knowledge of the Java programming language.

A more flexible approach to creating custom JSP tags is implementing a *tag handler*, a class that encapsulates the functionality of a JSP tag. A custom tag handler is usually derived from the `javax.servlet.jsp.tagext.SimpleTagSupport` class and has access to all the objects accessible from a JSP page. The most important part of the tag handler is the `doTag` method responsible for tag's evaluation. This method is invoked by the web container when a corresponding tag is encountered in a JSP page.

2.1.2.5 Scripting in JSP Pages

Scripting in JSP pages allows you to use Java statements in JSP pages, effectively mixing the static page content with scriptlets and JSP tags. Because this approach is deprecated, it can be disabled in the application's deployment descriptor. You should be aware of this fact and use the JSP scripting as less as possible.

2.1.3 JavaServer Faces Technology

The *JavaServer Faces* technology is a framework for creating server-side web user interface components that run on the server and render themselves to the client. This approach enables the web developer to create web user interface components in a similar way as with the client-side user interface architectures.

The JSF technology clearly separates the application and presentation logic. This allows web developers working in a team to carry out their particular tasks independently, eventually linking their pieces of work together. The architecture provides an API for component state management, data processing, user input validation and events handling.

Based on top of the Servlet API, the JSF technology is not limited to a particular presentation technology, such as JSP pages, or a markup language. It is possible to create custom JSF components that render themselves to various client devices in different markup languages. When used with JSP pages as presentation technology, the JSF standard tag library is used to represent the web user interface components and other objects on the page.

See [1] for more information about the JSF technology.

2.1.3.1 User Interface Component Model

The JSF technology provides component architecture for creating configurable and reusable user interface components. These components can be simple, such as a button or a hyper-

link, or complex, such as a table or a form, composed of other components. All of them are derived from the `UIComponentBase` class which provides common functionality to all user interface components. A default set of components is included with the JSF technology. These include components for working with forms, buttons, inputs, select boxes, tables, images and hyperlinks. Each default component class can be easily modified or extended into a new user interface component.

While the functionality of a JSF component is hidden within its component class, rendering of the component can be handled by different renderers. This enables component developers to code the functionality once and provide multiple renderers for the component. In addition, page authors are free to choose the combination of a component and an appropriate renderer that best suits their needs. The JSF implementation ships with standard HTML render kit.

User interface components in the JSF application can generate events, for example when a button is clicked. To handle events, an application must provide a listener and register it on the JSF component. When an event is fired, the JSF implementation invokes the listener method responsible for processing the event. The JSF technology provides tools for processing action events, value-change events and data-model events.

A JSF component is often bound to a JavaBeans component providing data for the JSF component. Since the data type used to store the component data may differ from the data type used for the presentation, a mechanism for conversion between these data types is required. The JSF technology handles the conversion of basic data types automatically, allowing the developer to implement custom converters when appropriate. These can then be associated with a JSF component.

Before the data from a JSF component are stored in the corresponding backing bean, the validation occurs. The JSF implementation provides a set of common validators, most of them customizable via attributes. The validation model also enables you to implement custom validators.

2.1.3.2 Navigation Model

The JavaServer Faces navigation model uses an XML configuration file to define a set of navigation rules. A *navigation rule* for the page defines which page to go next based on the logical outcome returned by command buttons and hyperlinks. There may be multiple outcomes defined for each page, each of which can point to a different page.

The logical outcome is simply a string, such as `success` or `error`. It is used in the JSF page as a value of the `action` attribute of the command buttons and hyperlinks. In more complicated applications, an action method of the JSF component's backing bean can be used to determine the logical outcome.

When the user performs an action, an action event is generated by the corresponding JSF component. The default `ActionListener` handles this event and invokes action method registered on the component. This method is responsible for returning the logical outcome. Based on the outcome, the default `NavigationHandler` selects a page to be displayed.

2.1.3.3 JSF Page Life Cycle

The life cycle of a JSF page is a bit more complex than that of a JSP page. It is divided into several phases due to the JSF component architecture. A JSF page is actually a tree of JSF components, called a *view*, which is used during each phase of the life cycle.

The JSF page life cycle distinguishes two types of requests – initial requests and postbacks. An *initial request* for a JSF page is made when the user requests the page for the first time, usually by clicking a hyperlink. A *postback* means submitting a form previously returned as a result of the initial request.

When the user requests a JSF page, the *restore view phase* begins. If it is an initial request, an empty view is created and the life cycle continues with the render response phase. Otherwise the JSF implementation uses the state information stored on the server or client to restore the view.

The life cycle continues with the *apply request values phase* in which each component in the view sets its temporary value from the request. If a data conversion error occurs, a corresponding error message is generated and stored.

Then the *process validations phase* follows. During this phase, the JSF implementation uses the validators registered on the user interface components in the view to validate the submitted data. If any validation fails, an error message is generated and the life cycle continues with the render response phase.

If the temporary data is determined to be valid, the *update model values phase* begins. Temporary values are stored in the corresponding components' server-side objects and then the life cycle advances to the next phase.

During the *invoke application phase*, the JSF implementation handles application-level events, such as navigation or action events.

Finally comes the *render response phase* during which the page is rendered. The JSF components are added to the view if it is an initial request. If the request is a postback and there are any errors, they are displayed at the required position(s) in the page.

Events generated during each of the described phases are broadcast to interested listeners. Further details about listeners and the JSF page life cycle can be found in [1].

2.1.3.4 Custom JSF Components

A component developer can easily extend the standard JSF components and enhance their functionality. He or she can also create completely new user interface components if required. The development of a custom JSF component has several steps.

First, the custom component class needs to be created – the heart of the component defining its functionality and state. This class is responsible for either rendering the component, or delegating the task to a specific renderer class that does the rendering.

When the custom component class is ready, the JSF component has to be registered. If it generates any events, corresponding event handlers have to be created.

Finally, a tag handler class is created for the component. The tag handler determines

which renderer will be used during the render response phase of the JSF page life cycle. Then the custom tag is defined in the corresponding TLD.

The entire process of creating a custom JSF component is described in detail in [1].

2.1.4 Internationalization and Localization

The process of preparing an application for easy adaptation for various national and cultural conventions is called *internationalization*, often abbreviated as i18n. *Localization*, often abbreviated as L10n, is the process of extending an internationalized application to support specific national and cultural conventions. This is achieved without the need to recompile the application.

2.1.4.1 Localization in Java

The `java.util.Locale` class is used to represent a specific geographical, political or cultural region. It is actually represented as a string consisting of ISO language code, ISO country code and optional variant separated by underscores. Locales for the most frequently used language and country codes are available as the static attributes. In Java EE, the default locale depends on the web container the application runs in.

Locale-specific data is stored in an instance of the `java.util.ResourceBundle` class. A resource bundle represents a map containing key-value pairs where each locale-specific value of the resource bundle is uniquely identified by its key. A resource bundle can be stored in a text file with the `.properties` file extension, or represented as a subclass of the `ResourceBundle` class. The latter approach enables the developer to store non-string objects in the resource bundle.

A family of resource bundles contains multiple resource bundles with the same base name. For a specific locale, the locale string representation is appended to the base name, representing a specific resource bundle implementation. Each resource bundle family should contain an implicit resource bundle instance with no locale string appended.

For more details on this topic see [3].

2.1.4.2 Date and Number Formatting

Java applications use the `java.text.DateFormat` subclasses to parse and format date in locale-independent manner. The abstract `DateFormat` class provides methods for obtaining the date and time formatters based on the locale and the formatting style provided. The class internally uses the `java.util.Calendar` to implement the formatting.

The abstract `java.text.NumberFormat` class is responsible for parsing and formatting numbers. It provides methods for obtaining specialized formatters for numbers, currency and percentage. The choice format for a range of numbers is also available, generally used for handling plurals. All the above classes are described in detail in [2].

Dates and numbers can also be parsed and formatted with the JSTL. The Internationalization tag library provides a set of tags that handle these tasks.

2.1.4.3 Character Sets and Encodings

A *character set* represents a specific set of textual and graphic symbols. A *character encoding* determines how the individual symbols of the character set will be encoded. Character sets may have multiple encodings.

Java uses the standardized Unicode character set. When a Unicode character has to be expressed in a non-Unicode source file, the `\uXXXX` escape sequence can be used, where the `XXXX` is a hexadecimal representation of the character. This approach is used for example in property resource bundles described earlier.

Web components have to consider *request*, *page* and *response* encodings. The request encoding is automatically extracted from the `Content-Type` HTTP header, or set to default if not present. All three encodings can be set by the developer to override the defaults.

2.2 Web Services

Web services are designed to provide support for computer interaction over the network. This interaction involves a *service provider* and a *service requestor*. A service provider publishes data in a specific format described in a web service descriptor. Based on the knowledge of this descriptor, the service requestor is able to use the web service.

The Java EE platform provides the *Java API for XML Web Services* (JAX-WS) used for creating web services and clients communicating over an XML-based protocol. The main advantage of this API is platform independence of the Java programming language. JAX-WS uses the SOAP messages over HTTP and the Web Service Description Language (WSDL) to describe a web service.

Further details and other related technologies, such as the Java Architecture for XML Binding, the Streaming API for XML and SOAP with Attachments API for Java, are described in [1].

2.3 Enterprise JavaBeans

Enterprise JavaBeans are server-side components encapsulating business logic of an enterprise application. These components run in the EJB container, a runtime environment provided by the application server. The EJB container provides services common to all business applications such as the transaction management and security. This enables the developer to fully concentrate on business logic of the application.

The EJB component architecture is suitable for scalable and transactional enterprise applications. Components can be transparently distributed across multiple servers, still allowing an easy access to the components. Application clients thus may be thinner since they can locate and use the enterprise beans remotely.

Session beans and message-driven beans of the EJB 3.0 specification are described in the following sections, as well as entity beans used in previous releases of the EJB component architecture. The complex EJB component architecture is thoroughly discussed in [5].

2.3.1 Session Beans

A *session bean* is a reusable component that encapsulates business logic of an enterprise application. A client code accesses the session bean deployed on the application server and invokes methods that perform tasks inside the application server. There are two types of session beans available – stateful and stateless.

Stateful session beans are used to represent the state of a client session. This type of session bean is appropriate when the state represents an interaction between the client and the corresponding session bean. A stateful session bean can also hold information about the client between the method invocations. If the client stops using the stateful session bean, it is discarded and the state disappears.

Stateless session beans, on the other hand, do not maintain any client's state at all. A stateless session bean is therefore capable of handling multiple clients when no client-specific data is needed. The state of a stateless session bean is shared by all the clients using it. It is the only type of an enterprise bean that can implement a web service.

2.3.2 Message-Driven Beans

A *message-driven bean* asynchronously receives and processes messages sent by the Java EE components or other applications using the Java Message Service. Message-driven beans are similar to stateless session beans because they do not maintain any client-specific data. Therefore a pool of equivalent message-driven beans can be used to process incoming messages concurrently. They are also suitable for time-consuming tasks.

Clients of message-driven beans do not communicate with them directly by invoking methods. Instead, a client sends a message to the message destination assigned to a specific message-driven bean. Incoming messages are processed as soon as they arrive by the message-driven bean's `onMessage` method.

2.3.3 Entity Beans

In previous EJB releases, *entity beans* represented application data and provided persistence capabilities for the application. These container-managed components were accessed remotely and thus caused unnecessary network overhead. Their complexity lead to evolution of a new persistence standard, the Java Persistence API.

2.4 Java Persistence API

The *Java Persistence API* (JPA) provides object-relational mapping for managing relational data within Java applications. The JPA has been introduced as a part of the EJB 3.0 specification, based on both open-source and commercial ORM products. The model of the Java Persistence API is powerful, flexible and yet easy to learn and use.

An *entity* of the relational model is represented as POJO, resembling a serializable JavaBean class. Each entity class is annotated using the `@Entity` annotation and provides

getters and setters for its attributes. One of these attributes must be marked as the entity identifier, using the `@Id` annotation. Entities must also have a non-parametric constructor.

Entities in the Java Persistence API are persistable, they are not automatically persisted when created. It is the application that decides when it is the right time to persist, update or remove an entity. The functionality required to perform these tasks is encapsulated in the `EntityManager` class of the JPA. Its responsibility is to manage entities within a certain persistent context, therefore it provides methods for persisting, refreshing, merging and removing an entity. An entity manager further provides methods for finding an entity by its identifier, querying multiple entities and handling transactions.

Entity managers are obtained from an `EntityManagerFactory`, a factory configured by a specific persistence unit. Configuration of the persistence unit is stored in an XML file called `persistence.xml`. This XML file contains the name of the persistence unit, details about the database connection, transaction type and other properties.

See [6] for more information about the Java Persistence API.

2.4.1 Persistence in the Web Tier

In the classic web applications, an `EntityManagerFactory` can be injected into classes managed by the web container. These include Java servlets and servlet context listeners. The injected factory can be then passed to an object that encapsulates the database functionality of the application.

When the Java Persistence API is used in this way, the developer has to manually begin and commit transactions. It is possible to either request injection of the `UserTransaction` resource, or use `EntityTransaction` provided by an entity manager. The latter approach is only possible when the persistence unit is configured not to use the Java Transaction API.

2.4.2 Persistence in the EJB Tier

Working with the Java Persistence API in the EJB tier is much easier because the EJB container does a lot of work for the developer. The only thing the developer has to do is to request injection of the entity manager for a specific persistence unit. The injected entity manager is used in business methods to implement the desired functionality of the enterprise bean.

2.4.3 Java Persistence Query Language

The *Java Persistence Query Language* (JPQL) is a portable query language based on the EJB Query Language introduced in EJB 2.0. It combines syntax and semantics of the SQL with the object-oriented approach to the data representation.

Although the JPQL is very similar to SQL, it is not SQL. It is a language for querying entities and their relationships. It provides portability in a way that it can be easily translated into the SQL dialects of major database vendors. Developers may not be concerned about the database being used and do not need to know how the entities are mapped to it.

Queries written in JPQL operate on a set of entities specified in the persistence unit configuration file. The JPQL distinguishes select, aggregate, update and delete queries. Select and aggregate queries are used to retrieve entities from a persistent store, filter data, group results and summarize data. Update and delete queries allow the developer to update or remove entities that meet certain conditions.

2.5 Services

The Java Enterprise Edition further provides a number of services for the applications based on this platform. One of them is *security* for enterprise components deployed into different containers. Each container provides security for components that it hosts. The Java EE platform provides the declarative and programmatic security that can be used in conjunction. These can be used to secure both the Java EE and web applications.

Next comes the *Java Message Service* (JMS) that provides an API allowing the developer to create, send, receive and read messages. Communication between a sender and a recipient is asynchronous and reliable. The JMS API guarantees that the message is received just once, even though the recipient is not available at the moment the message was sent. Both the sender and the recipient need to know only the message format and the message destination used. Therefore the JMS API is mainly used when loose coupling is required.

Enterprise applications typically access and manage database data that must stay accurate and consistent all the time. Therefore *transactions* are used to control concurrent access to the data and to ensure the data integrity. The Java EE platform provides container-managed as well as user transactions. The *Java Transaction API* can be used to manually demarcate transactions by the developer.

Java EE components can access various resources in a similar way. The platform uses the *Java Naming and Directory Interface* (JNDI) to lookup such resources. Each resource has its own JNDI name that is used to perform a lookup or to request injection of the resource by the container. Resources can be injected using the `@Resource` annotation.

For further details on the Java EE services see [1].

Chapter 3

Java Web Application Frameworks

A *web application framework* is a piece of reusable software that makes the development of web applications easier by providing tools for handling common tasks. These may include for example localization, templating, page layout reuse, forms processing, file uploads, AJAX support and a lot more. Web application frameworks can be divided into two main groups – request-based and component-based.

Request-based frameworks, as the name implies, are used to process incoming HTTP requests. This approach provides a low-level control over the entire web application and corresponding HTTP traffic. However, the development of web applications with request-based frameworks may be a bit tedious.

On the contrary, *component-based* frameworks allow the developer to create reusable user interface components that can respond to various events. These components are then used to compose user interfaces of web applications. Sophisticated development environments often allow dragging and dropping of components into the page. Unfortunately, the developer often loses control over the exact look of the web page.

There are dozens of Java web application frameworks available, many of them conforming to the Model-View-Controller design pattern described in the next section. It is not possible to cover every single framework available. Therefore only the most widely used Java web application frameworks are discussed in the following sections.

3.1 Model-View-Controller

As stated earlier, *Model-View-Controller* (MVC) is a design pattern used by many web application frameworks. The main idea behind the MVC architecture is to separate the business model functionality from the data presentation and control logic. This separation allows the developer to provide multiple views of the same data for various clients and improves the maintainability of the application. However, the MVC pattern increases the design complexity of the application.

The *model* represents the application data and corresponding business methods for accessing and modifying the data. In the Java EE applications, EJB components are often used to represent the model and its business methods.

The *view* is responsible for presenting the model data to the user. The JavaServer Pages or the JavaServer Faces technology can be used to render the view. There are also alternative presentation technologies available for the Java EE platform.

The *controller* processes user interactions with the view and interprets them as specific actions performed by the model. In web applications, user interactions are HTTP requests that lead to changes in the state of the model or to invocations of model business methods. Based on the outcome of the performed actions, the controller selects an appropriate view to be displayed. The controller is usually represented as a servlet or as a filter.

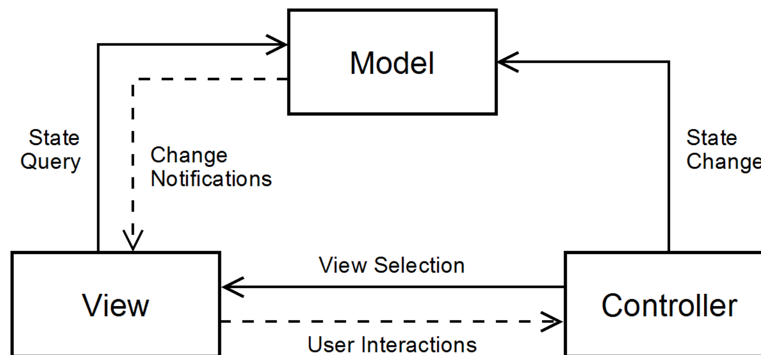

Figure 3.1: Model-View-Controller Diagram

3.2 Representational State Transfer

Representational State Transfer (REST) is an architectural style for distributed hypermedia systems. It is concerned with components and their interactions, rather than with the implementation details of these components and protocol syntax. REST introduces a set of constraints and principles applied to components within the architecture. Systems following these principles, such as the World Wide Web, are referred to as *RESTful*.

The key principle behind the REST pattern is a *resource*, virtually any information with a name. It may be an XML document, an image, a collection of other resources, or any other entity that might be somehow interesting for the client. Each resource has its internal representation and is identified by its URI, in terms of web applications by its URL. Clients use an uniform interface to access the resources.

REST requires a client-server protocol that that is stateless, cacheable and layered. HTTP is a good example of such protocol. The client sends all information required to access a resource in the HTTP request, then the server responds by providing the resource in an appropriate representation. No other messaging layers are required.

The REST architectural style is fully described in [8].

3.3 Request-Based Frameworks

This section introduces several request-based frameworks based on the Java EE platform. All of them conform to the MVC design pattern described earlier in this chapter.

3.3.1 Apache Struts

Apache Struts is an open-source web application framework based on Java EE technologies, XML and various Jakarta Commons packages. Struts provides a web controller for translating an incoming HTTP request into a specific action. This action is executed and the controller uses its result to select a view that renders a response. The framework further provides tools for working with static and dynamic web forms, including user input validation. Developers are also allowed to create own plug-ins that extend Struts functionality. The standard Tiles extension, for example, can be used to define a common page layout. Support for internationalization and localization is obvious.

The controller is represented by the `ActionServlet` class that is responsible for loading Struts XML configuration file at deployment time. This configuration file mainly contains mappings of URLs to specific actions. These actions are implemented as subclasses of the `Action` class that override the `execute` method. This method contains the logic behind the action and provides access to HTTP request and response objects, as well as to the associated form class. Subclasses of the `ActionForm` class may be used to represent these forms. Since this approach requires an individual class for each form in the application, the `DynaActionForm` class is often used to simply declare forms in the Struts configuration file. Validation rules for the forms can be then defined in a separate XML configuration file if automatic validation is desired.

The framework supports various presentation technologies, including the JSP technology with both JSTL and JSF, XSLT and Velocity. JSP tag libraries are provided, allowing the developer to access JavaBeans, display stored error messages, render HTML forms and perform other tasks. JSP tags are also used with the Tiles extension to build common page layouts. Actually, these tags are used to provide the predefined layout with static text or JSP fragments that form the content of a page.

Unfortunately, Struts does not have a native support for AJAX. The low-level approach to HTTP requests and response objects might be considered as a disadvantage as well, even though some developers find it useful. The most tedious job is synchronizing action classes with all the configuration files. On the other hand, Struts framework is widely used and thus the developer is provided with a good programming background.

More details about Apache Struts framework, its sources, library distributions, documentation and example applications can be found in [9].

3.3.2 WebWork

WebWork is another powerful Java web application framework claiming concepts of simplicity and interoperability. The use of WebWork should minimize the amount of code and enable the developer to focus mainly on business logic of the application. The Inversion of Control principle helps to achieve this objective. Actions can be successfully decoupled from validation rules and type conversion routines. Modular configuration files allow hundreds of actions to be divided into packages and namespaces. Interceptors may prevent an action

from being executed, or encapsulate common functionality applied to multiple actions. The framework further supports many view technologies, advanced AJAX features, and can be easily integrated with third party software.

The heart of the WebWork framework is the `FilterDispatcher` that processes incoming requests. It uses an `ActionMapper` do determine what action, if any, should be invoked. Before the action is really executed, an instance of the `ActionInvocation` class is created and used to call `Interceptor` classes. Once the action is finally executed, a corresponding view specified in the configuration file is used to generate a response.

As you can see, the architecture behind WebWork is very similar to Struts. However, WebWork introduces some concepts and features that are not available in Struts. Interceptors, for example, provide control over the execution of actions. Actions do not have to be thread safe because a new instance is created per each request. Additionally, WebWork actions can be easily unit tested outside the web container. Developers are actually shielded from servlets, but may choose to access request and response through action context.

WebWork documentation, tutorials and other related materials are available in [13].

3.3.3 Apache Struts 2

Apache Struts 2 is an elegant and extensible framework for creating Java web applications, originally known as WebWork 2. The WebWork and Struts communities joined forces few months ago and created a new version of Struts that is simpler to use. The Apache Struts 2 forms a new generation of action based frameworks, introducing bleeding edge technologies and new features. The framework is therefore suitable for more complicated web applications that can be easily profiled, debugged, extended and maintained.

Because the WebWork architecture was briefly described in the previous section, it is not necessary to describe it again. Tutorials, guides, HOWTOs and further documentation related to Struts 2 can be found in [9].

3.3.4 Stripes

Stripes is a light-weight web application framework based on latest Java technologies. The main goal behind Stripes is to make the development of web Java applications as easy as possible. There is no need for external configuration files, Stripes uses convention over configuration to map actions to specific URLs. Additional configuration or validation of user input is done simply by annotating action classes.

To get Stripes running, `StripesFilter` and `StripesDispatcher` servlet need to be configured in your application deployment descriptor. Stripes auto-discovers `ActionBean` classes at the deployment time, you only need to follow few naming conventions. An action bean is like the Struts action and action form put together. URL binding generated for an action bean can be easily overridden with the `@UrlBinding` annotation. Each action bean may provide multiple handlers for processing various requests. A default handler can be specified in case Stripes fails to determine the handler to be invoked.

Stripes includes interceptor design allowing the developer to simply add functionality to Stripes. This functionality might be used to handle application security, for example. The framework also introduces its own system for page layout reuse. Compared to Tiles, the Struts system is much simpler, only three JSP tags are required to define and use a page layout. Quite interesting are the features related to localization which works even when no action gets invoked. Stripes has also built-in support for AJAX.

See [11] for further details regarding Stripes, its documentation and tutorials.

3.3.5 Spring MVC

Spring MVC is a flexible web application framework based on the core functionality of the Spring Framework for the Java Enterprise Edition. Spring MVC is built directly on top of the Java Servlet API to provide full control over HTTP requests and their processing. This is achieved by defining strategy interfaces for important tasks and a clear separation of roles in the processing model. These are fully customizable, ranging from simple to sophisticated implementations. Configuration model is powerful and straightforward, allowing easy configuration of the framework, as well as of application classes. Multiple view technologies, including for example JavaServer Faces or Velocity, are supported by the framework.

The architecture of the framework is designed around a front controller represented by the `DispatcherServlet` class. It receives requests and dispatches them to corresponding handlers. These handlers are represented as Java classes implementing the `Controller` interface, usually extending one of the abstract controller classes provided. A controller itself may render a response or prepare model data that are then passed to a view that does the rendering. Ultimately, a response is sent back to the client.

You can learn more about the Spring MVC framework in [10].

3.4 Component-Based Frameworks

Two web application frameworks focused on user interface components are mentioned in this section. These are used mainly for illustration that different approaches to writing web applications are also available. Selection of the frameworks is not as wide as in the previous section.

3.4.1 JavaServer Faces

In the previous chapter, you learned that the JSF technology introduces a flexible framework for creating web user interface components. It has built-in support for AJAX so virtually any component can use its features. There are different strategies of providing AJAX support for the component allowing you choose the one that best suits your needs. However, without JavaScript the technology may be difficult to use because you are not able to use command links for navigation. It might be a suitable technology for web applications with great number of forms and other components.

3.4.2 Tapestry

Tapestry is an open-source Java web application framework for creating dynamic, robust and highly scalable web applications. Built upon the Java Servlet API, it works in any application server or web container implementing this API. A web application written with Tapestry is divided into a set of pages that are composed of components. You are not concerned with requests and query parameters, you just concentrate on developing your components, Java classes, their methods and properties. The distribution ships with both simple and complex components, allowing you to scale your application as required. Tapestry can be easily integrated with various back-ends, for example with the Spring Framework.

Developing web applications with Tapestry actually means writing HTML templates consisting of plain HTML tags with special `jwcid` attribute. Each tag containing this attribute is replaced by a code that renders a corresponding component. This makes the framework easy and efficient to use because the HTML templates can be easily previewed in a web browser. In addition, there is no need for a specialized editor, web designers can simply use an HTML editor. This might be considered as an advantage over JSP pages. To access data within the templates, a special Object Graph Navigation Language is used. It can be compared to expression language used in JSP pages that is quite alike.

Other concepts and advantages of the Tapestry framework are described in [12].

3.5 Related Technologies

Web application frameworks are not always used individually, they may work in conjunction. Sometimes a specific functionality not provided by the chosen framework is required. In such cases, other frameworks or single-purpose tools may be used. A web application can take advantage of multiple technologies that cooperate to bring the overall result. This sections describes one such tool.

3.5.1 UrlRewriteFilter

Based on Apache's `mod_rewrite` plug-in, *UrlRewriteFilter* is a filter which allows you to rewrite URLs before they are processed by your application. It is a powerful tool that enables you to keep URLs tidy and independent of the underlying technology or application framework you use in your web application.

UrlRewriteFilter uses an XML configuration file to define a set of rules that apply during the rewrite process. What makes the filter very powerful is the fact that most parameters can be entered as regular or wildcard expressions. You can specify inbound and outbound rewrite rules that are used before a request is processed and while a response is being generated, respectively. This way you can use application-specific URL format that is never seen by users of your application. In addition, it is possible to define various conditions that must be met before the rewriting occurs.

See [14] for more details about this filter.

3.6 Evaluation

It has always been difficult do decide which web application framework is the best, mainly due to the diversity of available frameworks. Since each framework may provide a different approach to developing web applications, it is much better to have a general overview and then choose a framework appropriate for your specific needs. You can also profit from the fact that many frameworks can be integrated with other frameworks or technologies. On the other hand, this flexibility may be considered as a disadvantage when you have got plenty of frameworks to choose from.

Unfortunately, the mentioned scenario applies to web application frameworks based on the Java EE platform. There are dozens of them, each of which introduces slightly distinct concepts and approaches. It is virtually impossible to try all of them in order to find out which one you should use for your application. At least you can decide whether to focus on requests or user interface components. It is obvious that each of these two approaches is suitable for different kinds of applications.

Request-based frameworks are appropriate when you need control over exact URL formats, query parameters or the HTML code that is sent in response to the client. It should be straightforward to implement a RESTful web application with this type of framework. Component-based frameworks, on the contrary, are suitable for complex web applications composed of user interface components. Such applications might for example include administrative interfaces or any other types of applications that need to process large amount of user input data. Reusable user interface components should make the development much easier and similar to writing a GUI-based application.

Nevertheless, you can always end up developing your own web application framework from scratch, or extending an existing one. This usually happens when you require stunning performance, or intend to use the framework for specialized applications, for example multiple online shops of a single company.

Chapter 4

Online Shop Web Tier

An online shop is usually used to present products or services that a certain company sells on the World Wide Web. Customers use a web browser to access the online shop web application that allows them to buy products. On the other side, there are employees of the company who maintain a product catalogue and provide support for the customers. Somewhere in between there are web developers who make this interaction possible. Therefore a simple online shop can be divided into three related parts:

- The web-based *customer interface* used to present the portfolio of products, allowing the customers to do the actual shopping. This is the part we will mainly discuss in this chapter, referring to it also as a *web tier*.
- The web-based or GUI-based *administrative interface* for employees of the company responsible for managing the product catalogue.
- And the *business logic* encapsulating business methods related to the online shop. This part makes the interaction between the company and customers possible.

We are not actually concerned about the processes that need to be carried out when the customer orders some products. Nor are we interested in the stock control and related matters. The business logic provides the web tier with sufficient functionality required to place an order. What happens next is not an objective of this thesis.

This chapter is divided into three main sections. In the first one, an analysis of the online shop web tier is performed. This analysis is used in the next section to specify requirements of the web tier. Finally, in the last section, a conceptual design of the online shop web tier is proposed.

4.1 Analysis

The web tier of an online shop is used by customers to browse the product catalogue and eventually buy products. Actually, they are the only users of the online shop web tier. However, the web tier has to be first designed and developed before the customers can use it. There is usually a programmer who does the programming, and a web designer who designs the look and layout of the web application. It is necessary to analyze all these three perspectives to successfully specify the overall requirements.

4.1.1 Customer's Perspective

A customer visits the online shop in order to look for a specific product, to compare prices and attributes of available products, to read user reviews, and eventually to buy chosen products. He or she might want to add few products in the shopping cart, but order them later that day. Regular customers might appreciate the ability to register, providing personal and shipping information that are then used whenever they place an order. They may also want to start shopping from a computer in their office and finish it later at home. Foreigners using the online shop would appreciate a localized web interface.

4.1.2 Web Designer's Perspective

A web designer is responsible for the graphic design, layout and content of the web pages used for presentation. He or she has to make sure that the layout of the web pages is independent of the web browser, and that the web application can be easily used by handicapped people. Another responsibility is to make the user interface attractive and easy to use, being aware of the fact that the customer is only a click away from a different online shop.

Web designers are usually not interested in programming, they might only want to use some client-side scripting to make the web pages more user-friendly. They work in cooperation with programmers, specifying their requirements and using various web components provided by these programmers.

4.1.3 Programmer's Perspective

A programmer is responsible for all the coding related to the web tier. His or her responsibility is to make the functionality of the business logic available in the web tier, providing support for the web designer. The provided functionality should be reusable and again easy to use. It is the programmer who should be thinking in terms of requests and query parameters, enabling the web designer to work only with objects and simple expressions. A programmer should also make the online shop available, no matter how many customers are using it at the moment. This also depends on the business tier of the application.

4.2 Specification of Requirements

Based on the previously performed analysis, we are now able to specify the requirements of a well-designed online shop web tier, regarding both the customer's and developer's perspectives. Since web designers and programmers work together on the overall result, it is suitable to specify their requirements in conjunction.

4.2.1 Customer's Perspective

A typical online shop customer does not usually pay attention to detail, he or she just wants to do the shopping and leave, probably never coming back again. However, there are some

details that the customer might not notice while using the online shop application. But when he or she does, it means that something is wrong, or does not feel right. These are the details resulting in the following requirements from the customer's perspective:

- The online shop should be *accessible*, meaning that it can be used by virtually any customer, no matter what web browser he or she uses. Additionally, the web application should work without any client-side scripting, or without the images being displayed. It should be also possible to use screen readers easily.
- The web pages should be *well designed*, providing the same look of the customer interface, no matter what web browser is used to display it.
- The application should be *intuitive* and *easy to use*, giving the customer a feeling that everything goes smoothly and nothing will be difficult.
- The web application should use *friendly URLs* that would be easy to remember.
- The customer interface of the web tier should certainly be *internationalized*, allowing easy localization into additional languages if required.
- Regular customers should be provided with an ordinary *non-persistent shopping cart* associated with their session. The products placed to this shopping cart should be available for at least 12 hours.
- Registered customers should be enabled to use a *persistent shopping cart* that would hold the products as long as the corresponding customer is registered.
- The online shop web tier should *guarantee secure access to customer's personal details* stored in his or her personal account.

Other requirements not directly related to the online shop web tier should be obvious. These might for example include up-to-date information, accurate product details, sufficient company details available, transparent business and shipping policy, and other.

4.2.2 Web Developer's Perspective

Web developers require appropriate technologies and tools that enable them to successfully meet the requirements related to the customer's perspective. From the developer's perspective, the web tier of an online shop should meet the following requirements:

- It should be *flexible* and *extensible*, enabling the developers to easily integrate new technologies, or extend the functionality of the web application.
- The entire web application should be *scalable*, allowing the developers to painlessly add extra processing power if required.

- The web tier should be *easy to maintain*.
- Programmers should find it *easy to create reusable components* and tools that the web designers can use to compose web pages, fill them with dynamic content, and provide the customer with desired functionality.
- Developers should have *perfect control over URLs construction*.
- Web designers should not need to know any programming language in detail.
- The web tier should provide web designers with facilities that make it *easy to define and modify a page layout* common to multiple web pages.
- Web designers should *conform to the latest web standards* and also *separate data from their presentation* in the resulting web pages.
- *Localization* of the customer interface should be straightforward.

These were probably the most important requirements of the online shop web tier. However, every web developer might consider another features important as well.

4.3 Conceptual Design

Considering the preceding requirements, we can finally propose a conceptual design of the online shop web tier. We will focus only on the web tier itself, assuming that the business logic of the online shop has already been implemented, most probably as a set of EJB modules. The conceptual design is introduced in the following sections, proposing specific technologies and concepts that can be used to implement the web tier in Java EE.

4.3.1 Presentation Technologies

XHTML Strict should be used to create web pages, rather than the good old HTML. One of the reasons is that XHTML forces the web designer to follow strict rules and thus create valid web pages. In addition, XHTML Strict disallows using any presentation-oriented tags. This results in the data and their presentation being separated. XHTML web pages can therefore be easily read by screen readers and that makes them accessible.

Cascading Style Sheets can be then used to provide a graphic layout of XHTML web pages. The layout is defined in a separate CSS file that can be attached to every web page. This makes the layout maintenance really simple, allowing the web designer to provide a brand new graphic layout if necessary. In addition, the web designer can provide another CSS file that defines a print layout.

Web designers should use *FLASH*, *Java applets* and other similar technologies as less as possible. These make the web pages inaccessible and therefore should be used for artistic effects only, certainly not for page navigation.

4.3.2 Web Application Framework

The web tier should serve as an interface between the customer and the business logic, therefore we need a mechanism for transforming customer requests into business method calls. This can be easily achieved by employing any *request-based web application framework conforming to the MVC pattern*. Notice that customers do not need to use web forms very often, they just send various requests. Because most of these requests are read-only, there is no need for sophisticated user interface components.

A request-based web application framework also provides the developers with control over constructed URLs. This enables us to employ any *URL rewriting engine* that can be used to construct friendly URLs (great for SEO) and make the web tier independent of the web application framework being used. This approach also makes scalability easier by providing capabilities for transparently distributing requests to multiple servers.

Many web application frameworks further provide a *templating engine*, such as Struts Tiles, that allows the web designers to prepare a page layout common to multiple web pages. This again makes the maintenance of web pages and their layout easier.

4.3.3 View Technologies

The *JavaServer Pages* technology can be easily adopted by web designers and used for creating dynamic web pages. It is possible to mix XHTML tags with JSTL tags and expression language to provide whatever content the web designers want. The Unified Expression Language enables them to easily access scope objects and their attributes. However, web designers should not be allowed to use scriptlets and therefore the *scripting in JSP pages should be disabled*. Web designers had better rely on the business logic provided.

The JSP technology also allows programmers to create custom JSP tags that encapsulate specific functionality required by web designers. Additionally, web designers themselves can create JSP tag files composed of JSP fragments that encapsulate simple functionality they need. This helps them to avoid duplication of code.

The *JavaServer Faces* technology does not seem to be suitable as the primary view technology. The main problem is that many JSF pages are not accessible without JavaScript. Another problem is that the developer does not have control over URL construction, every action results in submitting a hidden form. However, the JSF technology might be suitable for simple AJAX components. Nevertheless, many request-based web application frameworks can handle this task as well, making JSF unnecessary.

4.3.4 Client-Side Technologies

JavaScript might be used to make the customer interface more user-friendly. However, it should not be used to implement essential functionality of the customer interface. Every action not requiring the entire web page to be reloaded might use an *AJAX component*. The customer gets the feeling that everything goes smoothly, the HTTP traffic between the application server and customer's browser will be reduced.

4.3.5 Shopping Cart

The *Enterprise JavaBeans* component technology seems to be a good choice for implementing the shopping cart. The programmer can focus solely on the business logic of the shopping cart, other tasks are taken care of by the EJB container. The programmer should follow the *Decorator* design pattern and provide unified interface for accessing both the non-persistent and persistent shopping carts. These specific implementations can be hidden in the EJB module providing business interface of the shopping cart.

4.3.6 Internationalization

Since we use the JSP technology for the view, we can rely on the *Internationalization tag library* provided by JSTL. If we use JSP tags of this library, localization of the application into another language only means adding corresponding resource bundle. Many web application frameworks further provide their own internationalization and localization capabilities. Anyway, we should use only one of them to avoid inconsistency.

Unfortunately, we have only internationalized the web tier by using JSTL. If localized product details are required, then the business logic needs to be internationalized as well. The web tier can hence use it to provide localized content.

4.3.7 Security

All the data the web tier uses is accessed via the business logic that should be sufficiently secured. This security is usually provided by the EJB container the business logic runs in. We only have to ensure that any sensitive data the customer submits are transferred securely over the HTTP. The *Secure Sockets Layer* enables us to do so.

Chapter 5

Reference Implementation

A simple prototype of the online shop web tier has been implemented, following the conceptual design proposed in the previous chapter. It was also necessary to implement basic business logic of the online shop, and an administrative interface for managing related data. These would be normally provided by respective web development teams.

The entire enterprise application was developed in NetBeans IDE 5.5 with the Java SE Development Kit 6 Update 1. The Sun's Java EE 5 implementation was used in conjunction with the Sun Java System Application Server Platform Edition 9.0 Update 1 Patch 1. All these development tools were used on Microsoft Windows XP Professional.

A brief description of the reference implementation follows. All source codes are available on the attached CD, see the Appendix A for more details.

5.1 Customer Catalogue

The `CustomerCatalogue` is implemented as an EJB module encapsulating business logic related to customer's account. This account is actually represented by a `Customer` entity. Its implementation is very simple, only providing access to customer's full name, e-mail address and password. The EJB module uses the Java Persistence API to store the customer's details in a persistent store.

5.2 Product Catalogue

The `ProductCatalogue` is also implemented as an EJB module. It encapsulates business logic related to `Manufacturer`, `ProductType` and `Product` entities. Their names should be self-explanatory, but it is not obvious that the `ProductType` entity provides a mechanism that can be used to create a tree of product types. The product catalogue uses the Java Persistence API to store all the entities in a persistent store.

5.3 Shopping Cart

The `ShoppingCart` EJB module provides an interface to the shopping cart used in the web tier. The Decorator design pattern was used to combine the persistent and non-persistent shopping carts into one module with common interface.

The NonPersistentShoppingCartContent class encapsulates the functionality of the non-persistent shopping cart content, while the PersistentShoppingCartContent class enables the developer to manipulate with the persistent content of the shopping cart. Both classes implement the ShoppingCartContent interface defining common functionality. When a customer logs in, all the contents of his or her non-persistent shopping cart is moved to the persistent one.

Unlike the customer or product catalogue, this EJB module is not based on the Java Persistence API. It uses the Jakarta Commons DbUtils to access the database and execute SQL queries that effectively change the content of the persistent shopping cart.

5.4 Web Administration

The Web Administration module provides a JSF web application for managing data stored in both customer and product catalogues. The implementation was generated by appropriate tools of the NetBeans IDE to save some development time, and later customized to suit all my needs. This module does not use any functionality of the corresponding catalogue modules, it just uses their entities.

5.5 Web Tier

The web tier of this simple online shop implementation is based on the Apache Struts web application framework. There are certainly better web application frameworks, but Struts is directly supported by the NetBeans IDE I used for development. I wanted to use Struts 2 instead, but it did not work with the UrlRewriteFilter I use for rewriting URLs. Stripes might be a good choice next time, although it has some drawbacks as well.

There are two application-specific listeners for handling desired servlet life-cycle events. The PatroNgServletContextListener class is used to load the customer and product catalogue EJB modules at deployment time. The PatroNgHttpSessionListener class creates an instance of the shopping cart EJB when a new customer starts using the online shop, and places it into the customer's session object.

Struts actions use the abstract PatroNgAction class to encapsulate common functionality. Several Struts actions extending the abstract class are provided. These allow the customer to list the products, display product details, add products to the shopping cart, log in and log out. The JSP pages are used as the view technology.

The web tier also provides a custom JSP tag library. It contains the productTypeTree tag for rendering and expanding the product type tree. Web designers can further use the price tag to format price in CZK in the locale-specific manner.

The JSTL Internationalization tag library is used to handle internationalization and localization of the entire web application. It was not possible to use the internationalization mechanism provided by the Struts framework because it caused undesired results when the first requested locale was unavailable. Anyway, there are two localizations available – Czech and English.

Chapter 6

Conclusion

The main objective of this thesis was to study the technologies provided by the Java Enterprise Edition 5 platform and the web application frameworks based on this platform. This resulted in my deep understanding of the web tier related technologies in Java EE. Next, the analysis of an online shop was performed, resulting in the specification of requirements being summarized. Applying the knowledge gained in the first chapters, the conceptual design of the online shop web tier was proposed, including suggestions of the appropriate technologies and approaches to development. A reference implementation of the online shop web tier was developed following this conceptual design.

The Java EE 5 platform proved to be suitable for developing complex web applications such as online shops. The Java Servlet technology provides the developers with a powerful API for low-level processing of the incoming HTTP requests. Web developers with little or no experience with the Java programming language can easily use the JavaServer Pages technology. The JSP technology provides them with a more natural approach to creating HTTP responses. When the core functionality is not sufficient, the developers can easily extend it by implementing custom JSP tags encapsulating the application-specific functionality. More complex web applications may take advantage of the JavaServer Faces technology providing a mechanism for creating user interface componets that run on the server and render themselves to the client.

The Java web application frameworks are based on these technologies to provide sophisticated tools for developing robust and scalable web applications. Other powerful tools can be used in conjunction with these frameworks to provide additional functionality or to make the development easier. Many Java web application frameworks are request-based, but there are also frameworks focused on user interface components. The most widely used Java web application frameworks conform to the Model-View-Controller design pattern that separates the business model functionality from the data presentation and control logic.

The reference implementation introduced in this thesis serves as a proof of the proposed conceptual design. Even though the provided implementation is very simple, it could be easily extended into the implementation of a fully functional online shop. A framework for creating online shops might be further developed from such implementation. However, it will certainly require a lot of additional work.

Bibliography

[1] Jendrock, E. and Ball, J. and Carson, D. and Evans, I. and Fordin, S. and Haase, K.: *The Java EE 5 Tutorial* <http://java.sun.com/javaee/5/docs/tutorial/doc/>, Sun Microsystems, Inc., 2007. 2, 2.1.2.1, 2.1.2.2, 2.1.3, 2.1.3.3, 2.1.3.4, 2.2, 2.5

[2] *Java Platform, Standard Edition 6: API Specification* <http://java.sun.com/javase/6/docs/api/>, Sun Microsystems, Inc., 2006. 2.1.4.2

[3] *The Java Tutorials* <http://java.sun.com/docs/books/tutorial/>, Sun Microsystems, Inc., 1995–2007. 2.1.4.1

[4] *JavaServer Pages Standard Tag Library 1.1 Tag Reference* <http://java.sun.com/products/jsp/jstl/1.1/docs/tlddocs/>, Sun Microsystems, Inc., 2002–2003. 2.1.2.3

[5] Sriganesh, R. and Brose, G. and Silverman, M.: *Mastering Enterprise JavaBeans 3.0*, Wiley Publishing, Inc., 978-0-471-78541-5, 2006. 2.3

[6] Keith, M. and Schincariol, M.: *Pro EJB 3: Java Peristence API*, Apress, 978-1-59059-645-6, 2006. 2.4

[7] *Java BluePrints: Guidelines, Patterns, and Code for End-to-End Java Applications* <http://java.sun.com/reference/blueprints/index.html>, Sun Microsystems, Inc., 1994–2007.

[8] Fielding, R.: *Architectural Styles and the Design of Network-based Software Architectures* <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>, PhD thesis, University of California Irvine , 2000. 3.2

[9] *The Apache Struts Project* <http://struts.apache.org/>, project's web site, The Apache Software Foundation, 2000–2007. 3.3.1, 3.3.3

[10] *Spring Framework* <http://www.springframework.org/>, project's web site, Interface21, 2006. 3.3.5

[11] Fennell, T.: *Stripes* <http://mc4j.org/confluence/display/stripes/>, web site of the project, 2006. 3.3.4

[12] *Tapestry* <http://tapestry.apache.org/>, web site of the project, The Apache Software Foundation, 2006–2007. 3.4.2

[13] *WebWork* <http://www.opensymphony.com/webwork/>, web site of the project, OpenSymphony, 2000–2007. 3.3.2

[14] Tuckey, P.: *UrlRewriteFilter* <http://tuckey.org/urlrewrite/>, web site of the project, 2005. 3.5.1

Appendix A

Contents of Attached CD

The attached CD contains the following items:

- a PDF version of the thesis,
- DocBook and LaTeX source files plus both images required to compile the thesis,
- and a reference implementation of a simple online shop.

The PDF version of the thesis is located directly in the root directory of the CD. Source files and images used to compile the thesis are located in the `thesis` directory.

The reference implementation is available in the `project` directory. Its subdirectory named `PatroNG-Web` contains a NetBeans enterprise project used to build and deploy the entire application. The administrative web interface located in the `WebAdministration` subdirectory has to be built and deployed manually.