Chapter

4

Java EE

CHAPTER AUTHORS

Iryani Binte Mohd Nasril

Kaldybayev Ayan

Luc Francis Edouard Boissaye

Wen Zihe

CONTENTS

1 Introduction to Enterprise Systems

Definition of Enterprise Systems (ES)

"Enterprise System is software applications that have cross-organizational capabilities as opposed to department or group-specific programs."

-By Neil Kokemuller

Enterprise Systems also allows for collaboration, communication and information gathering across the organization. An example of Enterprise Systems is an Enterprise Resource Planning (ERP) system used in many large organizations.

Enterprise Systems requires a lot of infrastructure. The business logic is the main part of an Enterprise System. It can be really complex. The main task you have as a developer is to translate the business processes and workflow into code. If you want to be able to maintain and split the code you need to create components that are able to work together.

1.1 Features of Enterprise System

Availability

The system is supposed to be up and ready at any given point in time. Sometimes, there could be a sudden increase in user demands. If the system cannot handle the load, it will result in a system downtime and can impact the company's business performance. Hence, availability of an Enterprise System is critical because the system is supposed to be supporting 24x7 services.

Scalability

The way that businesses are handled and operated is changing. Enterprise Systems thus must have a flexible architecture, which can reply to fast changes that often happen in the organizations. The systems need to have the scalability feature in order to adopt new changes happening in the organizations.

Performance

Organizations and Enterprises often invest heavily in their Enterprise Systems which are meant to improve their business workflow, data control, client management as well as customer responsiveness. The expenditure in Enterprise Systems grows and eventually turns into a significant part of the total business cost. This has led to the fact that shareholders and managers care very much about the performance of Enterprise Systems used in their organizations.

Security

Security is always one of the most important requirements in Enterprise Systems. The systems have to be secured in order to ensure the continued system availability and data confidentiality.

Manageability

A large percentage of Enterprise Systems fail mostly because of their high complexity which leads to the fact that the systems are not easy to be controlled and managed. In this case, the Enterprise Systems Management (ESM) specialists have to monitor the system operations as well as the performance in order to track down the source of problems and then pinpoint and fix the problems in the underlying layers.

Data Integrity

One of the primary design considerations for the Enterprise Systems is data integrity. Data integrity means that data in the systems should not be lost or corrupted.

Interoperability

Interoperability is the ability of Enterprise System (or any general IT system) to use information and functionality of another system. One of the examples will then be data exchange between two systems. Interoperability is a key to building Enterprise Systems with mixed services. It helps in achieving improved efficiency of the system operations. Hence, Enterprise Systems are required to be interoperable so that they can collaborate with other systems.

1.2 Technologies

Two of the technologies commonly used for implementing enterprise systems are Java Enterprise Edition (Java EE) and .NET.

Java EE, currently in version 6, is an extension to Java Standard Edition (Java SE). It provides additional APIs which are not available in Java SE. The APIs are mainly for the development of Java enterprise applications.

In addition, .NET is another common technologies that is used to create and Enterprise System. However, in the remaining sections we will be focusing on Java EE rather than .NET and illustrate how easy is it to create an ES using Java EE.

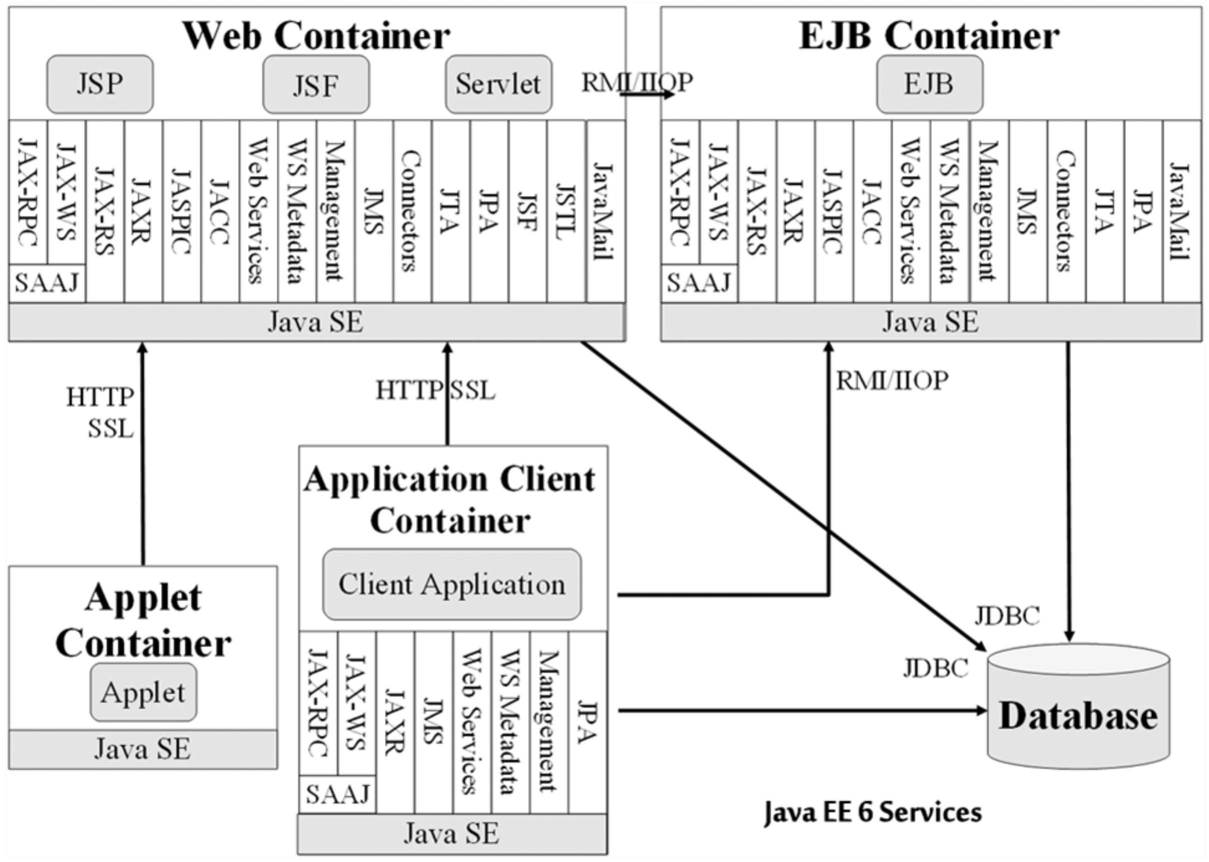
2 Overview of Java EE

Java EE provides you an infrastructure to create and maintain Enterprise Systems. Some of the services that Java EE provides are mentioned below:

JTA: Java Transaction API provides a standard interface for demarcating transactions. The Java EE architecture provides a default auto commit to handle transactions commits and rollbacks.

JPA: Java Persistence API is a solution for persistence. Persistence uses an object/relational mapping approach to bridge the gap between an object-oriented model and a relational database. JPA can also be used as a query language.

Security Services: Java Authentication and Authorization Services (JAAS) enables services to authenticate and enforce access controls upon users.



The diagram above depicts the main architecture of Java EE. Java EE itself is a huge topic. For the next few sub-chapters, we will be touching on the main important components in Java EE, mainly Entities and Enterprise Java Beans.

2.1 Understanding Java EE

Java EE is a specification for a server. When you say you are using Java EE, it means you are writing code that is following Java EE specification and so can be deployed on any Java EE compliant Server. Java EE is only a set of specifications.

2.2 Java EE6 Specifications

It is the Java Community process that is creating the Java Specification. The Java Community process is an open website where anybody can register to comment and participate on Java Specification Request (JSR). When a JSR is accepted by the community it will be added to the next Java release. The Java Community process is responsible for all the Java Product:

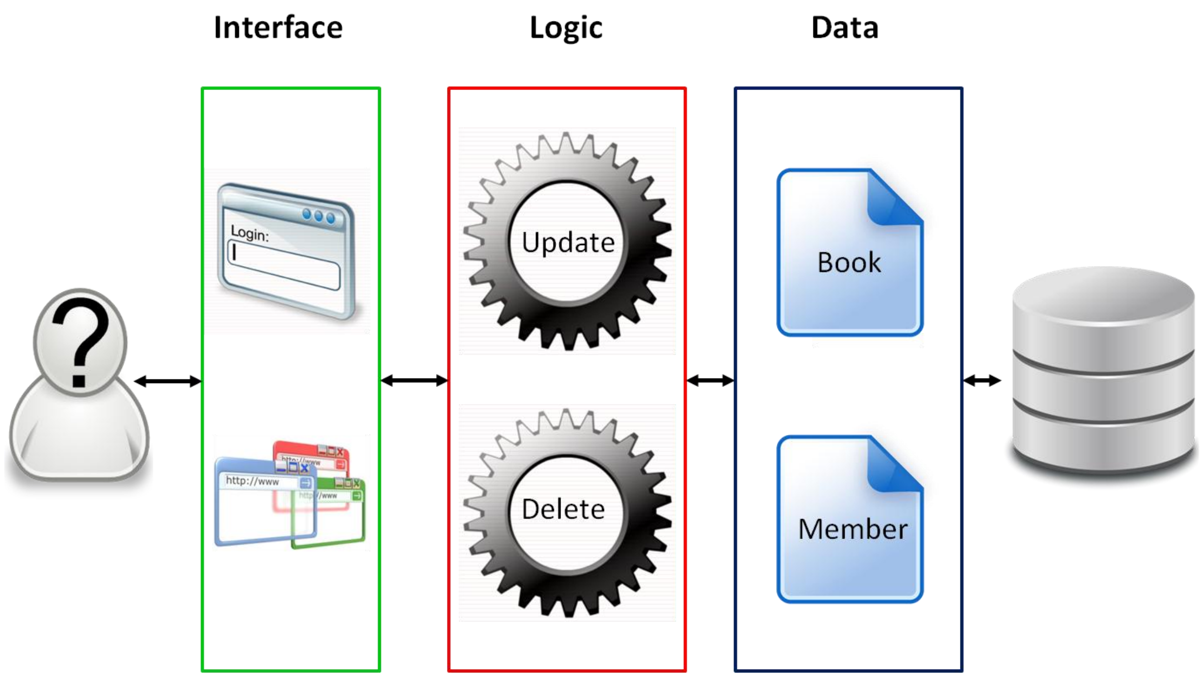
- Java Standard Edition (Java SE) which is the most commonly known as Java and is used to create desktop applications.
- Java Enterprise Edition (Java EE) which is made to create Enterprise system.
- Java millennium (Java ME) which is made to create Embedded Systems.
- Java Card which is made to create programmes for Smart Cards.

Java EE is a set of 28 specifications which provide common interfaces to use Java EE compliant server.

3 Creating an Application

Throughout the rest of the section, we'll be using a Library System that has been created as the example.

Applications are made up of user interface, business logic, and data as shown below. Without the interactions happening between them, no applications can be successfully built.



4 Data Layer

Database is important as it stores business data and act as a central point between applications. Persistent data is everywhere, and most of the time it uses relational database as the underlying persistence engine.

In Java, we manipulate objects as instances of classes. An object that can store its state to get reused later is called persistent.

4.1 JPA Overview

JPA can be used to access and manipulate relational data from Enterprise Java Beans (EJB), web components, and Java SE applications. All classes and annotations of this API are in the javax.persistence package. The main components of JPA are as follows:

- Object Relational Mapping, which is the mechanism to map objects to data stored in the database.
- An Entity Manager API to perform database-related operations such as Create, Read, Update and Delete (CRUD) operations.
- The Java Persistence Query Language (JPQL), which allows users to query and retrieve data easily.

4.2 Understanding Entities

As we have mentioned before, objects are instances of classes and objects can be persisted and reused. However when we are talking about mapping objects to a relational database or persisting objects, the term “entity” should be used rather than “objects”. It’s because objects are instances that live in a memory while entities are objects that live shortly in the memory and persistently in a database. JPA has a certain default mapping rules (eg., the table name is the same as the entity name).

4.3 Creating Entity Class

Let’s create a simple entity to illustrate above. Figure 2.1 shows a Book entity with some attributes. As you can see, some of it are annotated (id, title and description), while some is not.

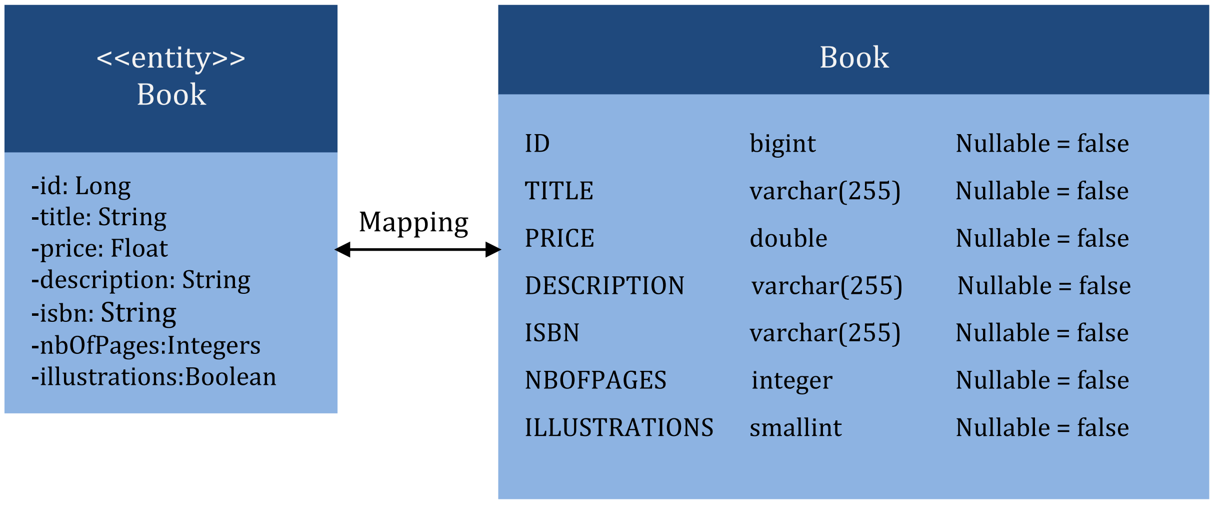
```
@Entity(name = “Book”)
public class BookEntity implements Serializable {

        @Id
        @GeneratedValue(strategy = GenerationType.AUTO)
        private long Id;
        @column(nullable = false)
        private String title;
        private Float price;
        @column(length = 2000)
        private String description;
        private String isbn;
        private Integer nbOfPage;
        private Boolean illustrations;
        private int CopyNumber;

        …….
}
```

Figure 1: A Simple Book Entity

To be recognised as an entity, the Book class must be annotated with *@javax.persistence.Entity* (or in short *@Entity*). The *@id* annotation is used to denote the primary key and the value of this identifier is automatically generated by the persistence provider (*@GeneratedValue*). The *@column* annotation is used in some attributes to customise the default column mapping (title becomes not nullable, and description has a length of 2000 characters). The persistence provider will then be able to map the Book entity to a Book table as shown below.



<<entity>>
Book
-id: Long
-title: String
-price: Float
-description: String
-isbn: String
-nbOfPages:Integers
-illustrations:Boolean
Mapping
Book
ID bigint Nullable = false
TITLE varchar(255) Nullable = false
PRICE double Nullable = false
DESCRIPTION varchar(255) Nullable = false
ISBN varchar(255) Nullable = false
NBOFPAGES integer Nullable = false
ILLUSTRATIONS smallint Nullable = false

Figure 2: Book Entity is mapped to a Book Table

4.4 Entity Manager

JPA allows users to map entities to database and also to query those using different criteria. The central piece of the API responsible for orchestrating entities is the Entity Manager (EM). Its role is to manage entities, read from and write to a given database and allow simple CRUD operations. The following snippet of code shows how to create an EM and persist a Book entity.

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("chapter02PU");
EntityManager em = emf.createEntityManager();
em.persist(book);
```

The Entity Manager with Persistence Context keeps track of the changes done to the managed entity instance. The method *em.flush()* is used to update the changes to database, *em.close()* is used to close the PersistenceContext and EntityManager, *em.find*(Entity object) is used to find the entity instance in database and *em.remove*(Entity Object) to remove entries in database.

The diagram shown below is the updated version with the data layer change as follows. JPA allows us to do persistence and relationships easily.

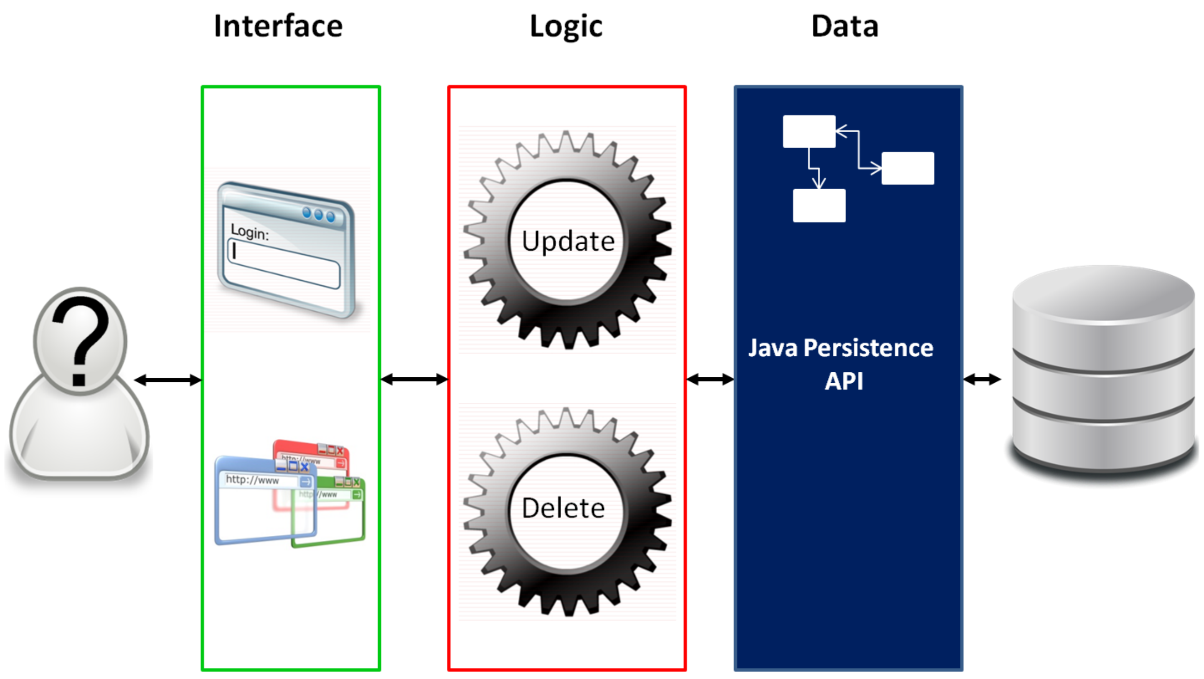


Figure 3: Application Diagram

5 Logic Layer

Now that we have looked at the data layer, let's move on to the logic layer. To create functionalities for the logic layer, we used Enterprise Java Beans (EJB). EJB will be further discussed in the next few sections.

5.1 EJB Overview

"Enterprise Bean is the server side deployable component which belongs to EJB technology and it can be loaded to the enterprise application. Enterprise beans are used to build the business logic of the Enterprise system. The business logic is the code encapsulates the purpose of the application. "[1]

We use EntityManager and the persistence context inside Enterprise Beans to update the database. A persistence context is a set of entity instances in which for any persistent entity there is a unique entity instance. Within the persistence context, the entity instances and their lifecycle are managed. The Entity Manager API is used to create and remove persistent entity instances, to find entities by their primary key, and to query over entities.

5.2 EJB Container

When we design our Enterprise System, the developer must handle the truncation management, security and clustering services. We call all those services as a middleware service where middleware stands for computer software that provides services to software applications beyond those available from the operating system. In Enterprise Java Bean technology, there is one component called the EJB Container, which is responsible for managing Enterprise Java Beans. The following list is some of the important service:

- Security
- Transaction management
- Remote accessibility
- Resource and life cycle management
- Clustering and load-balancing
- Concurrent requests support

EJB container also provides us the scalability and availability of Enterprise Beans:

Availability:

The Enterprise Java Bean Technology uses the internet protocol RMI-IIOP to allow interaction of Enterprise Beans and its container. EJB container is in charge of creating, deleting, maintaining the lifecycle of Beans and also the connection of Enterprise Beans and Client Object. Therefore we say that EJB container provides us Availability.

Scalability:

If there is a sudden increase of the number of clients which uses the Beans, EJB container would increase the number of Enterprise Beans in its Beans pool and decrease the number of Beans when the number of client accessing to the system decrease. So we achieve the Scalability of our server with the use of EJB container.

5.3 Benefits of EJB

Simplicity:

As the EJB container provides the system level service, the development of Enterprise Bean become easier and developers could focus mainly on building the functionality of the Enterprise System. The reason is that we do not need to worry about the middleware service such as transaction management and security authorization.

Light-weight Client:

Since we develop all the needed functionality as an Enterprise Beans, the Client class would be simpler and lighter. The benefit would be significant for small storage and low processing power devices.

Portability:

The Enterprise Beans are portable as we can move the beans from one server to another. Furthermore, we can add more Enterprise Beans which handle different Business Logic to the servers to achieve more functionality in the future.

5.4 Classifications of EJB

We classify the Enterprise Beans to two types:

- Session Bean

- Message-Driven Bean

We will illustrate the two Enterprise Bean in the following sections.

5.4.1 Session Beans

"A session bean encapsulates business logic that can be invoked programmatically by a client over local, remote, or web service client views."[1]

When a client interacts with the Session Bean, this is what will happen;

- The client can invoke the method of Session Bean,
- The Bean process the method with the arguments provided by the client,
- The Bean returns the result to client.

Since the Session Bean is short-life and it's reusable, the container could instantiate when the client make request on the server, and destroy the bean after the activation of client. We say the Session Bean is non-persistent. Moreover, the Session Beans are in memory object which would not be retrieved again if the server is down.

There are two types of Session Bean:

- Stateful Session Bean
- Stateless Session Bean

5.4.1.1 Stateful Beans

"Stateful session bean is the Enterprise Bean that maintains the client information throughout the whole accessing state of a client."[2]

The intuition of designing Stateful Session Bean is that some business process requests multiple method invocation and transaction. For example, when the client performs an online shopping, the state of shopping cart should be recorded throughout the purchasing session. In order to achieve it, we use a set of variable to maintain the state of the Bean.

Due to the limitation of memory, we would put the idle Stateful Session Bean to the secondary memory and restore the Stateful Session Bean back when the client performs a new task. The EJB container performs those jobs of a Stateful Session Bean.

The code below shows the shopping cart which need to be maintained by only one client, thus we use Stateful Session Bean to module the shopping cart. The *@Stateful* indicates that the bean is Stateful and the user's state would be maintained if the client is connected to our Enterprise System.

```
@Stateful
public class ShoppingCartBean implements shoppingCartRemotes{
…….
          public void addItem(Product item){
                    product.addElement(item);
          }
          …….
}
```

Figure 4: Snippet of ShoppingCart Class

5.4.1.2 Stateless Beans

"Stateless Session Bean is the Enterprise Bean that does not maintain any client information across method call. "[2]

The Stateless Session Bean instances are indifferent and equivalent for all clients. Thus, the Stateless Session Bean could be destroyed after method invocation or remained in the pools of session bean which is maintained by EJB container. The pool of Stateless Session Bean is maintained because we want to achieve high server performance as there are constant numbers of client doing the same tasks simultaneously.

```
@Stateless
public class TheLirarySystemManagerBean implements TheLibrarySystemManagerRemote{
…….
}
```

The code fragment above is how we define a Stateless Session Bean; the *@stateless* annotation indicates the java class is a Stateless Session bean.

```
@Stateless
public class TheLirarySystemManagerBean implements TheLibrarySystemManagerRemote{

        @EJB
        LoggingBean logging;
        @PersistenceContext
        EntityManager em;
        @RolesAllowed("admin")
        public boolean createBook(String isbn, String titleName, String author, String publisher, int
                copyNumber) {
        this.title = em.find(TitleEntity.class, isbn);
        if(this.title == null) {
                title = new TitleEntity();
                title.create(isbn, titleName, author, publisher);
                book = new BookEntity();
                book.create(copyNumber, title);
                title.addBook(book);
                book.setTitle(title);
                em.persist(title);
                em.persist(book);
        }else {
          book = new BookEntity();
          book.create(copyNumber, title);
          title.addBook(book);
          book.setTitle(title);
          em.persist(book);
        }
        logging.log("A book named "+title+" has been added.");
        return true;
}
```

Figure 5: CreateBook Method

The code above (which is defined in our Stateless Session Bean) could be used by the client instance. The following code fragment is used to define an EntityManager that is used to persist the entity instances to database.

@persistenceContext Entity Manager em ;

The rest of the code is the same as how we code a method in POJO (Plain Old Java Object).

5.4.2 Remote Business Interface

We must define a business interface before we code our Session Bean. The business interface encapsulates only the method that the client can invoke, so all the method should be declared with public access modifiers. The Session Bean could then implement the business interface. In order to let the Beans be available to all clients, we put the *@Remote* annotation before the interface declaration. If the Beans are available for the clients in the same container or processors, we could use *@Local* annotation instead of *@Remote* annotation.

The code below is the one of our interface class.

```
@Remote
public interface TheLiarySystemManagerRemote {
        public void createLibMember(String memberId, String name, String password, String
                        String department, String faculty, String phoneNum);

        public boolean terminate(String id);

        public boolean createBook(String isbn, String titleName, String author, String publisher,
                        int copyNumber);
        public List<BookState> getBooks();
        public List<BookState> getBooks(String memberId);
        public boolean deleteBook(long bookId);

        public boolean borrowBook(long bookId, String memberId);
        public boolean returnBook(long bookId, String memberId, Date date);
        public boolean renewBook(long bookId, String memberId);
        public boolean createReservation(long bookId, String memberId);
        public boolean deleteReservation(long bookId, String id);
…….
}
```

Figure 6: TheLibrarySystemManagerRemote.java

5.4.3 Message Driven Beans

"A Message-Driven Bean is an Enterprise Bean that allows Java EE applications to process messages asynchronously. " [1]

Message-Driven Bean does not have a local or remote interface and the client do not need to know the location of Bean to invoke desired functionalities.

Message-Driven Beans utilize the Java Message Service technology to achieve asynchronous invocation of Enterprise Java Beans. Let us study some of the basic Java Message Service in the following section.

Now let us see how we a Message-Driven Bean are constructed. First of all, the Bean does not have any interface and we use *@MessageDriven* annotation to indicate that the Bean is a Message-Driven Bean. The Bean uses JMS (Java Messaging Service) to implement the invocation process between user and the Bean.

```
@MessageDriven(mappedName = "jms/PaymentProcessingMDB", activationConfig = {

public class TheLiarySystemManagerBean implements TheLibrarySystemManagerRemote{
        @ActivationConfigProperty(propertyName= "acknowledgeMode", propertyValue =
                "Auto-acknowledge"),
        @ActivationConfigProperty(propertyName= "destinationType", propertyValue =
                "javax.jms.Queue")
        })

public class PaymentProcessingBean{
…….
}
```

Figure 7: TheLibrarySystemManagerRemote.java

5.4.4 Differences between the Beans

When client invoke a method of a Session Bean, the client communicate with the Bean synchronously; whereas when a client invoke a method in the Message-Driven Bean, the communication is asynchronous. In synchronous communication, the client must wait for the Session Bean to finish processing (the client is blocked) before accessing another Bean. On the other hand, when a client communicates with a Message-Driven Bean, the client can send a request to the Bean and proceed with other tasks immediately (the client is not blocked) without waiting for the response from the Bean. Once the Bean is done processing, it will then send the response back to the client.

Moreover, Message-Driven Bean's instances retain no data or conversational state for a specific client. All instances of the Bean are equivalent, allowing the EJB container to assign a message to any Message-Driven Bean instance. The container can pool these instances to allow streams of messages to be processed concurrently. On the other hand, the Stateful Session Bean maintains client state and could not use Stateful Session Bean concurrently.

5.5 Functionalities Provided by the Container

As we have seen in the example before there is a lot of annotations that we can use to customize the behaviours of our beans.

5.5.1 Security

Java EE provides security. Just by using annotations, you can secure a Bean or a function. Users have roles, therefore when you control access you're a just giving access to a role. You can use the annotation on a function or a Bean to limit the use of the users who have the role a_role :

```
@RolesAllowed("a_role")
```

5.5.2 Transactions

By default Java EE is protecting your data integrity with a transaction mechanism. Basically, if there is an error or an exception during the execution of one of your function, the container will rollback all the modification made. What is really impressive is that it also works when you are using remote Beans. The best example is a fund transfer system which interacts with two banks; normally you think of it as a complex transaction system. If there is a problem with one of the bank, you will have to cancel and rollback the whole transaction in both banks. Normally, huge and complex codes are needed with the entire 'if else' block to check the system. However with Java EE everything is automatic. You can still customize with some annotation the exact behaviour of the transaction in order to improve performance or get your exact need.

The diagram shown below is the updated version with the logic later changed.

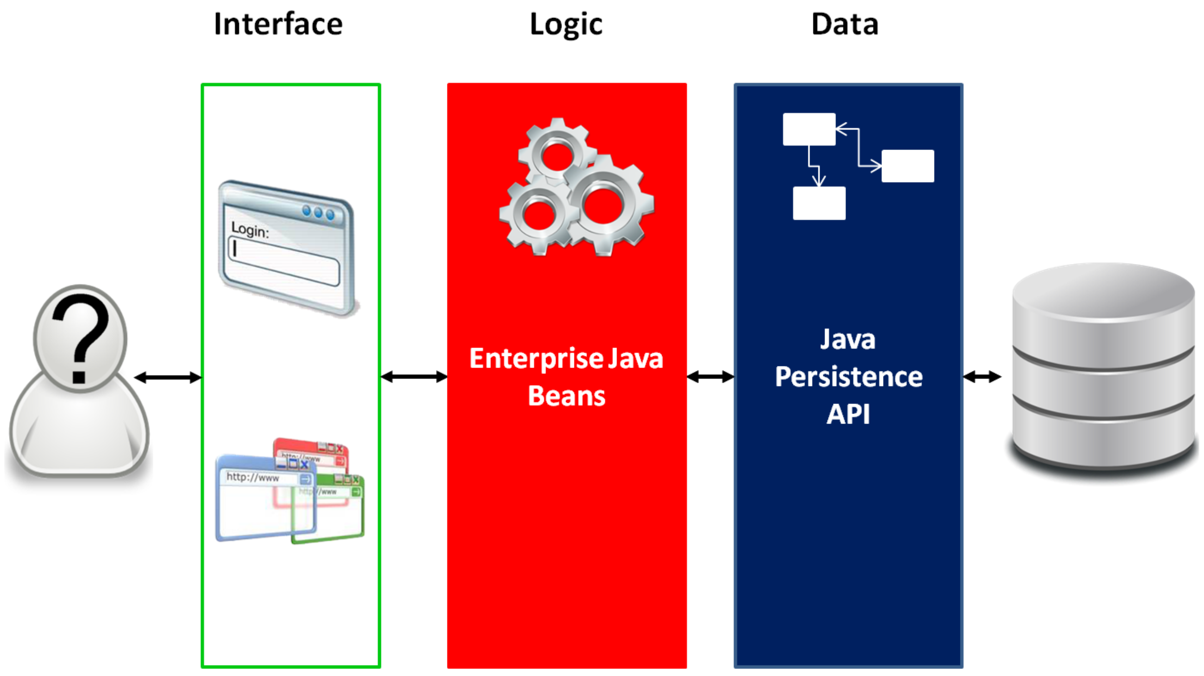


Figure 8: Updated Application Diagram

6 Creating the Interface

As we already did the most important parts which are the data and functions, we also need a User Interface for the user to interact with the system. The remaining section will be mainly focusing on how to create User Interface using Java.

In order to display results coming from the database and main program, a graphical representation is required and therefore a User Interface (UI) needs to be implemented. In this section, we are going to introduce you with the new standard framework, developed though Java Community Process (JCP), Java Server Faces (JSF). JSF offers easy way to build UI for java web applications. The main idea in JSF is assembling reusable components in a page.

"It is like a toolbox that is full of ready to use components where you can quickly and easily add and reuse these components many times in a page and capture events generated by actions on these components."[3]

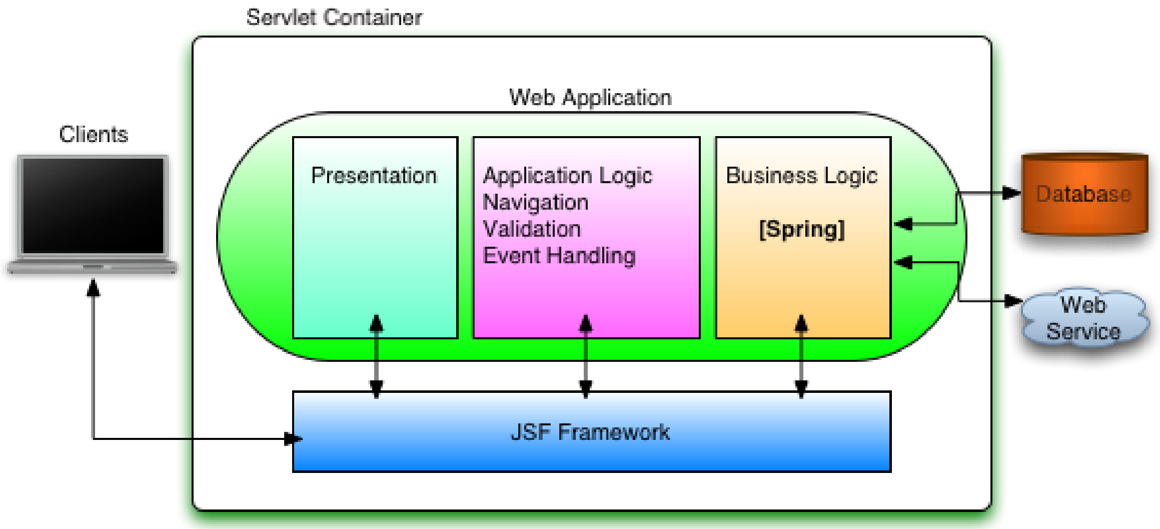


Figure 9: Web Application Architecture

In the web application's architecture (as shown in the Figure 9), we can clearly see how JSF fits into it.

As we mentioned before, JSF provides easy development of web applications based on Java technologies. Let us list some benefits of using JSF:

- It provides standard, reusable components for creating UI for web applications.
- It provides many tag libraries for accessing and manipulating the components.
- It automatically saves the form data and repopulates the form when it is displayed at client side.
- JSF encapsulates the event handling and component rendering logic
- JSF is a specification and vendors can develop the implementations for JSF.
- There are many GUIs available these days to simplify the development of web based application based on JSF framework.

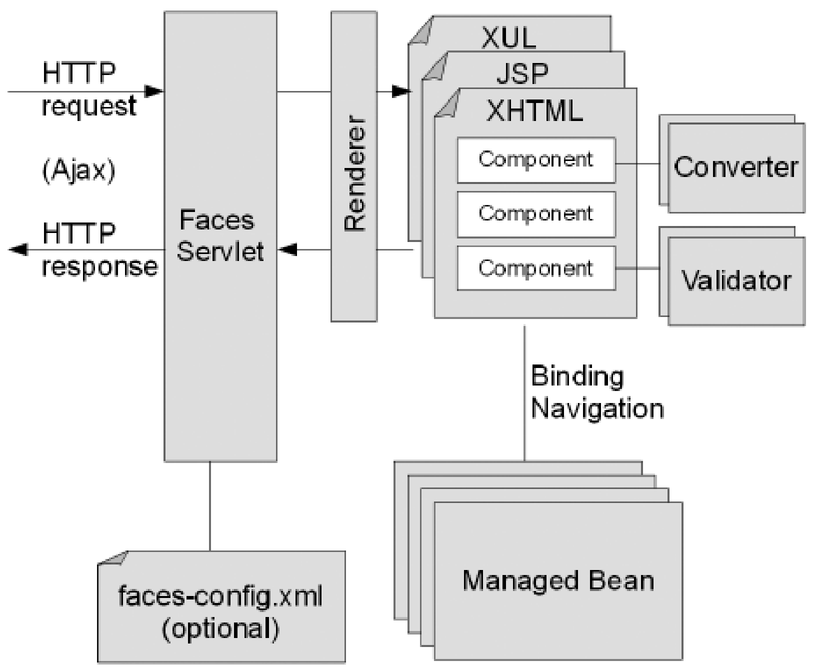


Figure 10: JSF Architecture

Figure 10 shows us the architecture of JSF. It consists of following parts.

- “FacesServlet”
- “Renderer”
- “Converter” and “Validator”

Examples of using Convertor and Validator

```
<h:outputText value = “#{bean.bookAddedDate}”>
<f:convertDateTime dataStyle = “full”/>
</h:outputText>
```

```
<h:inputText value = “#{registrationFormBean.validateEmail}”/>
```

- **“**Managed Bean” **–** It synchronizes values with components. In our general example of library system we can give the following code fragment which will use Managed Bean:

```
<h:inputText value = “#{bookControl.bookname}”/>
<h:commandButton value = “Create” action = “#{bookControl.CreateBook}”/>
```

First command takes a name from an input and puts it to the *“bookname”* property of managed bean *“bookControl”.* Afterwards, they are synchronized together.

The last, but not the least part of JSF architecture is **“Pages and Components”**. When most of the job is done the last thing that remains is to send the page to the client. For that JSF supports display technology called Java Server Pages (JSP). JSP is the technology which aims to develop dynamically generated web pages based on HTML or XML. JSP is the same as PHP, except that it uses the Java language. All the HTML and Java codes are to put inside .jsp extension file and it can be run to serve as dynamic web page. Java codes are to put inside “**<%**” and “**%>**” tags. In our following example we will show you how JSP is used in Login Page in *Figure 11*.

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN
"http://www.w3.org/TR/html4/loose.dtd">


<html> <head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Library CS4217 - Member Login</title>
</head>
<body> <center>
<h1>The Library System</h1>
<% if (session.getAttribute("memberid") != null) {%>
<p>You have already logged in.</p>
<% } else {%>
<% if (request.getAttribute("loginFailed") != null) {%>
<p>Login Failed. Try again!</p><%}%>
<form action="MemberLogin" method="POST">
<table><tr> <td align="right">ID:</td>
<td align="left"><input type="text" name="id" size="20" /></td></tr>
<tr> <td align="right">Password:</td>
<td align="left"><input type="password" name="password" size="20" /></td>
</tr></table><br />
<input type="submit" value="Submit" />
</form>
<% }%>
</center></body></html>
```

Figure 11: loginPage.jsp

This particular example of Login Page accepts login and password information on corresponding fields and gives a result whether logging in was successful or failed.

With an explanation of Java Server Faces and Java Server Pages, we now introduce programming language called “Servlet”.

Servlet is a small program that runs on a server. It is usually a Java applet that runs within a Web server environment. Java servlets are becoming increasingly popular as an alternative to Common Gateway Interface (CGI) programs. The biggest difference between the two is that a Java applet is *persistent.* This means that once it is started, it stays in memory and can handle multiple requests whereas a CGI program disappears once it has fulfilled a request. The persistence of Java applets makes them faster because there's no wasted time in setting up and tearing down the process.

In Java the Servlet has 4 types of Life Cycle such as:

- **init( )** – executes once when the servlet is first loads and will not be called for each request
- **service( )** – for each request this method is called in a new thread by server. It is called to process HTTP requests.
- **doGet( )**, **doPost( )**, etc – those methods handles GET, POST and other requests done by a user. These methods get dispatched by **service( )** method.
- **destroy( )** – whenever server wants to delete servlet instance it calls this method.

Now we would like to show you a very simple program called "HelloWWWServlet.java" which prints "Hello WWW" in the web page with all its standards.

```
package hall;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWWW extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
                                "Transitional//EN\">\n" +
                "<HTML>\n" +
                "<HEAD><TITLE>Hello WWW</TITLE></HEAD>\n" +
                "<BODY>\n" +
                "<H1>Hello WWW</H1>\n" +
                "</BODY></HTML>");
  }
}
```

Figure 12: HelloWWWServlet.java

In this code program first sets the content as *"text/html"* then prints "Hello WWW" as a title in the web page. This is the way Servlets are used in modern database driven webservers.

So we have discovered what is the JSF technology, what is it used to, and what are the components of it. We briefly discussed each of the components and its purpose in architecture. Let us then look at the final application diagram.

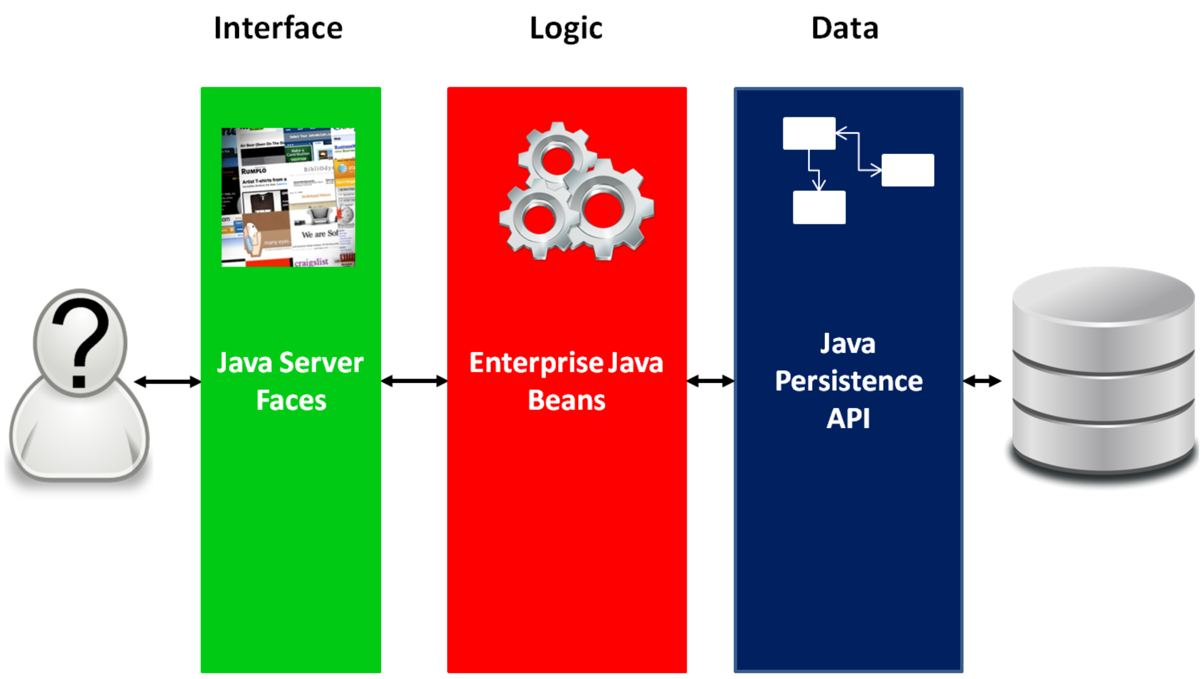


Figure 12: Finalised Architecture Diagram

7 CONCLUSION

As enterprise systems tend to become more and more complex, system designers must ensure the scalability of the systems and how they communicate with each other. This task requires more and more effort both in terms of business functionalities and regarding the technologies chosen. Using Java EE will help companies to be able to create an Enterprise System easily and effectively. Since Java EE is a very huge topic, we hope this small chapter will allow you to create a small Enterprise System. We have covered the major steps which will help you and allows you to look into the smaller details on your own.

8 RESOURCES

Amit Bahree, D. M. (2007). *Pro WCF: Practical Microsoft SOA Implementation [Paperback].* Apress.

Chappell, D. (March, 2010). *Introducing Windows Communication Foundation in .NET Framework 4*. Retrieved 9 March, 2011, from http://msdn.microsoft.com/library/ee958158.aspx

Corporation, O. (n.d.). *The Java EE 5 Tutorial*. Retrieved 10 March, 2011, from http://download.oracle.com/javaee/5/tutorial/doc/bnazq.html

Eichengreen, B. (May/June, 1999). *One Economy, Ready or Not: Thomas Friedman's Jaunt Through Globalization*. Retrieved 6 March, 2011, from http://www.foreignaffairs.com/articles/55017/barry-eichengreen/one-economy-ready-or-not-thomas-friedman-s-jaunt-through-globaliz

Erl, T. (2007). *SOA Principles of Service Design.* Prentice Hall.

Goncalves, A. (2009). *Beginning Java(TM) EE 6 with GlassFish(TM) 3: From Novice to Professional.* Apress, Inc.

Hewitt, E. (2009). *Java SOA Cookbook.* O'Reilly Media.

Jane Laudon, K. L. (2007). *Essentials of Management Information Systems.* Prentice Hall.

Judith Hurwitz, R. B. (2007). *Service Oriented Architecture for Dummies.* Wiley Publishing, Inc.

Kalin, M. (2009). *Java Web Services: Up and Running.* O'Reilly Media, Inc.

Klein, S. (2007). *Professional WCF Programming: .NET Development with the Windows Communication Foundation.* John Wiley & Sons.

9 References

[1] Corporation, Oracle. *The Java EE 6Tutorial.*

http://docs.oracle.com/javaee/6/tutorial/doc/.

[2] POO, Danny C.C. *Learn to program enterprise JAVABEAN*

Thomson Learning, 2007

[3] Rose India. *What is Java Server Faces?*

http://www.roseindia.net/jsf/whatisjsf.shtml

Goncalves, Antonio. *Beginning Java(TM) EE 6 with GlassFish(TM) 3: From Novice to Professional.*

Apress, Inc., 2009.