An Introduction to the Enterprise JavaBeans technology and
Integrated Development Environments for implementing EJB applications

Daniela Novak

Vienna University of Economics and Business Administration
2006
# Content

Content ........................................................................................................................................2

List of Figures ................................................................................................................................5

List of Tables ...................................................................................................................................8

Acronyms .........................................................................................................................................9

1 Introduction ................................................................................................................................11
   1.1 Motivation and goal .................................................................................................................11
   1.2 Thesis structure ........................................................................................................................11

2 Overview of Enterprise Java Beans ............................................................................................13
   2.1 Overall Goals of Enterprise JavaBeans .................................................................................15
   2.2 History of EJB Releases ..........................................................................................................16
   2.3 Developments, which influenced the origin of EJB technology ...........................................18

3 Overview of Java EE 5 ................................................................................................................21
   3.1 Introduction to Java EE 5 Specification ..................................................................................21
       3.1.1 The Java EE application architecture ...........................................................................22
       3.1.2 The way to Java EE .......................................................................................................26

4 EJB – Fundamentals ...................................................................................................................27
   4.1 What is an Enterprise Bean ....................................................................................................27
   4.2 Benefits and weak spots of Enterprise Beans ........................................................................28
   4.3 When to use Enterprise Beans? .............................................................................................29
   4.4 Types of Enterprise JavaBeans ............................................................................................29
   4.5 The EJB Container ................................................................................................................30
   4.6 EJB Roles ...............................................................................................................................31
   4.7 How to develop an EJB 2.1 Component .................................................................................33
   4.8 Ease of development in EJB 3.0 ............................................................................................34

5 Detailed View on EJB ..................................................................................................................36
   5.1 Types of Enterprise Beans .....................................................................................................36
       5.1.1 Session Beans ................................................................................................................37
5.1.2 Entity Beans in EJB 2.1 ............................................................... 43
5.1.3 Persistent objects in EJB 3.0 ....................................................... 51
5.1.4 Message-Driven Beans ........................................................... 52
5.2 Defining Client access with Interfaces ......................................... 55
5.2.1 Remote clients ........................................................................ 55
5.2.2 Local clients .......................................................................... 57
5.2.3 Web Service clients ................................................................. 57
5.3 Contents of Enterprise Beans ......................................................... 58
5.4 Naming conventions .................................................................... 59
5.5 Enterprise JavaBeans Query Language ........................................ 60
  5.5.1 Simplified Syntax of EJB QL ....................................................... 61
5.6 Java Persistence Query Language ............................................... 61
  5.6.1 Simplified Syntax ..................................................................... 62
6 Integrated Development Environments for EJBs ................................. 63
  6.1 Use of Integrated Software Development Environments .................... 64
  6.2 Selected IDEs for EJB development .............................................. 72
    6.2.1 NetBeans 5.0 ........................................................................ 72
    6.2.2 IBM Rational Application Developer for WebSphere Software .... 76
    6.2.3 Sun Studio Enterprise 8 .......................................................... 79
    6.2.4 Eclipse with Web Tools Platform 1.5.1 ................................. 82
7 Example Implementation - The Customer Book ..................................... 85
  7.1 The Customer Book application with NetBeans 5.0 ......................... 88
    7.1.1 Prerequisites ........................................................................ 88
    7.1.2 Creation of the Enterprise Application Project ......................... 88
    7.1.3 The Container managed Entity Bean – Customer ....................... 89
    7.1.4 The Session Bean – Customer Facade ..................................... 91
    7.1.5 The Web Module ................................................................. 94
    7.1.6 Building and Deploying the application ................................... 95
  7.2 The Customer Book application in RAD ......................................... 98
    7.2.1 Prerequisite .......................................................................... 98
    7.2.2 Creation of the Enterprise Application Project ......................... 99
    7.2.3 The Container managed Entity Bean ................................... 100
    7.2.4 The Session Bean – Customer Facade ................................. 102
7.2.5 Testing EJBs with the EJB Universal Test Client ............... 104
7.3 Compendium ........................................................................................................ 105

8 EJB IDEs for EJB 3.0 .................................................................................................. 107
8.1 NetBeans 5.5 ........................................................................................................... 107
  8.1.1 Development of a Message Driven Bean in NetBeans 5.5 .......... 108
8.2 JBoss Eclipse IDE ................................................................................................... 120
8.3 Oracle JDeveloper 10g Release 3 ............................................................................. 122

9 Conclusion ................................................................................................................. 124

10 References ................................................................................................................. 127
List of Figures

Figure 1 – Multi-tiered applications [cf. JATU06] .................................................. 22
Figure 2 – Web tier and Java EE applications [JATU06] ............................................. 25
Figure 3 – Business and Enterprise Information System tier [cf. JATU06] .............. 25
Figure 4 – Types of Enterprise JavaBeans ............................................................... 36
Figure 5 – EJB 2.1 - Life cycle of a stateless session bean [cf. JATU05] ............... 39
Figure 6 – EJB 3.0 - life cycle of a stateless session bean [cf. JATU06] ............... 40
Figure 7 – EJB 2.1 – life cycle of a stateful session bean [cf. JATU05] ................. 41
Figure 8 – EJB 3.0 – life cycle of a stateful session bean [cf. JATU06] .............. 42
Figure 9 – EJB 2.1 – life cycle of an Entity Bean [cf. JATU05] ................................. 45
Figure 10 – EJB 2.1 - Life cycle of a message driven [cf. JATU05] ........................ 54
Figure 11 - EJB 3.0 - life cycle of a stateless session bean [cf. JATU06] ............... 54
Figure 12 – EJB 2.1 - Interfaces for an EJB with remote access [cf. JATU05] . 56
Figure 13 – EJB 3.0 – Interfaces for EJB with remote clients [cf. JATU06] ....... 56
Figure 14 – Structure of the EJB JAR [cf. JATU05 and JATU06] ............................ 59
Figure 15 – Graphical user interface of NetBeans 5.0 ......................................... 73
Figure 16 – Graphical user interface of Rational Application Developer 6.0 ..... 76
Figure 17 – Graphical User Interface of Sun Java Studio Enterprise 8 .......... 80
Figure 18 – WTP – JST project structure ............................................................... 83
Figure 19 – Graphical User Interface of Eclipse 3.2.1 with WTP 1.5.1 .......... 84
Figure 20 – ER-diagram for the Customer Book application .............................. 86
Figure 21 – Class diagram for the Customer Book application .........................87
Figure 22 – Create the Enterprise Application project........................................89
Figure 23 – Create the CMP Entity Bean..........................................................89
Figure 24 – setting the JDBC connection for the Entity Bean .........................90
Figure 25 – Adding the tables for the Entity Bean.............................................91
Figure 26 – Create the Session Bean (1)..........................................................92
Figure 27 – Create the Session Bean (2)..........................................................92
Figure 28 – Call Enterprise Bean......................................................................93
Figure 29 – Creating the servlet CustomerDetail..............................................94
Figure 30 – Specification of the relative URL....................................................96
Figure 31 – CustomerBook Servlet...................................................................97
Figure 32 – Configuration of the Derby database .............................................98
Figure 33 – Creating the Enterprise Application project.................................99
Figure 34 – Add new modules to the Enterprise Application project...............99
Figure 35 – Create EJBs from DB tables............................................................100
Figure 36 – Mapping EJBs to RDB .................................................................101
Figure 37 – Class diagram of the Entity Bean Customer...................................102
Figure 38 – Creation of the Session Bean .........................................................103
Figure 39 – Configuration of the Session Bean.................................................103
Figure 40 – Welcome page Universal Test Client.............................................104
Figure 41 – EJBs for testing in the UTC............................................................104
Figure 42 – Building the project NewsApp.......................................................110
List of Figures

Figure 43 – Creating the persistence unit .......................................................111
Figure 44 – Wizard for the POJO NewsEntity .................................................112
Figure 45 – Generation of getter and setter methods by refactoring ...............113
Figure 46 – Selection of the refactored getter and setter methods ...............113
Figure 47 – Performing the refactoring for getter and setter methods ..........114
Figure 48 – Create the MDB and adding to the project ...................................114
Figure 49 – Servlet for posting a new message .............................................119
Figure 50 – Displaying the new message .......................................................119
Figure 51 – User interface of JBoss IDE for Eclipse .....................................120
Figure 52 – Wizard for EJB 3.0 projects in the JBoss IDE .............................121
Figure 53 – Oracle ADF design pattern ..........................................................123
Figure 54 – User interface of JDeveloper .......................................................123
List of Tables

Table 1 – Types of Enterprise JavaBeans and their purpose ..........................30

Table 2 – Differences between container-managed and bean-managed persistence........................................................................................................47

Table 3 – Contents of the enterprise beans ......................................................58

Table 4 - Naming conventions in EJB 2.1 [cf. JATU05] .................................59

Table 5 – Naming conventions in EJB 3.0 [cf. JATU06]..................................60

Table 6 – Vendor type and technology of the selected IDEs ..........................72
## Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>AWT</td>
<td>Abstract Window Toolkit</td>
</tr>
<tr>
<td>CICS</td>
<td>Customer Information Control System</td>
</tr>
<tr>
<td>CORBA</td>
<td>Common Object Broker Interface</td>
</tr>
<tr>
<td>DCOM</td>
<td>Distributed Component Object Model</td>
</tr>
<tr>
<td>EIS</td>
<td>Enterprise Information System</td>
</tr>
<tr>
<td>EJB</td>
<td>Enterprise JavaBean</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>JAR</td>
<td>Java Archive file</td>
</tr>
<tr>
<td>Java EE</td>
<td>Java Platform, Enterprise Edition</td>
</tr>
<tr>
<td>J2EE</td>
<td>Java 2 Platform, Enterprise Edition</td>
</tr>
<tr>
<td>JMS</td>
<td>Java Message Service</td>
</tr>
<tr>
<td>JSE</td>
<td>Java Studio Enterprise</td>
</tr>
<tr>
<td>JSP</td>
<td>Java Server Pages</td>
</tr>
<tr>
<td>HTML</td>
<td>Hypertext Markup Language</td>
</tr>
<tr>
<td>http</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>MDB</td>
<td>Message-Driven Bean</td>
</tr>
<tr>
<td>MTS</td>
<td>Microsoft Transactional Server</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>QL</td>
<td>Query Language</td>
</tr>
<tr>
<td>RAD</td>
<td>Rational Application Developer</td>
</tr>
<tr>
<td>SCM</td>
<td>Software Configuration Management</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>TP</td>
<td>Transactional Processing</td>
</tr>
<tr>
<td>UDDI</td>
<td>Universal Description, Discovery and Integration</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>WSIL</td>
<td>Web Service Inspection Language</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Service Definition Language</td>
</tr>
<tr>
<td>XMI</td>
<td>XML Metadata Interchange</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
1 Introduction

This Master Thesis is concerned with the technology of Enterprise JavaBeans, the server-side component architecture of the Java Enterprise Edition platform, and the development environments for Enterprise JavaBean applications.

1.1 Motivation and goal

The motivation of this Master Thesis is to provide an insight into the concepts of Enterprise JavaBeans, and how the implementation of Enterprise JavaBean application may be proceeded. Additionally, the focus lies on the introduction of appropriate development environments for the implementation of EJB applications.

The two main goals of this Master Thesis are to answer the questions:

- What is the Enterprise JavaBeans technology?
- Which Integrated Development Environments may be useful for implementing the Enterprise JavaBeans technology?

To provide the answers to these questions, the specification of EJB will be analysed and information about the requirements on IDEs will be worked out.

1.2 Thesis structure

The motivations and goals of this work are described in chapter 1. It also describes the structure of this Thesis.

Chapter 2 and chapter 3 give an overview of the Enterprise JavaBeans technology, and introduce the underlying Java EE concepts.

After the basic information provided, chapter 4 steps into the fundamental EJB concepts, and provides information about the usage, benefits and roles in EJB development.
Detailed information about the different types of beans, client access, and additional features of the EJB technology can be found in chapter 5.

For finding out, which IDE is appropriate for EJB application development, chapter 6 introduces the main requirements on IDEs in general, and presents four different development environments for Java Enterprise applications, based on EJB 2.1.

Two of the four selected IDEs, from the previous chapter, NetBeans 5.0 and IBM Rational Application Developer for WebSphere Software 6.0, are the environments for the implementation of the sample application in chapter 7. This chapter should point out how to implement an EJB application with the support offered by these IDEs.

Chapter 8 introduces three selected Integrated Development Environments for the EJB 3.0 specification, which was released in May 2006, and provides a sample application of an EJB 3.0 message-driven bean, developed in NetBeans 5.5.

The last chapter concludes this paper, with a short compendium of the lessons learned in the previous chapters.
2 Overview of Enterprise Java Beans

The first part of this Thesis offers basic introduction to the Enterprise JavaBeans technology, its goals and its history.

“Enterprise JavaBeans (EJB) technology is the server-side component architecture for the Java 2 Platform, Enterprise Edition (J2EE) platform. EJB technology enables rapid and simplified development of distributed, transactional, secure and portable applications based on Java technology.” [JAEB06].

The Enterprise JavaBeans (EJB) standard is a component architecture for server-side components in Java. It is an agreement between components and application servers that enable any component to run in any application server. Enterprise beans are deployable, and can be imported and loaded into an application server, which hosts those components [cf. RoAm02].

Physically, EJB is

- A specification: A document which lays out the rules of engagement between components and application servers. It constricts how to program in view to interoperability.

- A set of Java interfaces: Components and application servers must be conforming to these interfaces. Since all components are written to the same interfaces, they all look the same to the application server. Therefore, the application server is able to manage anyone’s components.

The main advantages of Enterprise JavaBeans are:

1. Simplification
   Enterprise beans simplify the development of large, distributed applications. This is because the EJB container provides system-level services to enterprise beans. Thus giving the advantage to developers to concentrate on solving business logic. A container hosts the runtime for application components, supplies a range of services and the communication infrastructure.

2. Reusability
   EJBs are reusable components which could be implemented in other busi-
ness applications with similar requirements or functionalities. The existing enterprise bean may be reused and customized according to new requisites. Thus, much development time and unnecessary resources can be saved.

3. Portability

EJB components are Java server-side components that are based on standard distributed object technologies. That means that it can be implemented and deployed on any compliant J2EE server. Not only the enterprise beans of an application are able run on different machines, but their location will remain transparent to the client.

4. Scalability

The EJB technology is able accommodate a growing number of users; its applications can be distributed across multiple machines. The Enterprise JavaBeans technology achieves this by putting common data and business logic into a single set of EJB components. The EJB components control the access to back-end data and manage database locking internally.

5. Transactional and concurrent access capabilities

Transactions are required to ensure data integrity. Enterprise beans support transactions, the mechanisms that manage concurrent access of shared data. The EJB technology automatically handles complex services and much of the communication infrastructure.

To get a deeper insight why the Enterprise JavaBeans architecture was developed, a closer look to the parts of the name Enterprise, Java and Beans is recommended:

- **Enterprise** means, that EJBs are used in business critical applications, where transaction control, distribution, multi-access due to several users, security, persistence, performance, easy extensibility and internet integration play a central role.

- **Java** stands for a platform-independent programming language, which is consequently object-oriented and widely accepted in software-development.

- **Beans** stand for components. JavaBeans are very successful in front-tier components. The Enterprise JavaBeans technology enables this success for the usage in the server-side middle-tier.
2.1 Overall Goals of Enterprise JavaBeans

According to the Enterprise JavaBeans specification the EJB architecture has the following objectives [cf. JAEB06]:

- The Enterprise JavaBeans will be the standard component architecture for building object-oriented and distributed business applications in the Java programming language.

- The Enterprise JavaBeans architecture supports web services in development, deployment, and usage.

- The Enterprise JavaBeans architecture eases the development of business applications. The application developer needs not considering about complex low-level APIs like connection pooling, multi-threading, or low-level transaction and state management.

- As Enterprise JavaBeans applications are developed once, and can be deployed on multiple platforms without recompiling or source modification, the philosophy of the Java programming language – *Write Once, Run Anywhere* is followed.

- The architecture of Enterprise JavaBeans defines the contracts which enable tools from different vendors to develop and deploy components, which work together at runtime.

- The EJB architecture provides interoperability between enterprise beans and other components of the Java Platform Enterprise Edition (Java EE), as well as applications that are not written in the Java programming language.

- As the Enterprise JavaBeans architecture has the goal to be compatible with existing server platforms, the vendors are able to extend their existing products to support the EJB technology.
• The Enterprise JavaBeans architecture will be compatible with other Java programming language APIs and it will be compatible with CORBA\(^1\) protocols.

### 2.2 History of EJB Releases

The development of the Enterprise JavaBeans architecture had following stages [cf. JAEB06]:

The EJB specifications 1.0 and 1.1, which was finally released on 17\(^{th}\) December 1999, focused on the following aspects:

- Defined the different EJB roles that are required by the component architecture.

- Defined the client view and the developer’s view of enterprise beans.

- Defined the responsibilities of an EJB Container Provider. These responsibilities together make up a system that supports the deployment and execution of enterprise beans.

- Defined the structure of the EJB’s part of deployment, the *ejb.jar* file.

- The EJB 1.1 specification detailed the responsibilities of the different EJB roles and provided more support for application assembly and deployment.

---

\(^1\) The Common Object Broker Architecture (CORBA) is an object oriented middleware, which was developed by the Object Management Group, and which defines platform comprehensive protocols and services.
The Enterprise JavaBeans Release 2.0 enlarged the EJB architecture by adding following aspects and was finally released on 14\textsuperscript{th} August 2001:

- The type of the message-driven bean was introduced and the integration of EJB and the Java Message Service was defined.
- Local clients are able to access enterprise beans by providing a local client view, and support for efficient and lightweight access was added.
- The new architecture for container-managed persistence\textsuperscript{2} (CMP) was defined.
- Support for the management of relationships between entity beans was provided and a declarative query syntax, the EJB Query Language (EJB QL) for finder and select methods for CMP entity beans was introduced.
- Provided support for a run-as security-identity\textsuperscript{3}.

On 12\textsuperscript{th} November 2004 the EJB 2.1 specification was finally released and it had the following goals:

- Enabling enterprise beans to implement and utilize web services.
- Providing a container-managed timer service.
- Enhancing EJB QL with the addition of ORDER BY and aggregate operators.
- Extending the message-driven bean component type to other messaging types.

\textsuperscript{2} See chapter 5.1.2.5 - Container Managed Persistence – Bean Managed Persistence.
\textsuperscript{3} The definition of run-as in the security-identity section of the deployment descriptor enables the EJB to call all methods with a specified security role. The access authorisation rules of this specified security role are used for calling other EJBs or resources, too.
The current version of the EJB architecture – EJB 3.0 - was released on 2\textsuperscript{nd} May 2006 and redesigned the EJB architecture in many aspects.

- The Java Persistence API replaced the type of Entity Beans, and simplified the entity persistence, including support for light-weight domain modelling, including inheritance and polymorphism.

- Persistent Entities require no longer any interfaces.

- The EJB deployment descriptor was eliminated.

- EJB applications can be annotated by the Java language metadata annotations.

- Elimination of the requirement for EJB component interfaces and EJB home interfaces for session beans

- The EJB QL was extended by the query language for Java persistence, including the support for native SQL queries.

### 2.3 Developments, which influenced the origin of EJB technology

The following chapter lists a few stages in software development history, which had an influence on the on the formation of Enterprise JavaBeans [cf. Mons04].

**Traditional TP-monitoring\textsuperscript{4}**

Traditional TP-monitors on mainframes (e.g. CICS from IBM, or Novell's Tuxedo) are working with a three-tier-architecture

- The upper layer (first tier or front end) is responsible for the representation of data on the user-interface
- The middle layer (middle tier) contains the procedural application logic

\textsuperscript{4} Transactional processing (TP) monitoring is the management of transactions between multiple application tiers, like client, server, and backend systems.
The bottom layer (third tier or backend) provides the persistent storage of data.

Traditional TP-monitors are widely accepted and distributed, because software developers are liberated from infrastructural tasks, as transaction control, or the handling of multiple user access.

Application objects

In the next stage in the development process of the Enterprise JavaBeans technology the separation of data and features was derestricted and data and application logic was encapsulated in application objects. The result of this encapsulating was a middle tier which was more flexible and better to maintain.

Allocation between Clients and Server

After the separating data and application logic the separation of clients and server revolutionised the software development process. The client and the application objects are now able to run on different machines, with the advantage of scalable systems.

Application server

The application server handles the infrastructural tasks, like a traditional TP-monitor do, and additionally implements a server side component model, which is an architecture for allocated application objects. The first application server was the Microsoft Transaction Server (MTS) which used DCOM\(^5\) to handle the component service. The next generation of application servers used CORBA as service for the components, but they all used different server side component models, which made portability impossible.

This fact leads to the requirement for a standard for the server side component model. Now the Enterprise JavaBeans specification offers this standard, be-

\(^5\) Distributed Component Object Model (DCOM) is a protocol for network communication from Microsoft for components.
cause it defines the minimum features which have to be provided by an application server.
3 Overview of Java EE 5

As Enterprise JavaBeans are basic parts of the Java Platform, Enterprise Edition (Java EE) a closer look to the industry standard for developing portable, robust, scalable and secure server side Java applications is recommended.

3.1 Introduction to Java EE 5 Specification

Today companies need to extend their scope, lower their costs and reduce response times of their services to customers, employees, and suppliers.

To reach these goals, business applications which provide these services have to combine existing business applications with new business functions that deliver services to a broad range of users.

These services must reach following goals [cf. JAEE06]

- High availability – to meet the requirements of modern business environment
- Security – for protecting the privacy of users and the integrity of the Enterprise
- Reliability and scalability – business transactions have to be accurately and promptly processed

Mostly, enterprise services are designed as multitier applications. The middle tier combines existing Enterprise Information Systems (EIS) with new business functions and data of the new services.

"The Java Platform, Enterprise Edition (Java EE) reduces the cost and complexity of developing multitier, enterprise services. Java EE applications can be rapidly deployed and easily enhanced as the enterprise responds to competitive pressures." [JAEE06].
Java EE defines a standard architecture with following elements:

- Java EE Platform – standard platform for hosting Java EE applications
- Java EE Compatibility Test Suite – for verifying that a Java EE platform product is compatible with the Java EE platform standard
- Java EE Reference Implementation – for prototyping Java EE applications
- Java EE BluePrints – set of best practices for developing multitier, thin-client services

### 3.1.1 The Java EE application architecture

In the Java EE architecture application logic is divided into components. The various application components of Java EE applications are usually installed on different machines depending on the tier in the Java EE environment. Figure 1 shows the different components in their tiers.

![Figure 1 – Multi-tiered applications](cf. JATU06)

The client-tier runs on the client machine (HTML pages or application client), the web-tier components (JSPs) and the business-tier components (EJBs) run...
on the Java EE server and the Enterprise Information System (databases) are running on the EIS server [cf. JATU06].

Java EE applications are three-tier applications, because the components are distributed on three different machines: the client machines, the Java EE server machine, and the legacy machine at the back end.

**The Java EE components**

“A Java EE component is a self-contained functional software unit that is assembled into a Java EE application with its related classes and files and that communicates with other components.” [JATU06].

Following components are specified in the Java EE specification:

- Components running on the client: application clients and applets
- Web components running on the server: Java Servlets, JavaServer Faces, and JavaServer Pages.
- Enterprise components running on the server: Enterprise JavaBeans

“The difference between Java EE components and “standard” Java classes is that the Java EE components are assembled into a Java EE application, are verified to be well formed and in compliance with the Java EE specification.” [JATU06]. In production environments the Java EE components are running in and managed by the Java EE server.

**Client Components**

**Web Clients**

A web client consists of two parts, on the one hand of dynamic web pages and on the other hand a web browser, which renders the pages received from the server. Web clients are often called “thin clients”, because they usually do not query databases, execute complex business rules, or connect to legacy applications. While using a thin client, these operations are executed by enterprise beans on the Java EE server.
Applets

An applet is a small client application that executes in the Java virtual machine installed in the web browser. For executing applets the client requires the Java plug-in and a security policy file.

Application clients

The application client enables users to handle tasks which require a richer GUI than the one which can be provided by a markup language. Typically the GUI of the application client is created from the Swing or the AWT API\(^6\). Application clients written in other languages than Java can interact with Java EE 5 servers, too.

Web Components

Servlets, JavaServer Faces and/or JSP pages are the web components in Java EE. Servlets are Java classes which dynamically process requests and responses. The JavaServer Faces technology builds on servlets and JSP technology and provides a user interface component framework for Web applications. JavaServer Pages are text-based documents which work as servlets, but allow a more natural approach for creating static content. [cf. JATU06]

Figure 2 shows that the web tier, like the client tier, may include a JavaBeans component to manage input and send this input to the Enterprise JavaBeans which are held in the business tier.

\(^6\) Swing and Abstract Window Toolkit (AWT) are the common Java APIs for developing graphical user interfaces.
Business Components

The business logic that solves the need of a specific business domain such as finance, banking, or retail is handled by enterprise beans which are running in the business tier. As illustrated in figure 3 the enterprise bean receives data from the client, processes it, and sends it to the legacy system for storage. The enterprise bean also receives data from storage and sends it back to the client application.
Enterprise Information System tier

The Enterprise Information System (EIS) tier includes infrastructure such as mainframe transaction processing, database systems, or other legacy information systems, and handles EIS software.

3.1.2 The way to Java EE

The first generation of enterprise applications was monolithic, centralized and based on mainframe architecture.

In the mid of the 1980ies Client/Server architecture became popular in enterprise application development. Typically, this Client/Server architecture is a two-tier architecture. The client combined GUI (graphical user interface) and business logic, the server was used as data-base system. One of the main advantages of this architecture is the ease of development. The presentation logic and the business logic are located in the same process and therefore easy to handle for the developer.

The advantage of simplicity is the main disadvantage at the same time. Because of the combination of presentation and business logic, these applications become difficult to maintain, if these applications are reaching a specific size. They are reduced to one specific type of presentation and they have limited scalability.

The next step in software-architecture development was the implementation of three-tier architecture, which separates presentation logic and business logic. The presentation logic is still on the client, but the business logic is located on the server. This three-tier architecture has several advantages. The business logic can handle expensive resources, like database-connections and make them available for multiple clients. Therefore the application becomes more scalable. At the same time the security level increases, because the business logic is not longer on the client. The disadvantage of three-tier architecture lies in the complexity of development. The developer has to maintain distribution, security and multi-threading. To reduce the effort for maintaining the tasks named above, application server and application frameworks were developed.
4 EJB – Fundamentals

This chapter describes the fundamentals of the Enterprise JavaBeans technology. It offers an introduction what an Enterprise JavaBean is and shows the benefits and the weak spots of this technology in chapter 4.1 and 4.2. Chapter 4.3 informs about the application area of EJB technology and a basic view on the different EJB types is given in chapter 4.4. The tasks of the EJB container are described in chapter 4.5 and the chapters 4.6 and 4.7 offer a short insight on the development procedure of Enterprise JavaBeans. Chapter 4.8 presents the basic principles of the EJB 3.0 specification.

4.1 What is an Enterprise Bean

“The Enterprise JavaBeans architecture is a component architecture for the development and deployment of component-based distributed business applications. Applications written using the Enterprise JavaBeans architecture are scalable, transactional, and multi-user secure. These applications may be written once, and then deployed on any server platform that supports the Enterprise JavaBeans specification.” [JAEB06].

An Enterprise JavaBean is a Java EE component that implements the Enterprise JavaBeans architecture. Enterprise beans are running within an EJB Container which provides system-level services such as security and transactions to its enterprise beans. These services enable the developer quickly to build and deploy enterprise beans, which are the core of transactional J2EE applications [cf. JATU06].

An Enterprise JavaBean is a server-side component, written in the Java programming language that encapsulates the business logic of an application. The business logic is the code that fulfils the purpose of the application.
4.2 Benefits and weak spots of Enterprise Beans

Using enterprise beans can simplify the development of large, distributed applications, but the use of the EJB technology may not be appropriate for solving all possible problems of an enterprise business application. “In order to decide whether the EJB technology can fit a project, it is necessary first to understand its major advantages and disadvantages.” [KrGa03].

**Benefits**

One of the main advantages of the Enterprise JavaBeans technology is that the EJB container offers system-level services, such as transaction management and security authorization, and the developer needs only to concentrate on solving business problems.

As the beans contain all business logic, and not the clients, the client developer can focus on the presentation and needs not coding the routines that implements business rules or database access. This separation between presentation and business logic results in thinner clients, which may run on smaller devices.

Additionally to the advantages mentioned above, Enterprise JavaBeans are portable components. This offers the possibility to build new applications from existing beans. These new applications can run on any compliant Java EE server which uses the standard EJB APIs.

**Weak spots**

One of the main disadvantages of the Enterprise JavaBeans technology is that, due to the complex and thorough specification, the developers do not take full advantage of it. A common problem when developing EJ Bs is that some parts are misused and some other are reinvented, which leads to solutions which are hard to maintain and which do not focus on the advantages of this technology [cf. KrGa03].

The EJB specifications from 1.0 to 2.1 added complexities compared to straight Java classes. Every session bean consisted of at least three Java classes, and
entity beans required at least four classes, and a deployment descriptor was required. But since EJB 3.0, which was announced in May 2006 and had some considerably changes in the types of beans and the interfaces required, these problems are at least partly solved due to the use of the Java language metadata annotations, and some other facilities, like the omission of the need for interfaces for entity objects\(^7\).

### 4.3 When to use Enterprise Beans?

The implementation of Enterprise JavaBeans is reasonable if a business application has to meet one of the following requirements:

- The application must be scalable. To provide a growing number of users, the developer may need to distribute application components across multiple machines.

- Transactions must ensure data integrity. Enterprise beans support transactions, the mechanism that manages the concurrent access of shared objects.

- The application will have a variety of clients. Remote clients can easily locate enterprise beans, with only a little effort in coding.

The use of the EJB technology is not recommended when there is no need for scalability, security, or transaction management. EJB also may be overkill for simple applications or for applications which have only low numbers of read-only users.

### 4.4 Types of Enterprise JavaBeans

The EJB specifications, 2.1 and 3.0, distinct, according to their purpose and application area, between session beans, entity beans, and message-driven beans.

\(^7\) See chapter 4.8 - Ease of development in EJB 3.0.
Table 1 summarises the types of enterprise beans and gives a short overview of the purpose of each type. A detailed description of the range of application of these EJB types can be found in chapter 5.1.

<table>
<thead>
<tr>
<th>Enterprise Bean Type</th>
<th>EJB Version</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Session</td>
<td>2.1 and 3.0</td>
<td>Implements a Web service; performs a task for a client</td>
</tr>
<tr>
<td>Entity</td>
<td>2.1, replaced with persistent entities in EJB 3.0 8</td>
<td>Represents a business object that exists in persistent storage</td>
</tr>
<tr>
<td>Message-Driven</td>
<td>2.1 and 3.0</td>
<td>Acts as listener for the JMS API, processing messages asynchronously</td>
</tr>
</tbody>
</table>

Table 1 – Types of Enterprise JavaBeans and their purpose

4.5 The EJB Container

Enterprise JavaBean containers are responsible for managing the beans.

The container interacts with the beans by calling the enterprise beans’ required management methods when necessary. These management methods are the enterprise beans’ call back methods that the container, and only the container, invokes. The management methods allow the container to alert the beans when middleware events take place, such as when an entity bean is about to be persisted to storage.

The most important responsibility of the EJB container is to provide an environment in where enterprise beans can run. EJB containers house the enterprise beans and make them available for clients to invoke remotely.

---

8 Since EJB 3.0 persistent entities are not longer called entity beans, they are now called EJB 3.0 entities or persistence entities, which are lightweight persistence domain objects.
The EJB container acts as an invisible intermediary between the client and the beans. It is responsible for connecting clients to enterprise beans, performing transaction coordination, providing persistence, and managing the life cycle of an enterprise bean.

EJB containers are abstract entities. Neither the beans nor the clients, which call enterprise beans, are ever explicitly coded to the API of an EJB container.

4.6 EJB Roles

The Enterprise JavaBeans architecture defines six distinct roles in the application development and deployment life cycle. Each EJB Role may be performed by a different party. The EJB architecture specifies the contracts that ensure that the product of each EJB Role is compatible with the product of the other EJB Roles [cf. JAEB06].

1. Enterprise Bean Provider
   The Bean Provider is the producer of enterprise beans. His output is an \texttt{ejb.jar} file that contains one or more enterprise beans. The Bean Provider is responsible for the Java classes that implement the business methods; the definition of the bean’s home, component, and/or web service endpoint interfaces; and the bean’s deployment descriptor, which was required until the EJB 3.0 specification was released. The deployment descriptor includes the structural information (e.g. the name of the enterprise bean class) of the enterprise bean and declares all the enterprise bean’s external dependencies.

2. Application Assembler
   The Application Assembler combines enterprise beans into larger deployable units. The input of the Application Assembler is one or more \texttt{ejb.jar} files produced by the Bean Provider(s). The Application Assembler’s output is one or more \texttt{ejb.jar} files that contain the enterprise beans together with their application assembly instructions. The Application Assembler inserts the application assembly instructions into the deployment descriptors. The Application Assembler also combines enterprise beans with other types of application components (e.g. JSPs) when composing an application.
The EJB specification describes the case in which the application assembly step occurs before the deployment of the enterprise beans. However, the EJB architecture does not preclude the case that application assembly is performed after the deployment of all or some of the enterprise beans.

3. **Deployer**

   The Deployer takes one or more `ejb.jar` files produced by the Bean Provider or Application Assembler and deploy the enterprise beans contained in the `ejb.jar` files into the operational environment. This environment includes a specific EJB server and container. The Deployer must resolve all the external dependencies declared by the Bean Provider, and must follow the application assembly instructions defined by the Application Assembler. To perform this role, the Deployer uses tools provided by the EJB Container Provider. The Deployer's output is a set of enterprise beans that have been customised for the target operational environment and that are deployed in a specific EJB container.

4. **EJB Server Provider**

   The EJB Server Provider is a specialist in the area of distributed transaction management, distributed objects, and other lower-level system-level services. A typical EJB Server Provider is an OS vendor, a middleware vendor, or a database vendor.

   The EJB architecture assumes that the EJB Server Provider and the EJB Container Provider is the same vendor. Therefore, it does not define any interface requirements for the EJB Server Provider.

5. **EJB Container Provider**

   The Container Provider provides:
   
   – the deployment tools necessary for the deployment of enterprise beans, and
   
   – the runtime support for the deployed enterprise bean instances.

   From the perspective of the enterprise beans, the container is a part of the target operational environment. The container runtime provides the deployed enterprise beans with transaction and security management, network distribution of remote clients, scalable management of resources, and other ser-
vices that are generally required as part of a manageable server platform. The EJB Container Provider’s responsibilities defined by the EJB architecture are meant to be requirements for the implementation of the EJB container and server. Since the EJB specification does not architect the interface between EJB container and server, it is left up the vendor how to split the implementation of the required functionality between the EJB container and server.

6. System Administrator

The System Administrator is responsible for the configuration and administration of the enterprise’s computing and networking infrastructure that includes the EJB server and container. The System Administrator is also responsible for overseeing the well-being of the deployed enterprise beans application at runtime.

4.7 How to develop an EJB 2.1 Component

Typically following order of operations is used while developing an Enterprise JavaBean component:

1. Coding of the .java files that compose the enterprise bean: the component interfaces, home interfaces, enterprise bean class file, and any helper classes.
2. Coding or generating the deployment descriptor (not required since EJB 3.0).
3. Compiling of the .java files from step 1 into .class files.
4. Using the jar utility to create the ejb.jar file, containing the deployment descriptor and .class files.
5. Deployment of the ejb.jar file into the container in a vendor-specific manner, by running a vendor-specific tool or by copying the ejb.jar file into a folder where the container searches for ejb.jar files to load.
6. Configuration of the EJB server, to ensure that the ejb.jar file is properly hosted. This step is also vendor-specific and might be done through a Web-based console or by editing a configuration file.
7. Starting of the EJB container and confirmation that the ejb.jar file is loaded.
8. Optionally, providing a standalone `testclient.java`.

### 4.8 Ease of development in EJB 3.0

In May 2006 the new EJB 3.0 specification was finally released and it includes a lot of simplification for implementing Java Enterprise applications.

“The EJB 3.0 release of the Enterprise JavaBeans architecture provides a new, simplified API for the enterprise application developer. This API is targeted at ease of development and is a simplification of the APIs defined by earlier versions of the EJB specification.” [JAEB06].

“The purpose of the EJB 3.0 release is to improve the EJB architecture by reducing its complexity from the enterprise application developer’s point of view.” [JAEB06].

The EJB 3.0 APIs were defined to reach following goals:

- Simplification of the enterprise bean types.
- The Java language metadata annotations can be used to annotate EJB applications. This reduces the number of program classes and interfaces to implement manually by the developer, and eliminates the requirement of a deployment descriptor.
- The Java Persistence API offers the simplification of entity persistence, and includes support for light-weight domain modelling\(^9\), including inheritance\(^10\) and polymorphism\(^11\).
- Interfaces are not longer required for persistent entities.
- Elimination of the requirement for home interfaces for session beans.

---

\(^9\) A domain model is a model of the domain within an Enterprise processes its business.

\(^10\) Inheritance is a way to form new classes using pre-defined objects or classes where new ones take over old ones’ implementations and characteristics. It is intended to help reuse of existing code with little or no modification.

\(^11\) Polymorphism is the ability of a programming language to process objects differently depending on their data type. It is the ability to redefine methods for derived classes.
• EJB component interfaces for session beans are not longer required. The necessary business interface for a session bean can be a plain Java interface rather than an EJBOBJECT, EJBLocalObject, or java.rmi.Remote interface.

• The EJB QL of EJB 2.1 is extended by a query language for Java Persistence, with addition of dynamic query capability and support for native SQL queries.

The EJB 3.0 API specification is divided into three sections:

• The EJB 3.0 Simplified API, which provides an overview of the simplified API introduced in the EJB 3.0 release.

• The Java Persistence API, which defines the management of persistence and the full specification of the Java Persistence query language.

• The EJB Core Contracts and Requirements Specification defines the contracts and requirements for the use and implementation of Enterprise JavaBeans.

Where the EJB 3.0 made development easier, the existing EJB 2.1 APIs are still available for use in applications and components written in EJB 2.1 can still be used in conjunction with components written in the new EJB 3.0 APIs.

The differences of EJB 2.1 and EJB 3.0 will be highlighted in Chapter 5 “Detailed View on EJB”.
5 Detailed View on EJB

Chapter 5 and its sub-chapters offer a detailed view on Enterprise JavaBeans. It shows the different types of EJBs (Session Beans, Entity Beans and Message Driven Beans) in chapter 5.1 and introduces the interfaces of EJBs for client access in chapter 5.2. The parts of Enterprise Beans and the proposed naming conventions of the EJB specification are described in chapter 5.3 and 5.4. As the EJB specification offer a special query language a short view on the EJB QL is offered in chapter 5.5 and 5.6.

5.1 Types of Enterprise Beans

Enterprise JavaBeans are divided into three categories, where each type handles different tasks and has different responsibilities.

- Session beans representing work being performed for the client code that is calling it.

- Entity beans representing persistent data.

---

12 Entity beans have been replaced with persistent objects in the EJB 3.0 specification.
• Message-Driven beans handle asynchronous JMS messages.

The next chapters describe the types of Enterprise JavaBeans in detail.

5.1.1 Session Beans

The following section describes the features and the application area of session beans. It also gives a description of the two sub-types of the session beans, stateful and stateless, and their life cycle.

5.1.1.1 What is a session bean?

Session beans are reusable components that have conversations or sessions with clients.

Session beans are the workmen of the EJB components. They provide direct functionality, manage or control interaction with resources and other beans, and they act as a facade or compound for cooperating beans.

Session beans have a relatively short lifetime and life-cycle, depending on the client. They are non-persistent objects and live no longer than the client, and possibly less.

The container may reuse session beans, but from the client’s perspective, they are permanently gone.

5.1.1.2 Types of Session Beans

There are two different types of session beans, stateless and stateful session beans, which differ in the relationship with their clients. In EJB 2.1 the container knows the type of the session bean through a deployment descriptor entry, and in EJB 3.0 the state of the session bean is given by the annotation @Stateful or @Stateless in the session bean class.

Generally, stateless session beans are components that perform transient services, e.g. validation and processing credit card orders, and stateful session beans are components that are assigned to one single client and act as an extension on the server for that client, e.g. the customer's session at an online shopping web site.
5.1.1.3 When to use Session Beans

Session beans should be used in following circumstances:

- At any given time, only one client has access to the enterprise bean’s instance.
- The state of the enterprise bean existing only for a short period.
- The enterprise bean implements a web service.

Stateless session beans could be used to improve performance, when the session bean’s state has no data for a specific client, or if the bean performs a task for all clients (in the case of single method invocation).\(^{13}\)

If the bean’s state represents the interaction between the session bean and one specific client, or the bean needs to hold information about the client across method invocations, a stateful session bean is appropriate. If the bean manages a workflow of several enterprise beans, the stateful session bean is useful as well.

5.1.1.4 Stateless session beans

Stateless session beans do not support dialog with any client for longer than the duration of a specific method call. From the client perspective, the stateless session bean could be seen as a group of methods that use only local variables, remembering nothing from previous client interaction. Additionally, the container may serve the same instance of a stateless session bean to multiple clients or use different bean instances at different times to service one client’s request.

One consequence of the way stateless session beans work is, that there can only be a single `ejbCreate()` method, with no arguments. The container may choose to passivate a stateful session bean, but never does so with stateless

\(^{13}\) For example: the stateless session bean sends an email to confirm an online order.
session beans. These characteristics can offer greater scalability and performance than most other types of enterprise beans.

**Stateless session bean – life cycle**

As stateless session beans are never passivated, their life cycle has only two stages “does not exist” and “ready” for the invocation of business methods.

![Diagram of stateless session bean life cycle](image)

Figure 5 – EJB 2.1 - Life cycle of a stateless session bean [cf. JATU05]

Figure 5 illustrates the life cycle of a stateless session bean in EJB 2.1. A container performs following three steps to place a session bean in the ready state:

1. instantiates a bean using the `Class.newInstance()` method

2. invokes the bean’s `setSessionContext()` method

3. invokes the bean’s single `ejbCreate()` method

When the bean is no longer required, the container calls the bean’s `ejbRemove()` method. After this the bean is back in the Does Not Exist state and is destroyed.

There is no necessary obligation for a correlation in time between a client’s `create()` and `remove()` calls and a container’s `ejbCreate()` and `ejbRemove()` calls.

Additional the bean class must have a public, no-arg constructor a arrange `Class.newInstance()` invocations.

Also the container may support a pool of methods ready on the start up of the container (pool of ready-to-serve beans).
In EJB 3.0 the client obtains a reference to a stateless session bean and initiates its life cycle. The container performs any dependency injection\textsuperscript{14} and if there is the method annotated @PostConstruct, it will be invoked, too. Now the session bean is ready for processing its business methods. When the EJB container calls the method annotated @PreDestroy, the bean’s life cycle ends and the instance is ready for garbage collection then.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{life_cycle.png}
\caption{EJB 3.0 - life cycle of a stateless session bean \cite{JATU06}}
\end{figure}

5.1.1.5 Stateful session beans

Stateful session beans are related to a specific client and are able to save session state across multiple requests from a client for the duration of a bean’s lifetime. Because the stateful session bean needs to retain state and the link to one client, they are not as scalable as the stateless session bean.

**Stateful session bean – life cycle**

In addition to the stages “does not exist” and “ready” a stateful session beans has a third stage in its life cycle, which is called “passive”.

---

\textsuperscript{14} Dependency Injection – The source for references on resources is not longer required to be implemented by the developer. The developer annotates the positions in the EJB component, on where the EJB container injects the desired reference.
Figure 7 shows the steps in the life cycle of a stateful session bean in EJB 2.1. The container performs following three steps to place a stateful session bean in the ready state:

1. instantiates a bean using the `Class.newInstance()` method
2. invokes the bean’s `setSessionContext()` method
3. invokes the bean’s single `ejbCreate()` method

When the container decides to deactivate, or “passivate” the bean by moving it from memory to secondary stage, the EJB container invokes the `ejbPassivate` method. If the client invokes a business method while the bean is in the “passivate” stage, the container activates the bean by calling the bean’s `ejbActivate` method, and moves it into the ready stage.

There may be multiple `ejbCreate()` methods with different input arguments, just as an ordinary Java class might have multiple constructors.

When the client invokes the remove method, and the EJB container calls the bean’s `ejbRemove` method, the bean is on the end of its life cycle. The bean is now effectively destroyed and back in the Does Not Exist state.
All methods except the “create” and “remove” methods are invoked by the container.

Stateful session beans must be linked to a single client and maintain state with the client. In contrast, stateless session beans are easy to pool.

The life cycle of stateful session beans in EJB 3.0 is illustrated in Figure 8. The client starts the life cycle when obtaining a reference to a stateful session bean. Now, like for stateless session beans, the EJB container performs any dependency injection and invokes the methods annotated with @PostConstruct, if any. The bean is now ready to perform the business methods invoked by the client.

Like in EJB 2.1 the EJB container may decide to passivate the bean by invoking the method annotated @PrePassivate. If a client invokes a business method on the passive bean, the EJB container calls the method annotated @PostActivate, and moves it into the ready stage.

The bean’s life cycle ends when the client invokes a method annotated @Remove, then the EJB container calls the method annotated @PreDestroy and the bean’s instance is ready for garbage collection.

The code controls only the invocation of the method annotated @Remove, all other methods are invoked by the container.
5.1.2 Entity Beans in EJB 2.1

As entity beans are replaced by the Java Persistence API in EJB 3.0 this chapter handles only entity beans in EJB 2.1. The description of entity objects in EJB 3.0 is handled in chapter 5.1.3.

5.1.2.1 What is an Entity Bean

The main task of an entity bean is handling persistence data, for example the storage of data in relational databases.

Usually entity beans represent, in object-oriented presentation, an object in the real world.

The data of one entity bean accords to one line in one or more database-tables. Normally, operations could be processed just on one exactly specified dataset (row of the table), unlike SQL, where more than one row of a table can be manipulated with the WHERE condition.

5.1.2.2 When to use an Entity Bean

The use of entity beans is a question of design, experience, exigency and technology. Entity beans are suitable as coarse-grained business components. Due to the properties of entity beans (persistent, transactional, multi-user capable) they should be used as a distributed, transactional and persistent object.

In following situations the use of entity beans is recommended [cf. SiSt02]

- **Representation of persistent data** - If the state of a business object needs to be stored persistently then it should be designed as an entity bean.

- **Representing object with defined identity** - Instances of business objects have their own unique identity. This identity distinguishes one object instance from another, and allows locating a specific business object. Entity beans provide a persistent identity for business objects and should be used when the access by the identity is important to the application.
• **Providing access for multiple clients** - When multiple clients need to share the state and behaviour of a business object, an entity bean should be modelled. The bean provider can rely on the EJB container to ensure the right synchronization for the entity beans when they are accessed from multiple clients at the same time.

• **Providing stable persistent data management** – Entity beans are not depending on a particular user session and they survive container re-starts.

• **Persisting data in portable manner** - Using container-managed persistence guarantees that the bean’s view of its own persistent state is any time the same, independent how it is stored in the database system used in its deployment environment.

• **Simplifying transaction handling** – The container handles the transaction logic for persistent data. Therefore the developer has not to consider about coding transaction handling.

### 5.1.2.3 Life cycle of an Entity Bean

After creating the instance by the EJB container, the `setEntityContext` method, which passes the entity context to the bean, is called. Now the bean is in the pooled stage, where all instances are identical. The identity to an instance is assigned by the EJB container, when it is moved into the ready stage. There are two ways from the pooled to the ready stage. First, the client invokes the create method, and the EJB container calls the `ejbCreate` and `ejbPostCreate` methods. Second, the EJB container invokes the `ejbActivate` method. The bean’s business methods can be accessed now. There are also two paths to put the instance from the ready into the pooled stage. On the first path, the client invokes the remove method, and the container calls the `ejbRemove` method. On the second path, the EJB container invokes the `ejbPassivate` method.

When the EJB container removes the instance from the pool and invokes the `unsetEntityContext` method, the life cycle of the bean ends.
5.1.2.4 Difference between Entity Beans and Session Beans

Entity beans are different from session beans in several ways:

- Entity beans are persistent
- Entity beans allow shared access
- Entity beans have primary keys
- Entity beans may participate in relationship with other entity beans

**Persistence**

The state of an entity bean is saved in a storage mechanism (database) therefore the entity bean is persistent.

Persistence describes a capability used by the software-developer to store data structures in non-volatile storage such as a file system or a relational database [cf. WIKI06].

Without persistence data exist only in memory, and are lost when the program exits. Persistence allows a program to be restarted and reloaded with the data from a previous call of the application.
Persistence for the entity bean means that the state of the bean exists above the lifetime of the application in the J2EE server process [cf. JATU05].

Entity beans can manage persistence in two ways, bean-managed and container-managed\textsuperscript{15}.

**Shared access**

Entity beans can be shared by multiple clients. As the clients may want to manipulate the same data, entity beans have to work within transactions. These transactions are provided the EJB container, by specification of the transaction attributes in the bean’s deployment descriptor. The transaction boundaries have not to be coded in the bean, the container marks the boundaries.

**Primary Key**

Each entity bean has a primary key which enables the client to locate a particular entity bean. The primary key is unique which identifies the entity beans. For example, a “customer” entity bean may have the “Customer ID” as primary key. Entity beans can only be used, when the objects have a unique identifier field, or when such field can be added.

**Relationships**

One entity bean may be related to other entity beans. For example, in a library application, “Author” and “Book” are related in a 1 – n relationship.

Relationships are implemented in a different way for entity beans with bean-managed persistence and those, with container-managed persistence. With bean-managed persistence, the relationship is implemented in the code. But with container-managed persistence, the EJB container manages the relationships (container-managed relationships.)

\begin{flushright}
\textsuperscript{15} See chapter 5.1.2.5 - Container Managed Persistence – Bean Managed Persistence.
\end{flushright}
5.1.2.5 Container Managed Persistence – Bean Managed Persistence

Two methods of handling persistence are available for entity beans: Bean-managed persistence (BMP) and container-managed persistence (CMP). BMP is implemented by the developer, and CMP is implemented by the container.

Table 2 outlines the main differences the responsibilities and coding between the two types of persistence.

<table>
<thead>
<tr>
<th>BMP/CMP Differences</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Difference</strong></td>
</tr>
<tr>
<td>Class definition</td>
</tr>
<tr>
<td>Database access calls</td>
</tr>
<tr>
<td>Persistent state</td>
</tr>
<tr>
<td>Access methods for persistent and relationship fields</td>
</tr>
<tr>
<td>findByPrimaryKey method</td>
</tr>
<tr>
<td>Customized finder methods</td>
</tr>
<tr>
<td>Select methods</td>
</tr>
<tr>
<td>Return value of ejbCreate</td>
</tr>
</tbody>
</table>

Table 2 – Differences between container-managed and bean-managed persistence

**Bean-managed persistence**

With bean-managed persistence (BMP) the entity object manages its own persistence. The bean provider has to implement persistent operations (e.g. JDBC,
JDO, or SQL/J) directly in the methods of the enterprise bean class [cf. SEAR06].

In comparison to container-managed persistence bean-managed persistence is more complex because the persistent logic, like saving, loading and finding of data, has to be implemented explicitly in the bean class. In order to handle the persistence in the bean class, the developer has to know which database will be used and how the bean class’s fields map to that database [cf. Mons04].

The implementation of bean-managed persistence allows a higher flexibility in how the state is managed between the instance of the Entity Bean and the database.

Bean-managed persistence should be used in following situations: [cf. SiSt02]

- When exact control over the database schema needed.
- When code has to be customized to match a specific legacy database schema.
- When the application has to be very finely tuned against the database that is used.
- When portability is not an issue. But in this case, however, it is recommended that data access objects (DAOs) are used. DAOs better enable to evolve into an entity bean with container-managed persistence or enable the bean to be adapted to a different database schema at a later date.\(^\text{16}\)
- When the queries need by the application exceeds the current capabilities of EJB QL.\(^\text{17}\)

---

\(^\text{16}\) DAO – Data Access Objects - In situations where the use of container-managed persistence is not suitable, data access objects can be used to encapsulate access to persistent data. A data access object (DAO) design pattern separates the interfaces to a system resource from the underlying strategy used to access that resource. Both entity beans with bean-managed persistence and session beans can use DAOs.

\(^\text{17}\) While EJB QL will continue to develop, it is not yet able to express a number of queries that are expressible in SQL.
• When the persistent store is not a database system that is not likely to be supported for container-managed persistence.

• When the deployment tools are inadequate for mapping the beans instance’s to the database.

One disadvantage of BMP is that a higher effort is required to develop the bean, because all database logic has to be implemented manually. This requires accuracy in using the EJB callback methods such as `ejbLoad()` and `ejbStore()` adequately. Additionally the find methods have to be coded explicitly in the bean’s home interface. Another disadvantage of BMP is the link of the bean to a specific database type and structure. Any change in the database requires changes to the bean instance’s definition. An entity which is BMP is not as database-independent as a CMP entity, but it can better accommodate a complex or unusual set of data\(^\text{18}\) [cf. Mons04].

**Container-managed persistence and relationship**

With container-managed persistence, the access to the persistent state of the bean is managed by the container. The bean provider does not have to implement any database access calls in the entity bean. As the container-managed bean’s code contains no database access (SQL) calls, the same entity bean can be redeployed on different J2EE servers which use different databases. Therefore, container-managed entity beans are more portable. Another advantage of using CMP is that the task of coding entity beans will be simplified, because the container takes the responsibility of generating the database access code.

Since EJB 2.0 the bean provider defines public abstract accessor methods (get and set) for each container-managed persistent and container-relationship “field”. The container provides the implementation of the get and set methods used at the runtime. The “field” itself is invisible to the bean.

\(^{18}\) Containers that use object-to-relational mapping tools in bean-managed persistence can ease this disadvantage.
In order to generate the data access calls, the container needs information, provided in the abstract schema of the entity bean. The abstract schema\(^\text{19}\) is a part of the entity bean’s deployment descriptor and defines the persistent fields and relationship of the bean. The name of the abstract schema is referenced by queries written in EJB Query Language (EJB-QL). For an entity bean with CMP an EJB QL query for every finder method (except `findByPrimaryKey`) has to be defined.

The persistent fields of an entity bean illustrate the state of the bean. At runtime, the container synchronizes this state with the database. During deployment, the container maps the entity bean to a database table and maps the persistent fields to the table’s columns. An `Author` entity bean, for example, may have `firstName` and `lastName` as persistent fields and these fields are virtually. They are declared in the abstract schema, but are not coded as instance variables in the entity bean’s class. These persistent fields are identified in the code by getters and setters (access methods).

A relationship field identifies a related bean; it is like a foreign key in the database table. Like the persistent field, the relationship field is virtually and is also defined in the beans class with access methods. But a relationship field does not represent the state of a bean. Container-managed relationships may be unidirectional or bidirectional, and the may be of any cardinality (1-1, 1-n, n - m). Entity beans relationships are manipulated by using get and set accessor methods and the methods of the `java.util.Collection` API.

**Benefit of container-managed persistence**

- It provides a layer of data independence in two locations: between the entity bean and the persistent store and between the client view and the entity bean.

\(^{19}\) The term abstract distinguishes this schema from the physical schema of the underlying data store.
• Enhances an entity bean’s portability:
   The migration between different EJB containers and different persistent
   stores could be done without recompiling the bean

• The use of EJB QL provides a way to specify queries for the finder and
  select methods of entity beans which is independent of their implementa-
  tion in a particular database.

• Development of the entity bean can be accelerated and simplified, be-
  cause the developer does not have to be concerned with the details of
   persistence management.

5.1.3 Persistent objects in EJB 3.0

“The model for persistence and object/relational mapping has been considera-
ably revised and enhanced in the Enterprise JavaBeans 3.0 release.” [JAEB06].

Since EJB 3.0 persistent entities are not longer called entity beans, they are
now called EJB 3.0 entities or persistent entities, which are lightweight persis-
tence domain objects. From the developer’s view EJB 3.0 entities are just “Plain
Old Java Objects” (POJOs) with annotations which specify how the object
should be stored in the database. The mapping of the objects to relational data-
base tables is done automatically by the EJB 3.0 container, and the developer
needs not implementing specific database access APIs.

5.1.3.1 Requirements on the Entity class according to the Java
Persistence API

• The entity class needs the annotation @Entity or must be denoted in
  the XML descriptor.

• The entity class must have a no-arg constructor, which has to be public
  or protected.
• The entity class must be a top-level class. Interfaces or enumerations\textsuperscript{20} should not be defined as an entity.

• Entities support polymorphic associations, polymorphic queries, and inheritance.

• Instance variables represent the persistent state of an entity, and they may correspond to Java Beans properties. The state of the entity is available for clients only through the entity’s accessor methods or other business methods. All instance variables must be private, protected, or package visibly.

```java
@Entity
public class Customer implements Serializable {
...
}
```

### 5.1.4 Message-Driven Beans

A message-driven bean (MDB) is an asynchronous message consumer that implements some business logic running on the server [cf. JAEB04]. The container invokes the MDB as a result of the arrival of a message from a JMS Queue or Topic. The message-driven bean acts as a JMS\textsuperscript{21} message listener, which is similar to an event listener except that it receives messages instead of events. Messages can be set by an application client, a web component or any other Enterprise JavaBean, or by a JMS application that does not use the Java EE technology.

Message-driven beans are anonymous, have no client-visible identity and their instances have no conversational state. All bean instances are equivalent when they are not involved in serving a client message. The message-driven bean has no home or remote interface.

\textsuperscript{20} An enumeration is an interface of an object, which generates a series of elements, one at a time.

\textsuperscript{21} The Java Message Service (JMS) API is a messaging standard that allows application components based on the Java 2 Platform, Enterprise Edition (J2EE) to create, send, receive, and read messages. It enables distributed communication that is loosely coupled, reliable, and asynchronous [cf. JAVA06].
The goals of message-driven beans are to allow concurrent processing of a stream of messages and to handle the processing of incoming messages as simple as developing the same functionality in any other message listener.

5.1.4.1 Difference between Message-Driven Beans and Session Beans and Entity Beans

The most important difference between message-driven beans and the other bean types (session and entity beans) is, that clients have no access to message-driven beans through interfaces. In comparison to session beans and entity beans the message-driven bean has only a bean class.

In several aspects message-driven beans are similar to stateless session beans.

- A message-driven bean’s instance contains no data or conversational state for a specific client.
- All instances of the message-driven bean are equivalent.
- A single message-driven bean can handle messages from multiple clients.

5.1.4.2 When to use a Message-Driven Bean

JMS messages can be sent and received with session beans and entity beans synchronously, but not asynchronously. To avoid binding server resources by blocking synchronous receives in a server-side component, the usage of message-driven beans is recommended, when receiving messages asynchronously.
5.1.4.3 Life cycle of a Message-Driven Bean

For each instance of a message driven bean the EJB container performs following tasks:

1. In EJB 2.1 the container calls the `setMessageDrivenContext` method to pass the object to the instance, and in EJB 3.0 the container injects these references before instantiating the instance, if the message driven bean uses dependency injection.

2. The instance’s `ejbCreate` method is called in EJB 2.1, or in EJB 3.0 the container calls the method annotated `@PostConstruct`, if any.

A message driven bean has only two stages: “does not exist” or “ready”, and it is never passivated.

At the end of the life cycle, the container calls the `ejbRemove` method (EJB 2.1), or calls the method annotated `@PreDestroy` (EJB 3.0).

![Diagram of EJB 2.1 life cycle](image)

Figure 10 – EJB 2.1 - Life cycle of a message driven [cf. JATU05]

![Diagram of EJB 3.0 life cycle](image)

Figure 11 - EJB 3.0 - life cycle of a stateless session bean [cf. JATU06]
5.2 Defining Client access with Interfaces

The client accesses a session or an entity bean through the methods defined in the bean’s interfaces\(^{22}\). These interfaces define the client’s view of a bean, whereas all other aspects of the bean, e.g. method implementations, abstract schemas, or database access calls, are hidden from the client. The bean’s interface simplifies the development and maintenance of Java EE applications, because the bean could be changed internally without affecting the client. One of the first decisions when designing a Java EE application is choosing the type of client access allowed by the enterprise bean: remote, local, or Web service.

5.2.1 Remote clients

The remote client of an enterprise bean has following features:

- The client can run on a different physical and Java virtual machine (JVM) than the enterprise bean it accesses.

- The client can be a web component, an application client, or another enterprise bean.

- The location of the enterprise bean is transparent to a remote client.

---

\(^{22}\) As message driven beans have a different programming model, they do not have interfaces which define client access. As entity objects in EJB 3.0 are POJOs, this section is only valid for Session Beans in EJB 3.0.
Interfaces for a remote client in EJB 2.1

For an enterprise bean which has remote access, a remote and a home interface has to be implemented. The remote interface defines the business methods of the bean, and the home interface defines the life cycle methods, create() and remove(), of the bean. For example, the remote interface for a CustomerFacade Bean might have the business methods getCustomerInfo(). For entity beans, the home interface also defines finder and home methods to locate entity beans.

Interfaces for remote clients in EJB 3.0

An enterprise bean in EJB 3.0 which has remote access has to be annotated @Remote. The remote interface defines all business and life cycle methods for this bean. The home interface is not longer required in EJB 3.0.
5.2.2 Local clients

If the client has to run on the same JVM as the enterprise bean it accesses, it has to be defined as local. A local client can be a web component or another enterprise bean and the location of the bean it accesses is not transparent to the local client. In EJB 2.1 it is often an entity bean that has a container-managed relationship with another entity bean. The local and the local home interface have to be coded, where the local interface defines the bean’s business methods, and the local home interface defines its life cycle and finder methods. In EJB 3.0 the business interface of the enterprise bean has to be annotated @Local. The local interface defines the bean’s business and life cycle methods.

5.2.3 Web Service clients

A web service client has two ways for accessing a Java EE application. First, the client can access a web service which is created by JAX-RPC, or second, the web service client can invoke the business methods of a stateless session bean. With the correct protocols (SOAP, HTTP, WSDL), any web service client can access a stateless session bean, even if the client is not written in the Java programming language. This flexibility allows the integration of Java EE applications with web services.

In EJB 2.1 the Web service endpoint interface defines the business method of the bean. Only methods which are defined in the Web service endpoint interface may be invoked by a Web service client. In EJB 3.0 only the business methods annotated @WebMethod may be invoked by the Web service client.

---

23 “The World Wide Web is more and more used for application to application communication. The programmatic interfaces made available are referred to as Web services.” [W3WS02].

24 JAX-RPC is a Java API for executing Remote Procedure Calls on the base of XML.
5.3 Contents of Enterprise Beans

Following files have to be implemented by the application developer for an enterprise bean (see table 3 and figure 14).

<table>
<thead>
<tr>
<th>Components</th>
<th>In EJB 2.1</th>
<th>In EJB 3.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enterprise bean class</td>
<td>Implements the methods defined in the interfaces</td>
<td>Implements the methods defined in the business interface and any life cycle callback methods</td>
</tr>
<tr>
<td>Interfaces</td>
<td>Remote and home interfaces; for local access, the local and the local home interfaces are required; for web services, the web service endpoint is required</td>
<td>Business interface – defines the methods implemented by the enterprise bean class</td>
</tr>
<tr>
<td>Helper classes</td>
<td>Other classes needed by the enterprise bean, e.g. exception and utility classes</td>
<td></td>
</tr>
<tr>
<td>Deployment descriptor</td>
<td>XML file with information about the bean’s persistent type and transaction attributes</td>
<td>Not longer required</td>
</tr>
</tbody>
</table>

Table 3 – Contents of the enterprise beans

The required files are packed into the module that stores the enterprise bean (ejb.jar file). This file is portable and can be used for different applications. To create a Java EE application one or more modules are packed into an archive file (EAR file), that holds the application.
5.4 Naming conventions

As enterprise beans consist of multiple parts, the EJB specifications propose following naming convention for enterprise bean applications.

<table>
<thead>
<tr>
<th>Item</th>
<th>Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enterprise bean name (DD)</td>
<td>&lt;name&gt;Bean</td>
<td>AccountBean</td>
</tr>
<tr>
<td>EJB JAR display name (DD)</td>
<td>&lt;name&gt;JAR</td>
<td>AccountJAR</td>
</tr>
<tr>
<td>Enterprise bean class</td>
<td>&lt;name&gt;Bean</td>
<td>AccountBean</td>
</tr>
<tr>
<td>Home interface</td>
<td>&lt;name&gt;Home</td>
<td>AccountHome</td>
</tr>
<tr>
<td>Remote interface</td>
<td>&lt;name&gt;</td>
<td>Account</td>
</tr>
<tr>
<td>Local home interface</td>
<td>&lt;name&gt;LocalHome</td>
<td>AccountLocalHome</td>
</tr>
<tr>
<td>Local interface</td>
<td>&lt;name&gt;Local</td>
<td>AccountLocal</td>
</tr>
<tr>
<td>Abstract schema (DD\textsuperscript{25})</td>
<td>&lt;name&gt;</td>
<td>Account</td>
</tr>
</tbody>
</table>

Table 4 - Naming conventions in EJB 2.1 [cf. JATU05]

\textsuperscript{25} The item is an element of the deployment descriptor.
<table>
<thead>
<tr>
<th>Item</th>
<th>Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enterprise bean name</td>
<td>&lt;name&gt;Bean</td>
<td>AccountBean</td>
</tr>
<tr>
<td>Enterprise bean class</td>
<td>&lt;name&gt;Bean</td>
<td>AccountBean</td>
</tr>
<tr>
<td>Business interface</td>
<td>&lt;name&gt;</td>
<td>Account</td>
</tr>
</tbody>
</table>

Table 5 – Naming conventions in EJB 3.0 [cf. JATU06]

5.5 Enterprise JavaBeans Query Language

“The Enterprise JavaBeans query language, EJB QL, is used to define queries for entity beans with container-managed persistence. EJB QL enables the Bean Provider to specify the semantics of query methods in a portable way.” [JAEB04].

The EJB QL is a query specification language for the finder and select methods of entity beans with CMP. The EJB QL is able to be compiled to a target language of a database, like SQL. The Enterprise JavaBeans query language uses the abstract persistence schemas of entity beans for its data model. It defines operators and expressions based on this data model.

The Bean provider defines EJB QL queries in the deployment descriptor of the entity bean. He can navigate from an entity bean to other entity beans by using the names of Container-Managed-Relation (cmr) - fields in EJB QL queries.

With EJB QL abstract schema types of entity beans could be used in a query, when the abstract persistent schemas of the beans are defined in the same deployment descriptor as the query.

EJB QL queries can be used in two ways:

- Queries for selecting entity objects with finder methods that are declared in the home interface.
• Queries for selecting entity objects derived from an entity bean’s abstract schema type with select method defined in the entity bean class.

5.5.1 Simplified Syntax of EJB QL

The EJB QL uses a SQL-like syntax to select objects based on the abstract schema types and relationships of entity beans. The Bean Provider can navigate over relationships with the path expressions of EJB QL.

An EJB QL query is a string which consists of following conditions:

• a SELECT clause, which defines the type of objects to be selected.

• a FROM clause, which defines declarations that identify the domain to which the expression specified in the SELECT and WHERE clause of the query apply.

• an optional WHERE clause, which may be used to restrict the results returned by the query.

• an optional ORDER BY clause, which may order the results returned by the query.

In BNF\textsuperscript{26} syntax the EJB QL query is defined as:

\[
\text{EJB QL} :: = \text{select\_clause from\_clause [where\_clause] [orderby\_clause]}
\]

5.6 Java Persistence Query Language

The Java Persistence query language is the extension of the Enterprise JavaBean query language, which was defined in the EJB 2.1 specification. It enhances the EJB QL with further operations like bulk update and delete, JOIN operations, GROUP BY, HAVING, and sub queries, and supports the use of dynamic queries.

\textsuperscript{26} Backus-Naur Form (BNF): A notation that describes the syntax of high-level languages.
The Java Persistence query language enables the developer to specify the semantics of queries in a portable way, independent of the particular database system used in the environment.

“The query language uses the abstract persistence schemas of entities, including their relationships, for its data model, and it defines operators and expressions based on this data model.” [JAEB06]. The queries can be defined in the XML descriptor or in metadata annotations.

5.6.1 Simplified Syntax

A select query has following clauses: SELECT, FROM, WHERE, GROUP BY, HAVING, and ORDER BY. The SELECT and FROM clauses are required, the other clauses are optional.

The clauses SELECT, FROM, WHERE, AND ORDER BY are described in chapter 5.5.1. The GROUP BY clause groups the results of the query according to a set of properties, and the HAVING clause is used with the GROUP BY clauses to restrict the query results.

The UPDATE and DELETE statements provide bulk operations over a set of entities and have following syntax:

update_statement ::= update_clause [where_clause]

delete_statement ::= delete_clause [where_clause]

The UPDATE and DELETE clauses determine the type of entities to be updated or deleted, and the WHERE clause may restrict the scope of the update or delete operation.
6 Integrated Development Environments for EJBs

For the development of EJB applications, as for programming in general, it is highly recommended to work within an Integrated Software Development environment. IDEs offer the developer lots of features to increase their productivity and the quality of applications.

Professional vendors, like the IBM, and open source groups, like Eclipse, provide a wide range of different development tools for the development of Java EE (former J2EE) applications. Because of the different features included, and other influences, the selection of the “right IDE for the Job” is not easy and should be mainly depending on the tasks the development team has to handle.

To find out, which IDE is the right one for the developer, following points have to be considered [cf. Neat05]:

- **Usability** – focus on the user friendliness of the IDE. This includes the design of the GUI, for example, where the features, which are used frequently, are placed. And one of the important parts of the usability is how fast the developer gets familiar with the IDE.

- **Functionality** – focus on additional features, which may be included in the development environment, e.g. profiler-support, debugging, code-completion, reflection engines, etc.

- **Performance** – the speed of the IDE while working or compiling is also an important point. The performance may be depending on the hardware, but some elements of IDEs may slow the performance, too.

- **Plug-in support** – The support of plug-ins may be helpful, when additional components for the IDE should be developed, or third-party plug-ins should be included.
• **Costs of licence and support** – for professional software development teams, the costs of buying and running the IDE are one of the main selection criteria when choosing one IDE.

• **Support of standards** – When developing Java EE applications, it has to be ensured, that the IDE is based on all standards and specifications.

### 6.1 Use of Integrated Software Development Environments

In the information technology the life cycle of new products shortens more and more. And the customers and initiators from software development projects expect that high quality software has to be created in less time.

Under these circumstances the capability to create high quality software in short time is a critical factor of success. Developers can only stay competitive, if they have effective technologies and tools available.

Integrated Software Development Environments can provide support for the day-to-day work of developers, increase their productivity and the quality of their products.

**Concepts and features**

To create software of high quality in adequate time specialised tools are necessary. Basically an editor for the input of the source code and a compiler for translating the source code into machine code are necessary. But in the view of toady’s complex software-systems this is not enough. A simple text editor is not enough to cover requirements, design, implementation and test cases and the translation process is so complex, that it could not be operated without any auxiliary tools.

Due to the increase of tools for development of new technologies the effort for developers rises because of different concepts and views. The developer has to arrange consistence between data and tools. The usage of many single tools increases the orientation effort and has a negative impact on productivity.
To reduce these problems, IDEs support the developer during the software development process. The IDE offers specialised tools for the different tasks in the development process. It combines these tools under one integrative surface.

The main advantages of IDEs are

- Interaction between compiler and editor: defective lines of code are marked.

- The combination of debugger and editor allows setting breakpoints directly in the source code.

- Complex activities are automated or simplified by intelligent user guidance (“wizards”).

IDEs offer a wide range of functionality, which eases and fastens the software development process. These functionalities can be divided into core functionality and advanced functionality. The features editing, navigating, translating and debugging are the basics of each IDE.

Basic features

Editor:

Every source code editor should be able to highlight the syntax of the programming language. In addition, the editor should be configurable for different additional languages, because in software development process variable files (configuration files or scripts) have to be modified.

The code formatter should be active during the input, for example putting the cursor on the right position. Due to the automated formatting existing code can be adapted to the company standard, which increases the readability and eases the understanding.

Another feature, which provides a better overview of the source code, is the expanding/collapsing of code blocks.

Configurable short-cuts are part of the standard functionality of an editor. Every editing function can be assigned to a key-combination and automated comple-
tion of types, method names and correction of typing mistakes are also basic features of modern IDEs.

**Navigation**

Usually IDEs work project oriented and administrate all project assigned information and artefacts. The analysis of this bundled information offers an intelligent navigation between the single artefacts. This is one of the main advantages of an IDE compared to tools, which have only access to single files. “Cross-referencing”, searching for classes, methods or attributes are also standard features for IDEs. Particularly in wide range projects filter functions in different views can offer more clearness.

**Compiler**

In IDEs the compiling process runs mainly in the background. The compiling process can be automated with integrated dependency tools, and the compiling takes not more than a “click”. Some IDEs compiles permanently in the background and mistakes are shown in the editor immediately. The recognition of dependency allows incremental translation, only changed source code, and referenced parts will be compiled.

If more than one compiler is available, each compiler should be configurable on its own and should be applicable and convertible on demand. Within the IDE configuration options like compatibility properties, for example between Java 1.3 and Java 1.4, error and warning messages, and other parameters can be set.

**Build-Process**

The creation of an application from the source code and the additional artefacts can be a complex process and can have many steps and dependencies. IDEs usually offer an internal build-process which is easy to use by the developer. Due to the abstraction of the file system and the artefacts, which are explicit necessary for the build, the effort for the developer is reduced seriously.

---

27 Artefacts are structured elements of software and their aggregations, which have to be versioned.
Additional to the internal build-process IDEs offer the usage of alternative Build-Tools like Ant or make. The according scripts can be started directly from the IDE and may be edited there, too.

**Debugging**

The combination between editor and debugger for setting breakpoints directly in the source code is available in all IDEs. Breakpoints can be stated absolutely in the source code or conditional for loops. Other standard features of the debugger are the different types for running the code step-by-step (step in, step out, step over).

Special features for the debugger are the finding of memory / resource leaks, controlling of input and output in files or network interfaces.

**Advanced features**

**Configuration management**

All artefacts of a software project are administrated in a central repository, to provide all involved developers a consistent basis. The changes on the artefacts are recorded in revisions, to provide traceability and to step back to older version, if the code is not correct.

Several artefacts of certain revisions are combined to a version. During the development process new revisions are created, and are integrated in new versions. With configuration management it is possible to access any version or to view the differences between the versions. This is particular necessary for the support of older versions.

**Code generation**

With the use of “wizards” and interactive created elements of programs (classes, features, components) the effort for writing source code is reduced considerably. Mistakes can be avoided and coding standards can be followed better. The developer states class name, package and inheritance relations, the code and all required artefacts are generated by the IDE. The so created elements are assigned to the project and to the build-process automatically. Be-
side the interactive, dialogue-based creation of single elements, the possibility of creating complex class-structures based on patterns with wizards is offered by IDEs with code generation. Another example for code generation is the creation of graphical user interfaces with visual tools (GUI designer).

**Testing**

If unit-tests and programming should be carried out parallel during the development process (Extreme Programming\(^{28}\)) the IDE should support this. Part of this is the comfortable creation of tests and the simple execution during the development process, without leaving the current context. The execution of the tests should be possible out of the editor or a navigation view, and the test results should be shown in a separate window. To find the defect code a step in the code (similar to a debugger) is useful.

**Refactoring**

Another important task in the software development process and integrative part of the Extreme Programming is the so called refactoring. Refactoring restructures the source code, without changing the behaviour of the application. Parts of refactoring are the creation of methods out of code fragments or the extraction of interfaces out from existing classes. The later moving of classes within the inheritance hierarchy, the renaming of classes, methods and attributes or the changing of method-signatures\(^{29}\) are tasks of a typical refactoring tool.

**Modelling / Design (UML)**

Beside the classic IDE, which is more code- and editor-oriented, design-tools with focus on software modelling are getting more important. First this tools were simple editors for diagrams, which creates software models in a given no-

---

\(^{28}\) Extreme programming (XP) is an agile software development process for small teams. The process allows to develop long-lasting software and to react to vague and rapid changing requirements.

\(^{29}\) Method signatures are the sum of all parameter and return-values of a method.
tation during the design process before implementing the software or afterwards, for documentation or communication reasons.

The predominant notation for software models is the Unified Modelling Language (UML), which established a standard for object oriented modelling. In UML2 different types of diagrams exist, which can be divided into structure describing (static) and behaviour describing (dynamic). Parts of the structure describing diagrams are class-, component- and deployment-diagrams. Examples for dynamic diagrams are activity-, sequence- and collaboration-diagrams.

The IDEs with UML support can be classified into code-oriented and model-oriented user guidance. Code-oriented IDEs focus on the manipulating the source codes, the UML-diagrams are used only for visualising, for example used in Borland’s JBuilder. The model-oriented IDEs use the graphical model for navigation and as a central element for creation of new classes and other elements, like in Borland’s Together.

Design tools offer import-/export interfaces, to interchange diagrams with other tools. In case of UML this is handled by XMI. The diagrams could be saved as graphics and whole models could be exported to HTML, to be readable outside of the diagram-editor of the IDE. Additional features of a modelling tool are validity- and consistence-reviews, support for “design patterns30” and “design critics31”.

With the similarities in the meta-model the graphical notations and the programming languages could be consolidated together. For example, out of the UML class diagram the Java source code can be generated (“Forward Engineering”) or reverse, source code can be visualised with diagrams (“Reverse Engineering”). The motivation for Forward Engineering is the reducing of typing errors and a reduced error rate because of defined generation rules. Reverse Engineering offers a good abstraction and therefore a better understanding for

30 Design patterns are recurring solutions to software design problems.
31 Design Critics is a system for finding bad design and suggestion of alternatives.
the code. “Round-Trip Engineering” is the combination of these two generation steps, model and code are handled alternating.

The use of external design tools is often a serious disruption in the software development chain, because the consistence between design and implementation has to be handled manually. Therefore nearly all diagram editors support Forward and Reverse Engineering, where the editing of the generated code is carried out in an external editor or IDE. On the other side, code-oriented IDEs are extended with diagram editors, which offer the visualising of the source- or byte-code in UML-diagrams. An adequate Round-Trip Engineering, a two-way editing in code and diagram with automated synchronisation of all artefacts, is offered in few tools only.

Databases

If a database has to be created or used during the software development, the IDE should offer adequate features or connection possibilities. For the creation of database schemas the modelling of ER (“Entity Relationship”) – diagrams is useful, from which code and scripts for database access can be generated. The possibilities of queries and visual editing of datasets can be integrated in the IDE, too.

Deployment/Runtime environment/Server

As mentioned before some IDEs allow a direct installing (“deployment”) and running of the implementation or components of the application on external or in the IDE integrated web- or application servers. This offers faster and less complicated testing and debugging. Therefore the IDE has to provide an own runtime environment or has to have the possibility to access external runtime environment. Thereby, mostly with commercial IDEs, the own platform is supported best, which can influence the choice of the application server. “Hot deployment” allows the installation or the changing of component, without stopping the server. This is a big advantage during the development process, because the time of restart can be saved.
GUI-Designer

The creation of graphical user interfaces (GUI) is eased with a GUI-builder. This tool allows the visual combination of the single elements (menu, toolbar, boxes), which are converted into code and linked into the program.

Internalisation

Ideally the IDE supports the internalisation of the developed software. Therefore a mechanism is necessary, which stores the labels and messages in the source code, and a tool for translation has to be provided. Beside, the user interface of the software has to be designed flexible, so that the size of the graphical elements could be depended on the language (usually labels have different length in different languages).

Help-System

The basic functionalities of an IDE are usually understood intuitive, but to use all features the study of manuals is needed. A good, context-sensitive online-help reduces the need of reading printed manuals.
6.2 Selected IDEs for EJB development

This chapter offers a closer look on the features of selected development environments for EJB applications.

The selection of four IDEs is based on the type of vendor (commercial or non-commercial) and on the technology they are based on.

<table>
<thead>
<tr>
<th>Product</th>
<th>Type of Vendor</th>
<th>Technology</th>
</tr>
</thead>
<tbody>
<tr>
<td>NetBeans IDE 5.0</td>
<td>Non-commercial</td>
<td>NetBeans platform</td>
</tr>
<tr>
<td></td>
<td>NetBeans community</td>
<td></td>
</tr>
<tr>
<td>IBM Rational Application Developer for WebSphere Software 6.0</td>
<td>Commercial - IBM</td>
<td>Eclipse platform</td>
</tr>
<tr>
<td>Sun Studio Enterprise 8</td>
<td>Commercial - Sun</td>
<td>NetBeans platform</td>
</tr>
<tr>
<td>Eclipse with Web Tools Platform (WTP)</td>
<td>Non-commercial</td>
<td>Eclipse platform</td>
</tr>
<tr>
<td></td>
<td>Eclipse community</td>
<td></td>
</tr>
</tbody>
</table>

Table 6 – Vendor type and technology of the selected IDEs

Even if some IDEs are provided by commercial vendors, they may be freely available, for example Sun Studio Enterprise 8. Both technology platforms, NetBeans and Eclipse, are written in Java language.

6.2.1 NetBeans 5.0

NetBeans is a development environment which is completely implemented in Java. The NetBeans development environment is open source and was founded by Sun Microsystems in the year 2000. The NetBeans project has a growing community of developers and users and nearly 100 partners worldwide which are supporting this project.

NetBeans was distributed by Sun Microsystems under the name “Forte” and there was a free and a chargeable Version. In the meantime this project has been divided into the free NetBeans project, which is available on the projects
web page http://netbeans.org and into the chargeable Version “Sun Java Studio” of Sun Microsystems\(^{32}\).

NetBeans supports the Java developer with a lot of tools and has a wide range of features, which exceeds the features of other free available development environments. One mentionable difference to other Java IDEs is that NetBeans expects a basic knowledge of Java programming from the user. Whereas other products offer wizards for some programming tasks and cover up the background, NetBeans supports the developer in configuration and creation of Java EE applications, but he has always to know what he is doing [cf. WIKI06].

The version of NetBeans 5.0 which is presented in this chapter was released on 25\(^{th}\) January 2006 and is available on the web page http://netbeans.org. Figure 15 shows the main window of the NetBeans 5.0 development environment.

![Figure 15 – Graphical user interface of NetBeans 5.0](image)

\(^{32}\)The Sun Java Studio is free available for members of the Sun Developer Network on the web page http://java.sun.com.
Selected Features in NetBeans 5.0

Editing improvement – NetBeans provides enhanced code completion and generation of code snippets through the code completion box. It offers syntax highlighting for Java, XML, HTML, CSS, JSP and IDL and fully support the new JDK 1.5 features. The most important refactoring features are renaming of classes, methods, fields and other Java related objects (e.g. local variables and parameters) and the finding of usages of classes, methods, etc.

Version control – Different version control could be used within NetBeans. CVS is integrated in the workflow and the IDE lists all changed files in the versioning window. Updates and commits could be executed directly over the main IDE windows.

Enterprise development – NetBeans 5.0 support Java EE 1.3 and 1.4 standards and offers wizards for the creating of all types of Java Enterprise Beans, and also for creating Entity Beans from existing databases. The IDE is able to generate calls to EJBs from other classes or from Web modules and validates EJB modules and Java EE applications using the Java EE Verifier\(^{33}\). When using the Sun Java System Application Server EJBs could be deployed by one click. Automated management of the deployment descriptor for EJB modules and Java EE application is also included, as well as the import of existing EJB modules and Java EE applications with or without existing Ant scripts.

The Web development is eased by templates for the creation of JSF Managed Beans, Struts Actions, and Struts Form Beans and by the enhanced support for JavaServer Faces and Struts. The Web application build support is based on Apache Ant and one click deployment of web application is offered for several server vendors (Sun Java System Application Server, Tomcat Web Server, JBoss and WebLogic).

---

\(^{33}\) Java EE annotations and deployment descriptors are validated against their corresponding DTD schema files. The Java EE Verifier notices when a module or application is not compliant to the application server and/or to the Java EE specifications.
Supported Software for NetBeans 5.0

Operating system

- Microsoft Windows XP professional SP2
- Microsoft Windows XP professional SP3
- Solaris 9, 10 Operating systems
- Red Hat Enterprise Linux 3
- Sun Java Desktop System
- Mac OS X 10.3

Application Servers

- Sun Java System Application Server 8.2
- BEA WebLogic Server 9.0
- JBoss Application Server 4.0

Database Servers

- IBM DB2 8.1
- Oracle 9i
- PointBase 4.2
- Microsoft SQL 2000 SP3
- MySQL 4.x

Version Control Systems

- CVS 1.1
- PVCS 7.5
- Microsoft VSS 6.0
- ClearCase
- StarTeam
- Subversion
- TeamWare

---

34 The NetBeans IDE runs on operating systems that support the Java Virtual Machine. The list of the OS includes the OS which had been tested with NetBeans.
6.2.2 IBM Rational Application Developer for WebSphere Software

The Rational Application Developer (RAD) is the IDE-platform provided by IBM under the brand WebSphere\textsuperscript{35}. The Rational Application Developer is a powerful integrated development environment for visually designing, implementing, testing and deploying J2EE applications, Web services and portals. WebSphere Studio Application Developer is built on Eclipse, an open, industry-supported platform for development tools.

The following section presents some selected features of the IBM Rational Application Developer in version 6.0, released on November, 11\textsuperscript{th} 2004. Figure 16 provides a main view on the GUI of the RAD 6.0.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{RationalApplicationDeveloper60.png}
\caption{Graphical user interface of Rational Application Developer 6.0}
\end{figure}

\textsuperscript{35} WebSphere refers to a brand of IBM integration and application software products. WebSphere is designed to set up, operate and integrate e-business applications across multiple platforms using web technologies and includes run-time components (application server) and development tools.
Features of Rational Application Developer 6.0

**J2EE development environment** - J2EE 1.4 is supported, including EJB 2.1, servlet 2.4, JSP 2.0. The RAD 6.0 also could generate codes for the former J2EE 1.2 and J2EE 1.3 specifications.

The EJB 2.1 specification is fully supported, including EJB Query Language (EJB QL), multiple-mapping support for Container Managed Persistence and message-driven beans.

The EJB development is simplified by the EJB Project Wizard and the EJB Editor. The EJB Client projects contain all the classes necessary to invoke an enterprise bean and the EJB Snippet Wizard simplifies the implementation of EJB client access code.

The RAD includes the O/R Mapping\(^{36}\) in different directions, top-down, meet-in-the-middle and bottom-up.

Furthermore RAD provides tools for import/export creation, code generation, editing and support for standard deployment descriptors, extensions and bindings specific to the WebSphere Application Server.

The Rational Application Developer supports the building of J2EE applications with JDK 1.4.2, targeting WebSphere Application Server v5.1.

**UML Support** - The UML visual editor enables the visualizing and editing of J2EE code, and eases the management of complex code. The visualization is created dynamically, and it is always synchronized with the underlying code.

RAD provides to generate new code directly from the UML class diagram into the Java or EJB project. The class diagram editor can be used to create new beans and edit relationships between beans.

**Portal development environment** - The IBM Rational Application Developer includes tools for visual portlet\(^{37}\) development and a test environment for the portlets. The portlets can be created by using the Struts framework and the Web diagram editor to visualize the structure and event flows.

---

\(^{36}\) O/R Mapping – object-rational mapping is a programming technique which links databases to object-oriented language concepts.

\(^{37}\) Portlets are web components designed to be accumulated in the context of a composite page.
**Java development environment** - Additionally to the EJB tools, Rational Application Developer provides different tools and wizards to ease Java development. The EGL\(^{38}\) development environment enables the developer to develop Web applications and business logic without coding in Java. RAD includes visual editors for developing GUI components (Swing\(^{39}\) or AWT\(^{40}\)) and a new JavaDoc\(^{41}\) generation. These features are combined with dynamic debugging, error reporting and correction.

**Web services development environment** - The WSDL Editor allows graphical editing WSDL files and embedded XML schemas and the Web Services Explorer reduces the time for creating, validating and detecting Web services. The Web service support in WSAD is based on open standards such as UDDI V2, SOAP, WSDL and WSIL.

**XML development environment** - Editors for XML, DTD, XML schema, and XSL are included in the Rational Application Developer. The RDB to XML mapping editor eases the definition of the mapping between relational tables and a DTD file with a visual interface.

**Relational database tools** - Tools like wizards, views, editors and other features alleviate the development and testing of database elements of the application. The included SQL query wizard and SQL query builder provides a visual interface for creating and executing SQL statements.

**Web development environment** - The Web development environment in RAD includes a Web Site Designer, a Page Designer visual tool, and JavaServer Faces\(^{42}\) (JSF) support for drag-and-drop Web application development.

---

38 EGL (Enterprise Generation Language) is a 4th generation language developed in the 1980s by IBM. EGL should help procedural programmers to learn the concepts and practices of object-oriented programming more easily.

39 API for implementing of graphical user interfaces (GUI) and integrated in Java since JDK 1.2.

40 AWT (Abstract Window Toolkit) is a standard API for implementing and representing platform independent GUI and is a part of the Java Foundation Classes (JFC).

41 JavaDoc is a documentation tool which generates HTML documentation out of Java source code. For additional information special comments could be included in the source code.

42 Framework Standard for Web applications. JSF allows the developer to simply add user interface components into Web sites and eases the definition of the navigation.
Team development - The team development environment provides open support for pluggable source code control repositories. This open support allows integrating many different SCM (Software Configuration Management) solutions. The Rational Application Developer 6.0 includes following SCM-tool: Version Control Management (VCM) provided by Eclipse and includes an adaptor for IBM Rational ClearCase and for CVS (Concurrent Versions System).

Server tools for testing and deployment - Integrated test environments are provided by the RAD to increase productivity. EJB components, JSP files, servlets and HTML files can be tested on local or remote server. Unit tests could be run on WebSphere Application Server, WebSphere Application Server – Express, WebSphere Portal and Apache Tomcat.

Additional features in the Rational Application Developer are tools for monitoring and analyzing performance, debuggers for detecting and analyzing errors in the code, a log and trace analyser and a wide range language support. The vendor’s description of the IBM Rational Application Developer 6.0 is available on the Web page http://www.ibm.com/developerworks/rational/products/rad/.

6.2.3 Sun Studio Enterprise 8

This chapter introduces the development environment for Java EE applications from Sun Microsystems inc., Sun Java Studio Enterprise 8, which was released in November 2005.

The Sun Java Studio Enterprise (JSE) is based on the NetBeans platform, which is well known in the open-source community. In addition to the features of NetBeans, the JSE development environment provides some extra functionality like UML support, developer collaboration, and application profiling. The JSE is a fast development environment with fast built times. As the modules are loaded while starting the IDE, the JSE is ready for action in a few seconds and fit with reactions while working. An intuitive GUI and sample code elements of-

---

fer even new or inexperienced developers to get familiar with the Sun Java Studio. Plugins and additional modules for JSE are available, and there is, of course, the possibility to add and remove plugins, but basically all tools and features are included in the IDE from the start. As this tool is developed by Sun Microsystems it is fully based on Java Standards. For critical development tasks this might be an advantage. In general, the Sun Java Studio is capable for applications which should run on Sun based Web- and Java EE application server.

Figure 17 – Graphical User Interface of Sun Java Studio Enterprise 8

Selected features in Java Sun Studio Enterprise 8

Editing support – The source editor is customizable, and it is used for all Java source, HTML and plain text files. The most mentionable features are the dynamic code completion, which displays associated JavaDoc in-place, customisable abbreviations and the automatic inclusion of Try Catch blocks. Comprehensive refactoring capability enables the restructuring of an existing body of Java application code.
Integration for web or enterprise development – the development of Java Web or enterprise applications is eased by the integration of Tomcat Web Server, Apache Ant, unit testing with JUnit, and database development support with PointBase Server. The IDE comes with pre-configured and ready for use Java System Application Server, Java System Web Server, Java System Directory Server, and Java System Identity Server.

Enterprise application development support – JSE 8 supports the J2EE 1.4 standards and wizards enable the fast development of EJB modules and Java EE application assembly. The Container Managed Persistence wizard is able to generate a set of CMP beans based on a database schema and of course the EJB components can be validated against the Java EE specification rules. The ease of generating EJB test applications along with the Java EE application is also mentionable.

UML Support – Sun Java Studio Enterprise integrates model-driven analysis and design by UML 2.0. This architectural design feature increases visual clarity and reduces complexity, and allows forward and reverse model-to-code engineering.

Supported Software for Java Sun Studio Enterprise 8

Operating system

- Microsoft Windows XP SP2
- Microsoft Windows 2000 professional SP4
- Solaris 9, 10 Operating systems
- Sun Java Desktop System

Application Servers

- Sun Java System Application Server 8.1 PE UR2 (bundled)
- BEA WebLogic Server 9.0
- JBoss Application Server 4.0
- WebSphere 6.0

---

44 For J2EE applications the Java Sun Studio Enterprise 8 supports JDK 1.4 on the source level of J2SE 5.0.
Database Servers

IBM DB2 8.2
Oracle 9i, 10g
PointBase 4.8
Microsoft SQL 2000 SP3
and several JDBC Drivers for Sybase, Microsoft, etc

Version Control Systems

CVS 1.11
ClearCase V2002 u2
Visual SourceSafe 6.045
PVCS 7.5.10

6.2.4 Eclipse with Web Tools Platform 1.5.1

The Eclipse Web Tools Platform (WTP)\(^4^6\) project extends the Eclipse development environment with tools for developing J2EE applications. The WTP version and the underlying Eclipse, version 3.2.1, which are described here, were released in September 2006. The WTP project is divided into different subprojects with focus on Web application development (WST) or J2EE applications (JST).

To meet the requirements for developing J2EE applications the Eclipse extension “J2EE Standard Tools Project” (JST) was implemented.

The scope of JST is the support of J2EE developing within the Eclipse environment. This includes the support of the J2EE 1.4 specifications for JSP, Servlets, EJBs, JDBC, etc.

\(^{45}\) Visual SourceSafe and PVCS support is for Microsoft Windows only.
\(^{46}\) The installation package and all relevant information can be downloaded from [http://www.eclipse.org/webtools/index.html](http://www.eclipse.org/webtools/index.html).
JST supports the annotations of JSR 175 – Metadata\textsuperscript{47} for code assist. For J2EE annotations technologies like XDoclet\textsuperscript{48} are included in the Eclipse extension JST.

The architectural principles of the J2EE Standard Tools Project are:

- Extending the Eclipse user experience.
- Using of common J2EE application project directory layouts.
- Offering source code features like syntax colouring, quick fixes, refactoring, etc.

Operation area of Eclipse WTP – JST

- Development of J2EE Web applications, including JSP pages, tag libraries, servlets, testing and debugging Web applications.
- Development of EJB applications, supported by wizards for generating the EJBs, or adding relationships and mappings to an EJB. JST also supports the packaging of the \texttt{ejb.jar}, with generating the necessary classes and running, testing and debugging the EJBs.
- Development of J2EE Enterprise applications, with the possibility of adding and removing client applications, Web applications and EJBs.

\textsuperscript{47} Java Specification Request 175 – A Metadata Facility for the Java Programming Language.
\textsuperscript{48} XDoclet is an open source code generation engine, which enables attribute-oriented programming in Java.
- Development of Java Web Services, based on Session Beans or Java Beans\(^\text{49}\), and creating Web Service clients.

Figure 19 presents the main view of Eclipse with the WTP 1.5.1 plug-in for development EJB applications.

![Figure 19 – Graphical User Interface of Eclipse 3.2.1 with WTP 1.5.1](image)

\(^{49}\) Java Beans are reusable software components written in the Java programming language. Java Beans should be not confused with Enterprise JavaBeans, which is the server-side component technology in the Java EE.
7 Example Implementation - The Customer Book

The implementation of this application should point out the differences and similarities of NetBeans IDE 5.0 and IBM Rational Application Developer 6.0.

These two IDEs were chosen, because they are based on different platforms, and therefore some differences in handling the EJB development process are expected. The installation and basic configuration of the IDEs are not part of this evaluation, the focus lies on the process of developing a simple J2EE – EJB application.

The coding sample for evaluating the differences of two selected integrated development environments is a customer address and information application, where data from customer can be found and inserted into a database. This sample is based on the reference implementation of the NetBeans tutorial “Quick start guide for J2EE applications” [cf. NEBE06].

The use case for this simple application can be defined as followed:

Name: Find customer address
Goal: Find the customer address by entering the customer ID in the browser
Scope: Selection of customer address information (Name, e-mail)
Preconditions: Customer address data have to exist in the database table
Success Condition: Customer name and e-mail is displayed in the browser
Failure Condition: Customer name and/or e-mail address is not displayed in the browser
Trigger: User wants to retrieve customer address information

The entity-relationship diagram for the Customer Book application can be found in figure 20.
The Customer Bean, which implements the business logic, is an Entity Bean with a Session Bean facade and it is based on the EJB 2.1 specification.

Following parts are necessary for implementing the sample application:

- The project `customer book` is the enterprise application project, which is the container for the web module and the EJB module.
- The entity bean `Customer` accesses a sample database (CUSTOMER) which has customer related content (e.g. ID, Name, Street, etc).
- The session bean `CustomerFacade` handles the access between the web application client and the information in the entity beans.
- The `CustomerFacade` is a stateless session bean with only remote interfaces. It calls the entity bean by a `lookupCustomerBean` method.
- The Business method of `CustomerFacade` gets the customer information and returns name and e-mail address of the selected customer.
- The web module consists the custom service locator which will used by the IDE when generating calls to the enterprise bean.
- The servlet `CustomerDetail` looks up and displays the information for each customer in the CUSTOMER table.
Figure 21 shows the class diagram of the CustomerFacade session bean and the Customer entity bean.
7.1 The Customer Book application with NetBeans 5.0

This chapter describes the necessary steps for setting up the NetBeans 5.0 IDE for the “Customer Book” sample application and a step-by-step description for coding the sample application.

7.1.1 Prerequisites

1. The Sun Java Application Server Platform Edition 8.2 has to be registered in the NetBeans IDE.

2. The Derby database server, which is included in the distribution of the Sun Java Application Server, has to be configured and connected.

The Derby database is a Java based, open source, relational Database-Management-System and it is developed by the Apache Derby project. Apache Derby is distributed with a jar-file, which may be added to the classpath. Because of the small size (2 MB) and the simple installation process, Derby is a light-weight DMBS. Derby offers an embedded JDBC-mode and a network JDBC-mode for developing Java applications.

7.1.2 Creation of the Enterprise Application Project

The Enterprise Application template enables the developer to create the EJB- and the Web-module automatically.

1. File > New Project > Enterprise Application project

2. The name and the location of the project have to be specified.

---

50 An enterprise application project pools the resources that are required for deploying a J2EE enterprise application. It contains a set of Web modules, EJB modules, and other modules, which are combined to compose an EAR file.
7.1.3 The Container managed Entity Bean – Customer

1. The CMP Entity bean from Database has to be created in the Customer-BookProject-EJBModule.

Figure 22 – Create the Enterprise Application project

Figure 23 – Create the CMP Entity Bean
2. The JDBC connection for the generation of the Entity Beans has to be selected now – jdbc:derby://localhost:1527/sample and the generated Entity Beans should be stored in the ejb package.

![Figure 24 – setting the JDBC connection for the Entity Bean](image)

3. After connecting to the Derby database, the desired table (CUSTOMER) can be selected. The IDE creates the CMP Bean for the CUSTOMER table and for any tables that are referenced from CUSTOMER (DISCOUNT_CODE). The JDBC connection pool and data source are also created, which will be registered on the application server at deployment time.
7.1.4 The Session Bean – Customer Facade

The session bean of this application is the facade between the Web application client, the servlet, and the information in the entity bean.

1. Creation of the session bean in the CustomerBook-EJBModule by choosing the EJBMODULE > New > Session Bean. The session bean’s name is **CustomerFacade** and it is placed in the **ejb** package. The session bean is stateless and has only remote interfaces.
2. Calling the Customer Entity Bean by using the option Enterprise resources>Call Enterprise Bean in the source code editor of CustomerFacade. This generates a `lookupCustomerBean` method.
3. The variable declaration `private.ejb.CustomerLocalHome custHome` has to be defined in CustomerFacade and a reference to the home interface of the entity bean has to be obtained in the session bean's create method:
   
   ```java
   public void ejbCreate() {
       custHome = lookupCustomerBean();
   }
   ```

4. Now the business method has to be assigned to the Session Bean by right-click in the bean class' body and choosing EJB Methods > add business method. Following values have to be defined in the wizard:

<table>
<thead>
<tr>
<th>Method name</th>
<th>getCustomerInfo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return type</td>
<td>String</td>
</tr>
<tr>
<td>Parameter</td>
<td>int custId</td>
</tr>
<tr>
<td>Exception</td>
<td>javax.ejb.FinderException</td>
</tr>
</tbody>
</table>

   The method `getCustomerInfo` needs now business logic added:
   
   ```java
   public String getCustomerInfo(int custId) throws javax.ejb.FinderException {
       ejb.CustomerLocal customer =
           custHome.findByPrimaryKey(new Integer(custId));
       return "Name: " + customer.getName() + ", E-mail: " +
           +customer.getEmail();
   }
   ```
7.1.5 The Web Module

The service locator

The service locator is a J2EE pattern to abstract all JNDI usage and to hide complexity of initial context creation, EJB home object lookup, and EJB object recreation [cf. JAVA06]. This J2EE pattern provides a mechanism to abstract all dependencies and network denials into the Server Locator.

NetBeans enables the creation of the service locator by a wizard. In this example the service locator is simply called ServiceLocator and it is put into the web package.

The Servlet CustomerDetail

The servlet enables the user to look up and display information stored in the CUSTOMER table.

1. The servlet could be created by selecting the CustomerBook-WebModule and choosing New > Servlet. The name of the servlet is CustomerDetail and it has to be stored in the web package.
2. In the source of CustomerDetail the CustomerFacade session bean has to be called by choosing Enterprise Resources > Call Enterprise Bean. The service locator strategy has to be set to web.ServiceLocator. Now the IDE inserts the lookup method into the servlet.

3. At last the processRequest method in the servlet has to be modified, to include the logic of the request.

```java
protected void processRequest(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<html>");
    out.println("<head>");
    out.println("<title>Servlet customerDetail<title>");
    out.println("</head>");
    out.println("<body>");
    out.println("<form>");
    out.println("Customer number: <input type='text'
name='customer_nr' />");
    out.println("<input type=submit value=Select />");
    out.println("</form>";
    String customerNr = request.getParameter("customer_nr");
    if((customerNr != null) & & !(customerNr.equals(""))){
        try{
            ejb.CustomerFacadeRemote custFacade =
lookupCustomerFacadeBean();
            out.println("Customer's info for nr. " + customerNr + ": " +
custFacade.getCustomerInfo(
            Integer.parseInt(customerNr)));
        }catch(javax.ejb.FinderException ex){
            out.println("Customer with nr. " + customerNr +" not
found");
        }
    }
    out.println("</form>";
    out.println("<form>");
    out.println("<input type='text'
name='customer_nr' />");
    out.println("<input type=submit value=Select />");
    out.println("</form>";
    out.println("</form>"
    out.println("</body>";
    out.println("</html>"
    out.close();
}
```

7.1.6 Building and Deploying the application

NetBeans does not require any additional configuration of deployment descriptors. They are already configured and the new connection pool and data source for the enterprise application are prepared, too.

As a Web application displays its index.jsp by default, it is necessary to change the properties of the project to display the CustomerDetail servlet
instead. Therefore the Relative URL has to be set to /CustomerDetail in the properties panel.

To run the application from the IDE, the CustomerBook enterprise application project has to be selected, and then right-click > Run Project.

Now NetBeans does the following:

- Builds the enterprise application project and all of its subprojects.
- Starts the application server.
- Undeploys the enterprise application if it is already deployed to the server.
- Deploys the enterprise application project.
- Opens the Web module at the specified relative URL.

The result is displayed in an external Web browser. When a customer number, which is held in the database, is entered, the page displays the information of this customer.
Servlet customerDetail at /CustomerBook-WebModule

Customer's info for nr. 999: Name: Daniela Novak, E-mail: h94259930@quantic.com as at

Customer number: [Field] Select

Figure 31 – CustomerBook Servlet
7.2 The Customer Book application in RAD

The following section contains a description how to implement the sample application “Customer Book” in the IBM Rational Application Developer 6.0 development environment.

7.2.1 Prerequisite

The Derby database server has to be configured and connected.

![Figure 32 – Configuration of the Derby database](image-url)
7.2.2 Creation of the Enterprise Application Project

1. File > Enterprise Application project.

2. The name and the location of the project have to be specified.

3. Generation of the related EJB and Web modules:

Figure 33 – Creating the Enterprise Application project

Figure 34 – Add new modules to the Enterprise Application project
After the enterprise application project is defined following modules can be found in the Navigator view:

- Customer Book
- CustomerBookEJB
- CustomerBookWeb

### 7.2.3 The Container managed Entity Bean

1. To create an Entity bean (container managed) from a database, the database table has to be selected in the project explorer of the RAD.

![Create EJBs from DB tables](Image)
2. The wizard maps the EJB to the RDB table CUSTOMER in the EJB project CustomerBookEJB.

![Diagram of EJB to RDB mapping]

Figure 36 – Mapping EJBs to RDB

3. After finishing the EJB to RDB mapping, the Entity bean Customer and all according methods and CMP fields are generated by the IDE and the class diagram opens.
7.2.4 The Session Bean – Customer Facade

The session bean CustomerFacade acts as the facade between the client and the entity bean.

The EJB module CustomerBookEJB has to be selected and by choosing “Create an Enterprise Bean” from the toolbar, the wizard for Enterprise JavaBeans opens. The session bean’s name is CustomerFacade and it is placed in the sample package. In the detail view of the EJB wizard the type “stateless” and “remote client view” have to be selected.
Figure 38 – Creation of the Session Bean

Figure 39 – Configuration of the Session Bean
7.2.5 Testing EJBs with the EJB Universal Test Client

The IBM Rational Application Developer includes an Universal Test Client (UTC) for testing Enterprise Java Beans. As the implementation of the Web Application is described in detail in chapter 7.1.5 for NetBeans and as the implementation of the Web Application is similar in RAD, the Client for the Customer EJB is not implemented in RAD, but the functionality of the Enterprise Beans is tested in this internal test environment.

Before working with the UTC the server with the deployed EJB project has to be started. After the server is in the ready mode, the Test Client Welcome page appears within the IDE.

Before working with the UTC the server with the deployed EJB project has to be started. After the server is in the ready mode, the Test Client Welcome page appears within the IDE.

![Welcome page Universal Test Client](image)

Figure 40 – Welcome page Universal Test Client

To find all testable EJBs the link JNDI-Explorer in the top left corner of the page has to be clicked. Now the selected EJBs are ready to be tested if the methods implemented within, return the right values.

![EJBs for testing in the UTC](image)

Figure 41 – EJBs for testing in the UTC
7.3 Compendium

The previous chapter gives an impression, how to develop a simple EJB application in NetBeans 5.0 and in Rational Application Developer 6.0. Both selected IDEs offer enough support for starters in EJB development, but as they are based on different technologies (Sun Studio and Eclipse) the main difference lies in the structure of the project view. The storage of required files is handled in different structures, but both of the IDEs are very intuitive.

Developers, which start with the EJB technology, may receive the impression, that NetBeans is more user friendly, because the preconditions for preparing the project are easier to handle. But on the other hand, the IBM Rational Application Developer offers a wider range of features, which are not closely linked to the EJB development process.

One of the main features which is missing in NetBeans 5.0 is the possibility to create EJ Bs with UML support, which may reduce the time for coding.

Developers with experience in Eclipse may find that working in the IBM RAD is easier to learn, because the basic functionality of Eclipse is given, and enlarged with special features, which have not to be plugged in separately.

New developers, with less experience in any integrated development environment may find NetBeans more convenient, because it is not too overloaded with features, which they do not need for their development tasks.

According to the points for selecting the IDE described in chapter 6 following statements can be pointed out for NetBeans and IBM Rational Application Developer.

**Usability** – both development environments are very user friendly, and both of them offer the possibility of configuring the GUI for the developer’s needs. As described before, beginners in developing EJ Bs may get faster familiar with the NetBeans IDE. This may be a result because the NetBeans project is closely linked with the development of the Enterprise Java Beans technology.
**Functionality** – for implementing the sample application Customer Book, RAD as well as NetBeans offer the required functionality.

**Performance** – due to the fact, that the sample is very simple, the performance in both environments was good. NetBeans and IBM Rational Application Developer reacted on user actions very quick and if the work station meets the system requirements of the IDE\(^{51}\), the performance seems to be equal.

**Plug-in support** – is provided in both selected IDEs.

**Costs of licence and support** – NetBeans IDE is free for commercial and non-commercial use. Support is available free in several developer networks and with costs in the Sun Developer Expert Assistance program. The IBM Rational Application developer is available for a charge of 2.186 EUR in the minimum licence\(^{52}\).

**Support of standards** - both, NetBeans 5.0 and IBM Rational Application Developer supports the J2EE 1.4 standard with the EJB 2.1 specification.

---

\(^{51}\) NetBeans 5.0: processor: 500 MHz Intel Pentium III or equivalent, RAM: 384 MB, disc space: 125 MB (Microsoft Windows OS).
IBM Rational Application Developer 6.0: 800 MHz Intel Pentium III or higher, RAM: 768 MB (1 GB recommended), Disc space: 3.5 GB (full feature).

\(^{52}\) Authorized User Initial Fix Term Licence with 12 month maintenance.
8 EJB IDEs for EJB 3.0

As both specifications of the EJB technology, EJB 2.1 and EJB 3.0, are used in software development at the time this Master Thesis was written, additional to the description of integrated development environments for the EJB 2.1 technology in chapter 6.2, a selection of integrated development environments for EJB 3.0 is provided in this chapter. It also includes a sample of a message-driven bean application developed in NetBeans 5.5, based on the EJB 3.0 technology.

The EJB 3.0 specification had the goal to simplify the development of enterprise business applications written in the Java programming language. A new persistence management was introduced, which replaced the complex entity beans with persistence objects, written in “Plain Old Java”, and the use of the Java language metadata annotations reduces the number of classes and interfaces that were required to be written manually in the previous EJB specifications. The use of the annotations also eliminates the requirement for the deployment descriptor. A detailed description of the features of EJB 3.0 and the difference to the EJB 2.1 specification can be found in chapter 5.

As a result of the improvements of the EJB 3.0 technology the different vendors of integrated development environments reacted on these changes by enlarging their products, with the goal to support the new EJB 3.0 specification. Because some of the vendors were highly integrated in the specification of Java EE, their experience with the new technology found deposition in the products they are offering.

8.1 NetBeans 5.5

The NetBeans 5.5, which was released in August 2006, is build on NetBeans 5.0 and adds support for Java EE 5 and the Sun Java Application Server PE 9. Additionally to the features listed in chapter 6.2.1 further support for the development of EJB 3.0 application was added.
It simplified the creation of session beans and message-driven beans, and “enhanced the Java source editor with code completion, on the fly error checking and verification, and hints specifically targeting EJB 3.0.” [NEBE06]. NetBeans supports the Java Persistence concept by generating Java entity classes from existing DB schemas, creates entity classes in the IDE and let the server generate the DB relying on the DB-from-Java feature in Glassfish\textsuperscript{53}, and the persistence runtime can be used in Web applications and plain Java applications without the need of a Java EE application server.

One further improvement of NetBeans 5.5 which is mentionable is the possibility for running SQL scripts directly from the IDE.

### 8.1.1 Development of a Message Driven Bean in NetBeans 5.5

The example for developing an EJB 3.0 application with a Message Driven Bean included is based on the NetBeans IDE 5.5 tutorial EJB 3.0 Enterprise Beans [cf. NEBE06].

The application called NewsApp uses a MDB for receiving and processing messages which are sent by a servlet. The messages are also displayed by a servlet.

The use cases of the NewsApp application can be defined as followed:

- **Name:** Post message
- **Goal:** Send a message to the database
- **Preconditions:** None
- **Success Condition:** Message is stored in the database
- **Failure Condition:** Message is not stored in the database
- **Trigger:** User enters a message

\textsuperscript{53} Glassfish is a project for the development of an open source Java EE 5 application server.
Name: List messages
Goal: Display messages which are stored in a database table in the browser
Preconditions: Messages have to exist in the database table
Success Condition: Messages are displayed in the browser
Failure Condition: Messages are not displayed in the browser
Trigger: User requests the listing of messages

Creating the Enterprise Application

1. In the New Project wizard the Enterprise Application from the Enterprise category has to be selected.

2. The name of the Enterprise Application is NewsApp and, for this example, the applications server has to be set to Sun Java System Application Server\(^\text{54}\).

3. In contrast to the sample application in NetBeans IDE 5.0, the J2EE version has to be set to Java EE 5 (see Figure 42 – Building the project NewsApp). The modules Web and EJB have to be created, too.

\(^{54}\) For implementing this example application a local instance of the Sun Java System Application Server has to be registered with the NetBeans IDE.
Creating a persistent unit

The persistent unit provides the container information for managing the entity objects in this application. It defines where and how the classes should be persisted.

The definition of the persistent unit in this example consists of:

- **Name of the persistent unit**: NewsApp-ejbPU

- **Persistence provider**: TopLink\(^{55}\), which enables the Sun Java System Application Server to automatically detect the database based on the connection.

- **Data source**: jdbc/sample

---

\(^{55}\) TopLink is an object-relational package, which provides a flexible framework for storing Java objects in relational databases [cf. WIKI06].
Coding the Entity Class NewsEntity

In EJB 3.0 the entity class is a simple “Plain Old Java Object” (POJO). With creating the NewsEntity class in the IDE, the required annotation @Entity is added automatically.

Each entity class must have a primary key, and the NetBeans IDE adds the @Id annotation to declare which field to use as the primary key. The annotation @Generated specifies the key generation strategy for the primary Id.

The NewsEntity class has following definitions:

- **Class name:** NewsEntity
- **Package:** ejb
- **Primary key type:** Long
After finishing the creation of the entity class, the source editor opens the `NewsEntity.java` file.

The field declarations `String title` and `String body` have to be added to the class manually.

With the wizard Refactor > Encapsulate Fields the setters and getters are generated. In the encapsulate fields dialog box the checkboxes for getter and setter have to be selected for all fields.
With clicking on “Do Refactoring” in the output window, the getter and setter methods are added in the `NewsEntity` class, and the visibility of the fields is changed to `private`.
Creating the Message-Driven Bean – NewMessage

The NewMessage class is created with the wizard New Message-Driven Bean, and it is packed into the ejb package. The Destination Type Queue adds the annotation `@MessageDriven(mappedName = “jms/Queue”)` to the class. This annotation informs the container that the component is a MDB and that the JMS resource is used by the bean.56
The annotations in EJB 3.0 allow the introduction of resource directly into the class. For the NewsApp example following annotations have to be added in the list of field declarations:

1. **MessageDrivenContext**
   ```java
   @Resource
   private MessageDrivenContext mdc;
   ```

2. **PersistenceContext**
   With right-clicking into the source and selecting Persistence > Use Entity Manager following annotation is added:
   ```java
   @PersistenceContext
   private EntityManager em;
   ```
   and following method is generated into the code:
   ```java
   public void changeMyName(Object object) {
     // TODO:
     // em.persist (Object);
   }
   ```

3. The method changeMyName has to be modified as followed:
   ```java
   public void save(Object object) {
     // em.persist (Object);
   }
   ```

**Modification of the onMessage method:**
```java
ObjectMessage msg = null;
try {
  if (message instanceof ObjectMessage) {
    msg = (ObjectMessage) message;
    NewsEntity e = (NewsEntity) msg.getObject();
    save(e);
  }
} catch (JMSException e) {
  e.printStackTrace();
  mdc.setRollbackOnly();
} catch (Throwable te) {
  te.printStackTrace();
}
```
4. To import the jms libraries the statements may be generating by Alt-Shift-F.

Creating the session bean – NewsEntityFacade

EJB 3.0 reduces the amount of required code for implementing session beans. With the NetBeans IDE wizard for session facades the PersistenceContext resource is injected directly when creating the session bean and the bean is annotated @Stateless.

1. The session facade for the NewsEntity class is created by selecting the persistence category (New > File/Folder), and choosing the Session Beans for Entity classes wizard.

2. From the list of available entity classes the NewsEntity class has to be selected.

3. The session facade bean has to be stored in the ejb package and the local interface has to be selected.

The Web module for NewsApp

The Web module for the sample application contains two servlets:

- ListNews – for displaying messages
- PostMessage – for adding messages

Creating the Servlet ListNews

1. After the class definition (name and package) the enterprise Bean is called by right-click in the source and selecting Enterprise Resources > Call Enterprise Bean.

2. By confirming the selection NewsEntityFacade, the entity bean resource is injected in the servlet using the @EJB annotation.

descriptors for the specification of the JMS resources are not longer required.
3. To fill the servlet with displayable context, the body of the method processRequest has to be filled with the bold printed content.

```java
out.println("<h1>Servlet ListNews at " + request.getContextPath() + "</h1>");

List news = newsEntityFacade.findAll();
for (Iterator it = news.iterator(); it.hasNext();)
    { 
    NewsEntity elem = (NewsEntity) it.next();
    out.println(" <b>"+elem.getTitle() +" </b><br/>");
    out.println(elem.getBody()+"<br/>");
    }
out.println("<a href='PostMessage'>Add new message</a>");
out.println("</body>");
```

4. As the classes from the util package have to be imported, the import statements have to be generated with Alt-Shift-F.

**Creating the Servlet - PostMessage**

1. In the Web module the Servlet which enables to post messages is created and putted into the web package.

2. In the field declaration part of source code the resources ConnectionFactory and Queue are injected with the annotation @Resource.

```java
@Resource(mappedName="jms/NewsConnectionFactory")
private ConnectionFactory ConnectionFactory;
@Resource(mappedName="jms/Queue")
private Queue queue;
```

3. The processRequest method has to be modified to send the JMS messages (lines in bold).

```java
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        String title=request.getParameter("title");
        String body=request.getParameter("body");
        if ((title!=null) && (body!=null)) {
            try {
                Connection connection =
                    connectionFactory.createConnection();

                Session session =
                    connection.createSession
                    (false, Session.AUTO_ACKNOWLEDGE);
```
MessageProducer messageProducer = 
    session.createProducer(queue);

ObjectMessage message = 
    session.createObjectMessage();

NewsEntity e = new NewsEntity();
    e.setTitle(title);
    e.setBody(body);

message.setObject(e);
messageProducer.send(message);
messageProducer.close();
connection.close();
response.sendRedirect("ListNews");

} catch (JMSException ex) {
    ex.printStackTrace();
}

PrintWriter out = response.getWriter();

To add the Web form for posting messages to the servlet the bold printed lines have to be added to the processRequest method.

out.println("<body>");
out.println("<h1>Servlet PostMessage at " + 
    request.getContextPath () + "</h1>");

out.println("<form>");
out.println("Title: <input type='text' name='title'><br/>");
out.println("Message: <textarea name='body'></textarea><br/>");
out.println("<input type='submit'><br/>");
out.println("</form>");

out.println("</body>");

4. Before finishing the coding of the PostMessage class the java.jms libraries have to be imported.

**Running the NewsApp application**

Before the NewsApp sample is able to run from the project window of NetBeans it is necessary to specify the relative URL. Therefore the relative URL /ListNews has to be stated in the properties of the Enterprise Application.

With a right-click on the NewsApp enterprise application node and choosing Run Project, the application starts and the servlet ListNews opens in the browser.
The link “Add new Message” opens the PostMessage servlet and the message can be posted now. After that the message is sent to the message driven bean for writing to persistent storage.

After sending the message the ListNews servlet opens again and displays the message entered before. As the message service is asynchronous, the messages may not be displayed immediately after sending.
8.2 JBoss Eclipse IDE

The JBoss IDE is a set of plug-ins for the Eclipse platform which enables developers to create, deploy, test, and debug JBoss Enterprise Middleware System – applications within the established development environment of Eclipse.

As the JBoss group was heavily involved in the specification of EJB 3.0 they were the first ones, which offered an open-source development environment for the implementation of EJB 3.0. The JBoss Eclipse IDE version 1.5.1, which is presented here, was released in March 2006. Figure 51 shows the main window of the JBoss Eclipse IDE.

![Image of JBoss Eclipse IDE](image)

Figure 51 – User interface of JBoss IDE for Eclipse

The development of EJB 3.0 applications is eased by project-wizards (see screenshot below), and wizards for EJB 3.0 Session Beans and Message-driven Beans. The EJB 3.0 classpath container is based on the JBoss application server installation.
Additionally the IDE offers an editor for XML-mapping files from Hibernate\(^{57}\) and the development of database schema in the reverse engineering\(^{58}\) procedures has been improved.

The highlights of JBoss IDE 1.5 are the open-source license, full integration in the Eclipse environment, which speeds the introduction time for developers, which are familiar with Eclipse. The J2EE development is accelerated and eased by a lot of wizards, not only for EJB 3.0, and when using the JBoss application server, the debugging, monitoring and administration of the server can be handled due to the IDE.

---

\(^{57}\) Hibernate is an open source persistence framework for Java. This technology is needed to convert objects into relational tables. It has an important role in the standardisation of the EJB 3.0 persistence standard.

\(^{58}\) Reverse Engineering – The process of creating an ER-Schema out of a structure of a table.
8.3 **Oracle JDeveloper 10g Release 3**

The Oracle JDeveloper is an integrated development environment with support for modelling, developing, debugging, optimizing, and deploying Java applications and Web services. It is free available for developers since June 2006. The goal of JDeveloper is to simplify the development of Java applications with focus on visual and declarative approach to J2EE development and with including the Oracle Application Development Framework (Oracle ADF) into the IDE [cf. Shme06].

The Oracle ADF is based on the Model-View-Controller (MVC) design pattern and is based on four layers:

- The Business Service layer which provides access to data from various sources and handles the business logic of an application.

- The Model layer is an abstraction of the Business Service layer, enabling the View and Controller layers to work with different implementations of Business Services in a consistent way.

- The Controller layer offers a mechanism to control the Web application flow.

- The View layer is the user interface of the application.

---

59 Oracle ADF is a J2EE development framework that implements design patterns and eliminates infrastructure coding.
The J2EE application development is simplified by including UML modellers for creating EJBs, simple Java classes, and Oracle ADF Business Components.

The JDeveloper provides support for EJB 3.0 and EJB 2.1, with wizards for creating the necessary classes and interfaces. Entity beans can be created from database tables, and with a simple dialog session facades beans can be generated for entity beans. JDeveloper supports the new annotation-based syntax for developing EJB 3.0.
9 Conclusion

The main goals of this Master Thesis were to give an insight to the Enterprise Java Beans technology with its main features and its application area, and to provide information about Integrated Development Environments for implementing this technology.

The first part is concerned with the theoretical basis of the EJB technology, its background and the parts of which EJBs consist. It offers a description of the different EJB types, how to define client access, and a view on the Java Persistent Query Language.

The second part describes the requirements on Integrated Development Environments, and provides a selection of IDEs which are appropriate for the development of Enterprise Java Beans applications. It also includes the implementation of a sample application in NetBeans 5.0 and in IBM Rational Application Developer 6.0.

The third section of this Master Thesis presents some Integrated Development Environments, which support the specification for EJB 3.0 by the end of the year 2006. A reference implementation of EJB 3.0 with Message Driven Beans is included there.

The Enterprise Java Beans technology meets the requirements of the server-side part of business applications. Due to its main advantages in simplification, reusability, portability, scalability, and transactional capabilities, it is broadly accepted in the developer’s community. As the EJB architecture is based on Java it is not difficult to understand for experienced Java application developers and with the redesign in the EJB 3.0 specification, the development process for Enterprise Java applications was eased. The developments in simplifying the enterprise part in Java made development faster and the goals of business applications can now be reached easier than in the previous versions of J2EE. One competitive advantage of EJB 3.0 is the handling of persistence. Entity objects need not longer to be implemented as container managed or bean managed entity beans, it is now possible to create entity object in “Plain Old Java”. An-
other mentionable improvement is the implementation of annotations in Java EE. The Java annotations have two key advantages: they replace the XML-base configuration files (e.g. deployment descriptors) and eliminate the need for the rigid component models [cf. Yuan05].

As EJBs, and the Java EE technology in general, are widely accepted in Enterprise application development, it is obvious, that there is a wide range of tools provided by different types of vendors, which support the development process for these applications. The tools are based on different technologies, like Eclipse or NetBeans, with the aim to offer “the right IDE for the Job” [Rajk05] to professional development teams. But not only commercial vendors, like Oracle or IBM, distribute Java development environment, there are a lot of developers communities, which offer their tools for free, and sometimes as open source, for example Eclipse.

The selection of the appropriate development environment is based on different requirements of the developer and his team. First of all, the chosen tool has to provide the best working area for the implementation goal; it has to support its users as best as possible, and of course, the price for licence and support is an important selection criteria, too.

Enterprise JavaBeans are not the only concepts for software components for the server-side part of business applications, there are some more technologies like Microsoft’s .NET framework, CORBA, or Spring, which all offer different solutions for the requirements on enterprise business applications. It should be always kept in mind, that each technology has its advantages and disadvantages, depending on the application area they are implemented in and the skills available for implementing the enterprise business application. When choosing the appropriate technology for a business solution not only the technical aspects should be analysed, the costs of maintaining and possible cost of chang-

---

60 The Microsoft .NET framework is a software component which provides pre-coded solutions, and manages the execution of programs written specially for the framework.
61 CORBA defines a framework for the implementation and configuration of business components.
62 Spring is a lightweight alternative to EJB based on Java. It offers a simplified and unified API layer over many Java SE APIs, Java EE APIs and open-source-frameworks. Even if Spring is seen as an alterna-
ing the existing architecture of the business applications have to be considered, as well.

The Enterprise JavaBeans technology may offer an advantage against their competing technologies, for example .NET, which is de facto limited to the Windows platform\(^{63}\), because it is based on the platform independent and broadly accepted Java programming language, and the implementation of EJBs was considerably eased by the release of the EJB 3.0 specification in May 2006.

Anyone who may get in touch with the EJB technology should analyse his requirements, and then try to find the appropriate development environment for implementing the business solution.

\(^{63}\) The design of .NET supports platform independence, but the full framework is implemented on the Windows operating system, only. Parts of the framework are implemented on non-Windows systems, but they are not widely used [cf. Sess01].
10 References

[Cowo06] Computerwoche.de, Article Oracle JDeveloper
http://www.computerwoche.de/nachrichten/557723/


http://java.sun.com/products/ejb/docs.html
Requested in August 2006.

Core part of the EJB 3.0 specification
http://java.sun.com/products/ejb/docs.html
Requested in August 2006.

http://java.sun.com/javaee/
Requested in August 2006.

http://java.sun.com/j2ee/1.4/docs/tutorial/doc/
Requested in August 2006.

Requested in August 2006.

http://java.sun.com
Requested in August 2006.
Requested in November 2006.


http://www.zdnet.de/builder/print_this.htm?pid=39131583-20000201c

[NEBE06] NetBeans IDE 5.0 Quick Start Guide for J2EE applications
http://www.netbeans.org/kb/50/quickstart-j2ee.html
Requested in August 2006.

[NEB506] EJB 3.0 Enterprise Beans
http://www.netbeans.org/kb/55.ejb30.html
Requested in August 2006.

[Rajk05] Rajkowski, M.: Choosing the Right IDE for the Job, January 2005
http://www.osnews.com


[SEAR06] Web Services and Service-Oriented Architecture
Article about bean-managed persistence
http://www.service-architecture.com/application-servers/articles/bean-managed_persistence_bmp.html
Requested in August 2006.
References


http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/titlepage.html,
Requested in August 2006.

[W3WS02] W3C Web Service Activity
http://www.w3c.org/2002/ws
Requested in August 2006.

[WIKI06] Web-based free content encyclopaedia project
http://en.wikipedia.org
Requested in August 2006.


http://javaworld.com