Instructional Design for
Java Enterprise Component Technology
Development of Component-Based Enterprise Applications

Open Universiteit Nederland, School of Computer Science
Master Thesis Computer Science

Marco Marcellis
837243998
26 October 2009
Instructional Design for Java Enterprise Component Technology
Development of Component-Based Enterprise Applications

Open Universiteit Nederland, School of Computer Science
Master Thesis Computer Science

Marco Marcellis
837243998
26 October 2009

Chairman
Prof. Dr. A. (Lex) Bijlsma
Open Universiteit Nederland
Dean of the faculty of informatics
045-5762870
lex.bijlsma@ou.nl

Supervisor and secretary
Dr. E. (Ella) Roubtsova
Open Universiteit Nederland
Assistant professor of the faculty of informatics
040-2450197 / 045-5762513
ella.roubtsova@ou.nl

Advisor
Dr. A.W.M. (Bert) Hoogveld
Centre for Learning Sciences and Technologies
Education technologist
045-576 2565
bert.hoogveld@ou.nl
Last year, I worked one day a week as a computer science teacher at the school “Inholland” of Alkmaar (the Netherlands). In this role, I was responsible for the course “Component Based Development”. Preparing this course, I noticed that there was a lack of sound tasks for instruction on this subject at the school “Inholland”. Therefore, I conducted a small investigation and discovered that there are no suitable textbooks for students. In addition, the task on the Internet did not satisfy me. I contacted Dr. Roubtsova, the examiner of the course “Component Based Development” of the Open Universiteit Nederland (I attended her course in March 2007) and asked her permission to use some of her learning material; this is how we became acquainted. We agreed that I should perform my master’s thesis in instructional design for component technologies.

I have studied at the Open Universiteit Nederland for more than ten years now. It was very hard to combine this with a fulltime job. I could not have finished this thesis all by myself. Many people have helped me in the process. I would like to thank my colleague Marko Draisma, my brother-in-law James Naylor and my friend Maurice Kool who helped me to improve my English. I would like to thank Jan Hoeve and Ad Ommen both master students at the Open Universiteit Nederland, for performing some of the instructions and checking the first part of this thesis. I would also like to thank Rene de Jong who is a freelancer Java teacher and consultant. He also checked the first part of this thesis.

I would like to thank the members of the graduation commission: Bert Hoogveld, Ella Roubtsova and Lex Bijlsma. Bert Hoogveld for helping me understanding the “Ten Steps to complex learning” method and Lex Bijlsma for listening to my presentation at the NIOC seminar in Utrecht. Special thanks for Ella Roubtsova, the supervisor and secretary of his thesis. She has read many drafts of my thesis and has given me a lot of feedback. I have sent her a million emails and called her many times. She was always there for me.

Last but certainly not least, I would like to thank my wife Inge Smid. I have sat for hours and hours behind my computer in our loft. In addition, the thesis joined us on every vacation we took. Thank you Inge for your support and patience with me.
Summary

Modern businesses do not exist without distributed enterprise applications. Development of such applications is supported by technologies for development of component-based applications. Because of their wide use, these component technologies belong to curricula of many master programs in computer science. One of the most popular component technologies is the Java enterprise component technology. Studying separate elements of this technology is not sufficient to apply this technology for the development of component-based enterprise applications, because of the requirements for the development of these applications do not correspond to the components. The requirements demand integration of components depending on the semantics of the requirements.

In this work, tasks for instruction are developed, which allow students to study the elements of the Java enterprise component technology and cover the process of development of component-based enterprise applications. The development of these tasks for instructions is based on the “Ten Steps to complex learning” method developed by Jeroen van Merriënboer and Paul Kirschner. This method prescribes that in each task the students have to perform the complete process of the development of a component-based enterprise application. The first task, which the students perform, consists of a case study in which the students get complete support. In the next tasks, the level of support is decreased; so that in the last learning task, the students have to build an enterprise application on their own. The CRUD (create, retrieve, update delete) and component specific requirements are used as the basis for the tasks, because they are required in any enterprise application. A set of learning tasks is bundled into task classes, which are separated, based on the requirements of the user access. The first task class focuses on Web browser access, the second task class on the Application client and Web browser access, and in the third task class, the Web service client is added. The contribution of this thesis is the design and the implementation of the first two task classes.

A web environment is chosen as the learning environment, which allows for the tasks for instruction to be accessed from any computer accessing the Internet. Another benefit is that this environment can easily be integrated into existing learning environments. The tasks can be used to as an extension of the course “Component Based Development” at the Open Universiteit Nederland. Another possibility is to use these tasks as an open course at the Open Universiteit Nederland.
Samenvatting

Hedendaagse bedrijven hebben veelal te maken met gedistribueerde applicaties. Het ontwikkelen van deze applicaties wordt ondersteund door technologieën voor het ontwikkelen van component gebaseerde applicaties. Vanwege het feit dat deze technologieën wijdverspreid zijn behoren deze component technologieën tot het curriculum van veel master informatica programma's. Een van de meest populaire component technologieën is de Java enterprise component technologie. Het bestuderen van de afzonderlijke elementen van deze technologie is niet voldoende om dit toe te kunnen passen op het ontwikkelen van component gebaseerde applicaties, omdat de requirements voor het ontwikkelen van deze applicaties niet corresponderen met de componenten. De requirements vereisen integratie van de componenten afhankelijk van de inhoud van de requirements.

In deze afstudeeropdracht worden leertaken ontwikkeld die het mogelijk maken om studenten de elementen van de Java enterprise component technologie te laten bestuderen en het complete proces van het ontwikkelen van component gebaseerde applicaties behels. Het ontwikkelen van deze taken is gebaseerd op de "Ten Steps to complex learning" methode van Jeroen van Merriënboer and Paul Kirschner. Deze methode schrijft voor dat in iedere leertaak de studenten het complete proces van het ontwikkelen van een component gebaseerde enterprise applicatie moeten doorlopen. De eerste leertaak die de studenten maken bestaat uit een case study die de studenten volledige ondersteuning biedt. De mate van ondersteuning neemt af in elk volgende leertaak; in de laatste leertaak moeten de studenten zelfstandig een applicatie ontwikkelen. De CRUD requirements (create, retrieve, update delete) en component specifieke requirements worden gebruikt als basis voor de taken omdat deze in elke enterprise application nodig zijn. Een set van leertaken wordt gebundeld in taakklassen die worden onderscheiden op basis van gebruikerstoegang. De eerste taakklas concentreert zich enkel op de toegang met behulp van een Web browser, de tweede taakklaas op de Application client en de Web browser en in de derde taakklaas wordt de Web service client toegevoegd. De bijdrage van deze opdracht is het ontwerp en de implementatie van de eerste twee taak klassen.

Een web omgeving wordt gekozen als de leeromgeving. Dit maakt het mogelijk dat de leertaken vanaf elke werkplek benaderd kunnen worden via het Internet. Een ander voordeel is dat deze omgeving eenvoudig geïntegreerd kan worden in bestaande leeromgevingen. De taken kunnen gebruikt worden als aanvulling voor de cursus "Component Based Development" van de Open Universiteit Nederland. Een andere mogelijkheid is dat de leertaken gebruikt worden als een open cursus aan de Open Universiteit Nederland.
# Table of contents

Foreword ............................................................................................................................... 5
Summary ................................................................................................................................. 7
Samenvatting .......................................................................................................................... 9
List of figures .......................................................................................................................... 15
List of tables .......................................................................................................................... 17

1 Introduction ....................................................................................................................... 19
   1.1 Problem statement ........................................................................................................ 20
   1.2 Research goal and question .......................................................................................... 20
   1.3 Research approach ....................................................................................................... 21
   1.4 Research limits ............................................................................................................. 22
   1.5 The Open Universiteit Nederland ................................................................................ 22
   1.6 Thesis outline .............................................................................................................. 23

Part I: Prerequisites ............................................................................................................... 25

2 Java Enterprise Edition platform ...................................................................................... 27
   2.1 Enterprise application ................................................................................................. 27
      2.1.1 Multi-tier architecture ........................................................................................... 28
      2.1.2 Application server ............................................................................................... 29
   2.2 Java EE ....................................................................................................................... 30
      2.2.1 Java enterprise application ................................................................................... 30
      2.2.2 Java EE components ............................................................................................. 32
      2.2.3 Middleware services ............................................................................................. 34
      2.2.4 Java application server .......................................................................................... 36

3 Java enterprise component technology ........................................................................... 37
   3.1 EJB component .......................................................................................................... 37
   3.2 EJB component model ............................................................................................... 39
      3.2.1 Interfaces .............................................................................................................. 40
      3.2.2 Naming ............................................................................................................... 41
      3.2.3 Meta data ............................................................................................................ 41
      3.2.4 Interoperability ................................................................................................... 41
      3.2.5 Customization .................................................................................................... 42
      3.2.6 Composition ....................................................................................................... 42
      3.2.7 Evolution support ............................................................................................... 43
      3.2.8 Packaging and deployment .................................................................................. 43
   3.3 EJB container ............................................................................................................. 43
      3.3.1 Execution of a stateless session bean ..................................................................... 44
   3.4 Difficulties of applying the EJB technology ............................................................... 46
Ten steps to complex learning ................................................................. 49
  4.1 Three persistent problems ............................................................... 49
  4.2 The ten steps to complex learning ................................................... 49
  4.3 Step 1: Design learning tasks .......................................................... 50
  4.4 Step 2: Sequence tasks classes ......................................................... 51
  4.5 Step 3: Set performance objectives .................................................. 52
  4.6 Step 4, 5 and 6: Design supportive information ................................. 52
  4.7 Step 7, 8 and 9: Design procedural information ................................ 52
  4.8 Step 10: Design part-tasks practice ................................................ 52
Part II: Development of tasks .................................................................. 53
  5 Analysis and design ............................................................................. 55
    5.1 Step 1: Design learning tasks ......................................................... 55
      5.1.1 Real life task ............................................................................. 55
      5.1.2 Whole task practice ................................................................. 55
      5.1.3 Task support ............................................................................ 56
      5.1.4 Variability ................................................................................ 57
    5.2 Step 2: Sequence tasks classes ....................................................... 57
    5.3 Step 3: Set performance objectives ................................................ 59
    5.4 Step 4, 5 and 6: Design supportive information ............................... 61
    5.5 Step 7, 8 and 9: Design procedural information ............................... 62
    5.6 Step 10: Design part-tasks practice ................................................. 62
    5.7 The training blueprint ..................................................................... 62
  6 Development of task part one .............................................................. 67
    6.1 Development of the task classes ..................................................... 67
    6.2 Task one: World enterprise application .......................................... 68
      6.2.1 Requirements ........................................................................... 69
      6.2.2 Goal state ................................................................................ 70
      6.2.3 Solutions .................................................................................. 71
    6.3 Task two: Pet Store .......................................................................... 71
      6.3.1 Requirements ........................................................................... 72
      6.3.2 Goal state ................................................................................ 73
      6.3.3 Solutions .................................................................................. 74
    6.4 Task three: Web shop ....................................................................... 74
      6.4.1 Requirements ........................................................................... 75
      6.4.2 Goal state ................................................................................ 76
      6.4.3 Solutions .................................................................................. 77
    6.5 Task four: feedback ......................................................................... 78
  7 Development of tasks part two ............................................................ 79
    7.1 Task one: Library enterprise application ......................................... 79
Appendix E  Case study Library enterprise application

E.1  Database Layer

E.1.1  Building the database

E.2  NetBeans

E.2.1  Define project

E.3  Persistence Layer

E.3.1  Database Connection

E.3.2  Persistence Unit

E.3  Creating the Entity Classes

E.4  Business layer

E.4.1  Creating the Session Bean Façade

E.5  Client layer

E.5.1  Application client

E.5.2  Design JFrame

E.5.3  Source JFrame

E.6  Run the application
List of figures

Figure 1: Research approach ................................................................. 21
Figure 2: The outline of the thesis .......................................................... 23
Figure 3: Three principal layers ......................................................... 28
Figure 4: Java enterprise application layering ...................................... 31
Figure 5: Java enterprise application architecture .................................. 31
Figure 6: Java enterprise application parts .......................................... 32
Figure 7: Containers ........................................................................ 33
Figure 8: Distributed computing ......................................................... 33
Figure 9: Middleware services ............................................................ 34
Figure 10: Enterprise JavaBeans ......................................................... 38
Figure 11: Component contract ......................................................... 39
Figure 12: Composition .................................................................. 42
Figure 13: Package and deployment .................................................... 43
Figure 14: Deployment descriptor ...................................................... 43
Figure 15: EJB 2.1. Architecture ....................................................... 44
Figure 16: Stateless session bean lifecycle .......................................... 45
Figure 17: EJB 3.0 architecture .......................................................... 45
Figure 18: A schematic overview of the ten steps ............................... 50
Figure 19: Support and problem solving process ............................... 51
Figure 20: Level of support ............................................................... 51
Figure 21: Three task classes ............................................................ 58
Figure 22: Skill hierarchy ................................................................. 59
Figure 23: Variability ..................................................................... 62
Figure 24: Design of learning tasks and task classes ......................... 63
Figure 25: Start page web environment ............................................. 67
Figure 26: Task one World enterprise application ............................. 68
Figure 27: ERD diagram World ......................................................... 69
Figure 28: Input interface ................................................................. 70
Figure 29: Language interface ........................................................... 70
Figure 30: City interface ................................................................ 70
Figure 31: Task two Pet Store .......................................................... 71
Figure 32: ERD Diagram Pet Store ................................................... 72
Figure 33: Seller contact info ............................................................. 73
Figure 34: Edit screen of a seller ........................................................ 73
Figure 35: Task three Web shop ........................................................ 74
Figure 36: ERD diagram Web shop .................................................. 76
Figure 37: Register page.................................................................76
Figure 38: Shopping .................................................................77
Figure 39: Shopping cart ...........................................................77
Figure 40: Task four Feedback ...................................................78
Figure 41: Task one Library enterprise application .......................79
Figure 42: ERD diagram Library ..................................................80
Figure 43: Task two BaMaS .......................................................82
Figure 44: ERD diagram BaMaS ..................................................83
Figure 45: Task three Feedback ..................................................85
Figure 46: Use-case diagram .......................................................95
Figure 47: Model View Controller .............................................96
Figure 48: Session bean facade ..................................................96
Figure 49: Component diagram ..................................................97
Figure 50: Sequence diagram .....................................................98
Figure 51: Deployment diagram ..................................................98
Figure 52: Define the database connection ..................................102
Figure 53: Create data source ....................................................103
Figure 54: Persistence context .....................................................107
Figure 55: Entity Life Cycle .......................................................108
Figure 56: Communication between Servlet and JSP .................110
Figure 57: Define the database connection ..................................123
Figure 58: Design JFrame .........................................................126
List of tables

Table 1: Basic elements of a component model ................................................................. 39
Table 2: Completion strategy ......................................................................................... 56
Table 3: Communication protocols of the task classes .................................................. 58
Table 4: Classifying performance objectives ................................................................. 60
Table 5: Objectives of the Enterprise JavaBeans ......................................................... 61
Table 6: Blueprint task class one .................................................................................... 63
Table 7: Blueprint task class two ................................................................................... 64
Table 8: Primary and foreign Keys of the BaMaS tables ............................................... 84
Table 9: Source code entity ........................................................................................... 104
Table 10: Source code session bean ............................................................................. 106
Table 11: Entity Manager API ....................................................................................... 107
Table 12: JSP constructs .............................................................................................. 111
Table 13: Variables names ............................................................................................ 126
1 Introduction

Nowadays, almost any business uses some kind of enterprise application. Enterprise applications support businesses such as commerce, manufacturing, healthcare etc. The goal of building enterprise applications is to solve business problems or to realize business opportunities. The examples of enterprise applications are online shopping systems such as Amazon (www.amazon.com), interactive product catalogues, automated billing systems, shipping tracking etc. These enterprise applications are distributed systems, which mean that the applications are situated at different remotely accessed servers. Development of enterprise applications demands a lot of knowledge to make the right choices. As the size and complexity of software systems grows, the identification and proper management of interconnections among the pieces becomes a central concern. Component-oriented programming provides manageable solution to deal with the complexity of software, the constant change of systems, and the problems of reuse. It enables enterprise applications to be constructed from prefabricated software components. Component-oriented programming is now the de facto paradigm for developing enterprise applications (Wang & Qian, 2005; Stojanović, 2005). Therefore, it belongs to curricula of many master programs in computer science. For example, the University of Manchester, the Southern Polytechnic State University and the Open Universiteit Nederland have included component-oriented programming courses in their curricula.

In this work, we want to develop tasks for instructions for teaching component-oriented programming. Currently, the two most used component technologies for building component-based enterprise applications are the Java enterprise component technology\(^1\) of Sun Microsystems and Microsoft .NET (Councill & Heineman, 2001). This research assignment focuses on the Java enterprise component technology, which is the choice of the Open Universiteit Nederland (OUNL).

Next sections present the research assignment. Section 1.1 examines the problem statement and section 1.2 the research goal and research question. Section 1.3 and 1.4 discuss respectively the research approach and research limits. Section 1.5 introduces the Open Universiteit Nederland and finally section 1.6 illustrates the thesis outline.

\(^1\) Also known as the Enterprise JavaBeans (EJB) technology.
1.1 Problem statement

Development of component-based enterprise applications is a complex task. Each enterprise application is unique because of the different requirements of each customer. In the process of implementing the enterprise application, many choices have to be made. Many aspects such as patterns, clients, frameworks and architectures are involved when creating the distributed parts of the application. Tutorials in the field of the Java enterprise component technology are mostly provided by vendors of an Integrated Development Environment (IDE). In these tutorials, the IDE plays a key role. The wizards of the IDE are demonstrated showing how to automate parts of the implementation of a component-based enterprise application. In addition, the tutorials focus on the separate components. However, the choice of necessary components and the glue code is to be done by the developer who should take into account the requirements to the complete enterprise application. The requirements for development do not correspond to the components; they demand integration of components of different layers depending on the semantics of the requirements. Knowing component specifications and wizards of IDEs does not help to reach the objectives of mapping requirements onto integrated components; students are not able to handle new problems in real life.

The problem statement of this thesis is defined as follows:

*Studying separate elements of the Java enterprise component technology is not sufficient to apply this technology for development of component-based enterprise applications because of variety of requirements to enterprise applications and difficulties of mapping them onto components.*

1.2 Research goal and question

The goal of this research assignment given by the Open Universiteit Nederland is to develop tasks for instruction, which allow students to study the elements of the Java enterprise component technology and cover the process of development of component-based enterprise applications. The target group are master level students studying computer science.

The research question is formulated as follows:

*Can the tasks for instructions be built in such a way that they cover the process of development of component-based enterprise applications when studying the elements of the Java enterprise component technology?*

The choice of how the tasks for instructions should be given is important to provide the most important information and demonstration within a restricted time. That is why the tasks for instructions for the development of component-based enterprise application should be carefully designed. From the teaching perspective, the development of a component-based enterprise
application is considered as a complex task where knowledge of components, architecture, an IDE and the ability to map requirements onto components should come together and has to be taught as a whole. We need a new approach to develop the tasks for instruction.

According to Merrill (Merrill, 2002) and Hoogveld (Hoogveld, 2003) the 4C/ID-model is the most comprehensive recent model of instructional design. The “Ten Steps to Complex Learning” (or just “Ten Steps”) method by Jeroen van Merriënboer and Paul Kirschner is a practical version of the Four Component Instructional Design model (for short, 4C/ID-model2), which is one of the successful approaches for development of effective learning material. These methods are best suited for dealing with complex tasks. We choose to apply the “Ten Steps to complex learning” method.

1.3 Research approach

The “Ten Steps” method is usually applied within the context of the ADDIE model (van Merriënboer & Kirschner, 2007). This ADDIE model provides a systematic approach to the analysis, design, development, implementation and evaluation of tasks for instruction (Penn State University, 2000). The “Ten Steps” method covers only the analysis and design phase. We choose to adopt the ADDIE model to cover all the phases. Figure 1 illustrates the research approach. All the different steps are interrelated.

---

2 The 4C/ID model is developed originally by van Merriënboer and others in the early 1990s (Merriënboer, Clark, & de Croock, 2002).
Analysis is the process of defining what has to be learned and design is the process of specifying how it is to be learned. Development is the process of authoring and producing the materials and implementation is the process of installing the tasks in the real world context. Finally, evaluation is the process of determining the adequacy of the tasks (Penn State University, 2000).

1.4 Research limits

In this research assignment, we focus on the first three phases of the ADDIE Model. The implementation and evaluation will not make part of the assignment.

1.5 The Open Universiteit Nederland

The Open Universiteit Nederland develops, provides and promotes innovative distance education. As the prime university for lifelong learning, it addresses the wide-ranging learning needs of people during their course of life, plus the need to achieve a considerable increase of the knowledge level of the community at large (Open Universiteit Nederland, 2008). The Open Universiteit Nederland consists of several schools. My employer is the school of Computer Science. This school provides both a Bachelor programme and a Master programme in computer science.

Research and education focus on five main areas (Open Universiteit Nederland, 2008):

- Information Systems and Business Processes
- Software Technology
- Distributed Systems and Communication Technology
- Mathematics and Artificial Intelligence
- Human and Social Aspects of Computing

The Open Universiteit Nederland consists, besides the schools, of three expertise centres (Open Universiteit Nederland, 2008):

- Centre for Learning Sciences and Technologies (CELSTEC)
- Netherlands Laboratory for Lifelong Learning (NeLLL)
- Ruud de Moor Centrum

Dr. Hoogveld, member of the graduation commission, works at the CELSTEC. CELSTEC aims to research, develop and provide sustainable and evidence-based solutions for the advancement of learning at work, at school, at home and on the move. This is accomplished by combining state-of-the-art research in the Learning Sciences with the innovative powers of new media, mobile devices and the Internet. New solutions are extensively tested in CELSTEC’s laboratories and in real practice, following the principles of open innovation. With the portfolio of CELSTEC’s activities
CELSTEC is able to help their partners to innovate and improve their education, training and professional development system in all essential aspects (About CELSTEC). CELSTEC provides the methodology to build pedagogical material to the schools of the Open Universiteit Nederland.

1.6 Thesis outline

Figure 2 shows the outline of this thesis.

Part I: Prerequisites
- Chapter 2: Java Enterprise Edition platform
- Chapter 3: Java enterprise component technology
- Chapter 4: Ten Steps to complex learning

Part II: Development of tasks
- Chapter 5: Analysis and design
- Chapter 6: Development of tasks part one
- Chapter 7: Development of tasks part two

Part III: Conclusions
- Chapter 8: Conclusions and recommendations

Figure 2: The outline of the thesis

Part I of this thesis presents the prerequisites. First, chapter 2 introduces the Java Enterprise Edition platform of Sun Microsystems (Java EE) of which the Java enterprise component technology is part of. Next, chapter 3 examines the Java enterprise component technology. Chapter 4 introduces the “Ten Steps to complex learning” method.

Part II of this thesis explains the development of the tasks; chapter 5 examines the analysis and design phase, and chapter 6 and 7 discuss respectively the development of the tasks.

Finally, part III presents the conclusions and recommendations.
Part I: Prerequisites
2 Java Enterprise Edition platform

The Java Enterprise Edition platform (Java EE) support developers to build enterprise applications. First, section 2.1 examines enterprise applications. Next, section 2.2 introduces Java EE.

2.1 Enterprise application

Martin Fowler describes some characteristics typical for enterprise applications, namely (Fowler, 2006):

1. Persistent data
2. A lot of data
3. Many people who access data concurrently
4. A lot of user interface screens
5. Integration with other systems

1. Almost all businesses operate with data stored into a database. These data need to be persistent, because they are usually used by multiple programs and needed for many years. The algorithms implemented in enterprise applications support the transformation, presentation and analysis of the data.

2.3. Enterprise applications are normally used by many people; therefore, many data is involved. Those people often access the same data at the same time, which leads to many challenges. The enterprise application handles these issues. The triggers of the data transformations may also be different sensors, devices and even other enterprise applications.

4. Different people in different (business) roles often demand different user interfaces. Hence, many user interfaces have to be available within an enterprise application. These user interfaces are mostly accessible through a web browser.

5. Another specific element of an enterprise application is that it often has to integrate with other systems such as legacy or back-office systems.

In addition, enterprise applications are distributed in space. This means that databases usually run on their own server, the business services and algorithms are situated at other servers and the users usually use their own computers with Internet or local connections. Sub-section 2.1.1 discusses the multi-tier architecture (structure), which deals with the complexity of the distributed enterprise applications.
2.1.1 Multi-tier architecture

Layering is one of the most common techniques that developers use to break apart a complicated system. Figure 3 illustrates the three principal layers of an enterprise application (Fowler, Patterns of Enterprise Application Architecture, 2006).

![Figure 3: Three principal layers](image)

The presentation layer is responsible for displaying the graphical user interface (GUI) and handling user input. The domain layer hosts the business logic and the database source layer is typically a database that is primarily responsible for storing persistent data. These “logical” tiers are usually mapped onto separate physical tiers (Councill & Heineman, 2001). These tiers normally consist of a client machine, a middleware server and a database server. This architecture is called multi-tier architecture. The benefits of the multi-tier architecture are (Councill & Heineman, 2001):

- **Flexibility**: relocating the business logic onto a middleware server means that the same business services can be accessed from many different types of clients. These clients can potentially use one of multiple, interchangeable middleware servers that offer the same services. These servers can be upgraded, or more servers can be added, as the application scales.

- **Scalability**: all enterprise applications contend for certain scarce resources, such as database connections. By concentrating these resources in the application server, caching and recycling strategies can ensure that many clients share these resources optimally.

- **Robustness**: by delivering business services from interchangeable servers as described above, a server failure is no longer catastrophic. Clients of the failed server can be redirected to another server providing the same services.

The multi-tier architecture has evolved from the client-server architecture (Councill & Heineman, 2001). The client-server architecture typically consists of a fat client hosting the business logic and a server hosting the database. Fat client means that many operations are performed on the client. The client-server architecture with a fat client has two main disadvantages. First, the fat client was found to be inappropriate for many kinds of Internet-based solutions. Second, the growing variety
in client technologies raised another concern with respect to installation and upgrade of the client software. A more cost-effective means for supporting many clients was required. This became a more important issue with the growing popularity of the Internet, and the availability of non-traditional client devices such as network computers, Internet appliances, and handheld devices (Brown W., 2000).

The multi-tier architecture solves these issues. The middleware server plays a key role in this architecture. This server is usually called application server. Sub-section 2.1.2 discusses the application server.

2.1.2 Application server

Building an enterprise application leads to many challenges for the developer such as:

- **Transaction management.** What if two (or more) users of an enterprise application want to change the same piece of data in the database? Alternatively, what if a user for example put some items in an online shopping basket and the network crashed? If the customer has not finished shopping, he or she does not want that the chosen articles are actually bought. A transaction is a mechanism that deals with these issues.

- **Security.** There are many possible violations, vulnerabilities and risks, threatening the enterprise application resources. There is a need to reduce the overall risk using authentication, authorization, data integrity protection and data confidentiality protection (Sriganesh, Brose, & Silverman, 2006). Authentication checks if the user really is who he/she represents himself to be. Authorization checks the privileges of the user to resources. Data integrity protection prevents modification of data and data confidentially and data protection prevents unauthorized disclosure of information.

- **Messaging support.** A messaging system is a peer-to-peer facility: a messaging client can send messages to, and receive messages from, any other client. Messaging is a technology for building asynchronous communication, which helps building loosely coupled systems. Loosely coupled systems are systems, which are more flexible and easier to maintain.

- **Persistence management.** The enterprise application has to be able to store the data in a relational database. In addition, the enterprise application needs to perform CRUD operations on the data. CRUD means: create, retrieve, update and delete.

In order to build enterprise applications, transaction, security, message and persistence services are needed. These services are called middleware services, because they are required at the middleware tier. Building these services ourselves is time-consuming and very complicated. Application servers provide these services, which enables the developer of enterprise applications...
to focus on developing the enterprise application instead of developing the middleware services. Using middleware services to handle the complexity of the development of an enterprise application is called dividing and conquering (Sriganesh, Brose, & Silverman, 2006). Section 2.2 discusses Java EE that applies the multi-tier architecture and uses the application server to develop enterprise applications.

2.2 Java EE

Java EE was known as Java 2 Platform, Enterprise Edition (J2EE) until the name was changed to Java EE 5 or just Java EE3 with the release of version five. Java EE is defined by its specification. This specification describes the components and middleware services of which Java EE consists. Java EE is the standard for building enterprise applications based on Java technology. Java EE divides the application logic into components according to their function in the application, and the various application components that make up the enterprise application are installed on different machines depending on the tier (Sun Microsystems, 2007). Enterprise applications built with Java EE are called Java enterprise applications. Sub-section 2.2.1 explains these kinds of applications. Sub-section 2.2.2 introduces the Java EE components and sub-section 2.2.3 presents the Java EE middleware services.

2.2.1 Java enterprise application

A Java enterprise application consists of the following layers: client, web, business, persistence and data (Sun Microsystems, 2007). The client and web layer correspond to the presentation layer of the three principal layers (See Figure 3). These layers are responsible for handling user input. The business and persistence layer correspond to the domain layer of the three principal layers. The business layer is responsible for the business logic of the enterprise application. The persistence layer provides an object/relational mapping facility to developers for managing relational data in a Java enterprise application. The data layer is related to the data source layer of the three principal layers (Fowler, Patterns of Enterprise Application Architecture, 2006). Figure 4 shows the layering of a Java enterprise application.

---

3 The next version, Java EE 6, will be expected during the summer of 2009.
The client layer usually runs on a client machine. The web layer, the business layer and the persistence layer are running on the application server. The data layer normally runs on a database server (Fowler, Patterns of Enterprise Application Architecture, 2006). An application server that implements the Java EE specification is called Java application server. Figure 5 illustrates the architecture of a Java enterprise application.

A Web browser, an Application client or a Web service client implements the client layer. An Application client provides a way for users to handle tasks that require a rich user interface. It typically has a GUI. An application client does not use the web layer. The web layer is needed when a Web browser is used. The web layer is implemented by web components. On the business

---

4 An Applet is also a client but this is hardly used anymore so we omit the Applet in this thesis.
5 The GUI is created by the Swing or Abstract Window Toolkit (AWT) API (Sun Microsystems, 2007).
layer run the business components, responsible for the business logic. The persistence layer is implemented by entities. An entity is a Plain Old Java Object (POJO); this means that an entity is a simple Java class. An entity typically represents a table in a relational database. Entity instances correspond to individual rows in the table. The data layer is usually implemented by a database.

Figure 6 shows the several parts of a Java enterprise application.

![Diagram of Java enterprise application parts]

Some of the parts are Java EE components. Sub-section 2.2.2 discusses these components.

### 2.2.2 Java EE components

The following components are defined within Java EE (Sun Microsystems, 2007):

1. **Client components**: Application client.
2. **Web components**: Servlet, Java Server Page (JSP) and Java Server Face (JSF). Servlets and Java Sever Pages both produce HTML pages and receive and respond to requests of Web clients. The difference is that a Servlet is programmatic and a Java Server Page provides a document-centric alternative. With a Java Server Page, a developer can focus more on the actual layout of the HTML page that provides the user interface. Java Server Faces are new within Java EE. This technology offers a complete component framework for building user interfaces.
3. **Business components**: Enterprise JavaBeans (EJBs). The Java enterprise component technology specifies the EJBs. Chapter 3 examines these EJBs.

The Java EE components run in a container. There are three containers available: an Application Client (AC) container, a Web container necessary for the web components and an EJB container for the business components. The AC container is running on a client machine, the EJB and Web container are running on a Java application server.
Figure 7 illustrates these containers.

The containers actually run on a JVM. With the use of the containers, we can accomplish distribution transparency. Distribution transparency is an important aspect of building distributed enterprise systems. With distribution transparency, a client can use a Java EE component, which is running on another Java Virtual Machine as if it is on the same Java Virtual Machine. This means that developers can perform the same activities for local as well as distributed applications. Distribution transparency is achieved with RMI. This middleware service is running under the hood of Java EE. RMI uses stub and skeleton object to provide the connection between the client and the remote object. Figure 8 shows the architecture of RMI.

A stub is a proxy for a remote object, which means that the stub is responsible for forwarding method calls from the client to the server where the actual remote component resides. Therefore, a client's reference to a remote object is actually a reference to the stub. A skeleton is a server-side object, which contains a method that dispatches calls to the actual remote object (Sriganesh, Brose, & Silverman, 2006). Components, which are running on the same JVM, do not use RMI.

The components can access the middleware services. This is handled by the container, which also manages the components.
Figure 9 shows the middleware services related to the containers.

Sub-section 2.2.3 discusses these middleware services.

2.2.3 Middleware services

The interfaces of the middleware services are part of the specification of Java EE. These are the Java EE APIs\(^6\). There is a clear separation between the API interfaces and its implementation. The implementation can be different for each Java application server. The Java EE APIs provide access to the middleware services of the Java application server. The most common middleware services are in the field of persistence, transaction, messaging and security. The Java EE APIs are:

1. Java Persistence API (JPA)
2. Java Transaction API (JTA)
3. Java Message Service (JMS) API
4. Java Authentication and Authorization Service (JAAS) API

Next sub-sections introduce these APIs briefly.

2.2.3.1 Java Persistence API (JPA)

JPA is used for persistence. JPA uses an object-relational mapping approach to bridge the gap between an object oriented model and a relational database. The JPA implements the persistency layer using entities. Entities are managed by the JPA. Entities are not components, because they are not running in a container.

---

\(^6\) Application Programming Interface
JPA defines a standard for:

- The creation of Object Relational Mapping (ORM) configuration metadata for mapping entities to relation tables. We need an ORM framework such as Hibernate or Toplink to implement the O/R mapping.
- The Entity Manager API for performing CRUD (Create, Read, Update and Delete) persistence operations for entities.
- The Java Persistence Query Language (JPQL) for searching and retrieving persisted application data (Panda, Rahman, & Lane, 2007).

JPA can even be used outside the Java EE platform, for example, in Java SE applications (Sun Microsystems, 2008).

### 2.2.3.2 Java Transaction API (JTA)

The JTA defines the core of transaction support. A transaction is a sequence of actions that should all run to complete all or none of them. A transaction has to be atomic, consistent, isolated and durable (ACID) (Sriganesh, Brose, & Silverman, 2006). Atomicity means “all or nothing”; if one part of the transaction fails, the entire transaction fails. Consistency ensures that the database remains in a consistent state before the start of the transaction and after the transaction is over (whether successful or not). Isolation refers to the requirement that other operations cannot access or see the data in an intermediate state during a transaction. Durability refers to the guarantee that once the user has been notified of success, the transaction will persist, and cannot be undone. This means it will survive system failure.

### 2.2.3.3 Java Message Service (JMS) API

The JMS API is a messaging standard that allows components to create, send, receive, and read messages. It enables distributed communication that is loosely coupled, reliable, and asynchronous (Sriganesh, Brose, & Silverman, 2006). The JMS is implemented by message-oriented middleware (MOM) such as IBM MQ Series and Oracle Advanced Queuing. JMS supports two messaging models:

- point-to-point
- publish and subscribe

In the point-to-point model, a producer posts messages to a particular queue and a consumer reads messages from the queue. Here, the producer knows the destination of the message and posts the message directly to the consumer’s queue. The publish and subscribe model supports publishing messages to a particular message topic. Subscribers may register interest in receiving messages on a particular message topic. In this model, neither the publisher nor the subscriber knows about each other. A good metaphor for it is an anonymous bulletin board (Sun Microsystems, 2007).

---

7 Java SE is the Standard Edition of the Java platform.
2.2.3.4 Java Authentication and Authorization Service (JAAS)

The developer has to deal with authentication and authorization provided by the Java Authentication and Authorization Service (JAAS). JAAS is a set of APIs that enables services to authenticate and enforce access controls upon users. JAAS provides a pluggable and extensible framework for programmatic user authentication and authorization. Java Authentication and Authorization Service (JAAS) provides a way for a Java EE application to authenticate and authorize a specific user or group of users to run it. Declarative and programmatic security mechanisms can be used to protect resources (Sriganesh, Brose, & Silverman, 2006).

The Java application server is responsible for providing these middleware services. Sub-section 2.2.4 presents this server.

2.2.4 Java application server

A compliant Java application server implements the Java EE specification. Many Java application servers implement the Java EE specification; Sun Microsystems has certified more than thirty Java application servers (Sun Microsystems, 2008). Popular Java application servers are Sun Glassfish Enterprise Server, JBoss, Oracle Weblogic and IBM Websphere. The difference between the servers is that each offers specific features such as load balancing. Load balancing is a technique to spread work between two or more computers. Java enterprise applications are portable between the different servers unless these features are used.

A compliant Java application server has to provide, among other things, the middleware services and the containers. The EJB container is specified by the Java enterprise component technology. This technology is the focus of this work. Chapter 3 examines this technology.
3 Java enterprise component technology

Enterprise JavaBeans (EJB) technology is the working name for the Java enterprise component technology. Sun Microsystems introduced this technology in 1998 as an extension of its client-side component model JavaBeans. The Enterprise JavaBeans technology is defined as follows: “Enterprise JavaBeans technology is the server side component architecture for Java EE. EJB technology enables rapid and simplified development of distributed, transactional, secure and portable applications based on Java technology” (Sun Microsystems, 2008). The Enterprise JavaBeans technology is the cornerstone of a Java enterprise application. The EJB technology is not intended for the client side, but for server side operations such as transactional business operations.

The current version of the Enterprise JavaBeans technology is version 3.0 (EJB 3.0) which is part of Java EE. The previous version of the Enterprise JavaBeans technology is EJB 2.1, this version is part of the J2EE 1.4 platform, which is the predecessor of Java EE. This chapter examines both versions of the EJB technology, because both versions are used in practice.

The EJB technology specifies the business components. These components are also known as EJB components. Section 3.1 examines these EJB components. Section 3.2 discusses the EJB component model and section 3.3 presents the implementation of this component model, namely the EJB container. Finally, section 3.4 illustrates the difficulties of applying the EJB technology.

3.1 EJB component

A software component is a software element that conforms to a component model and can independently be deployed and composed without modification according to a composition standard (Councill & Heineman, 2001). Within the EJB technology the components are Java classes which conform to interfaces and rules defined by the EJB specification. These components are the Enterprise JavaBeans (EJBs). There are three types of Enterprise JavaBeans within version 2.1 (Sierra & Bates, 2003; Sriganesh, Brose, & Silverman, 2006):

1. Entity beans
2. Session beans
3. Message-driven beans
1. An entity bean represents persistent data maintained in a database. Each entity bean usually has one underlying table in a relational database, and each instance of the bean corresponds to a row in that table. An entity bean is recognizable as the “noun” in a particular business domain: for example an order, a customer or a product.

2. A session bean models business processes. Session beans are like “verbs” because they perform actions. Session beans are used for price quoting, order entry, video compression, stock trades, banking transactions, complex calculations etc. A session bean is different from an entity bean, as it does not represent persistent data. A session bean can be stateful or stateless. A stateful session bean keeps the state of a session, which means that the server remembers the previous actions of a client. Keeping the state of a session is an important issue in an enterprise application. The shopping cart of a web shop is a well-known example of keeping the session state; the contents of the shopping basket remains available throughout the whole session (Panda, Rahman, & Lane, 2007). A stateless session bean does not keep state and therefore is the same for each client.

3. A message-driven bean is used to process messages. Asynchronous communication is build with messages. The difference between a session and a message-driven bean is that a session bean is called synchronously by simply calling a method of the session bean and a message-driven bean is called asynchronously by sending a message. Clients never invoke a message-driven bean directly. The message-driven bean is integrated with JMS (See sub-section 2.2.3.3). The message-driven bean consumes and processes asynchronous JMS messages. Figure 10 shows the different beans.

![Figure 10: Enterprise JavaBeans](image)

In EJB 3.0, there are only two components available: the session bean (stateless or stateful) and the message-driven bean. EJB 2.1 entity beans can still be used within EJB 3.0, but it is recommended to use the Java Persistence API (JPA). This API is the replacement of the entity beans. The entity bean has many drawbacks such as not supporting OO features like inheritance and polymorphism (Panda, Rahman, & Lane, 2007). The JPA solves these issues.

---

8 A client can be a Servlet, JSP, JSF or even another EJB.
The life cycle of the components consists of three main phases (Wang & Lau, 2006):

1. Design phase: components are constructed in source code.
2. Deployment phase: component executables are created and deposited into the EJB container.
3. Run-time phase: component executables are instantiated with initial data, so that they have states and are ready for execution.

In the design phase, the EJBs are written. To write these EJBs the developer has to follow the EJB specification. This specification defines the component contract. The developer should implement all the elements defined in this contract. Figure 11 illustrates these elements.

![Figure 11: Component contract](image)

The component contract compels the developer to write the necessary interfaces, the business logic, which is implemented by the Enterprise JavaBean class and the necessary environment information. Message-driven beans do not have interfaces because clients do not invoke them. The EJB specification provides blueprints for building the Enterprise JavaBeans. These blueprints guarantee that the EJBs follow the EJB component model. Section 3.2 explains this component model.

### 3.2 EJB component model

This section examines the EJB component model using the framework defined by Heineman and Councill (Councill & Heineman, 2001). Table 1 presents this framework, which addresses the basic elements of a component model.

<table>
<thead>
<tr>
<th>Standards for</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interfaces</td>
<td>Specification of component behaviour and properties; definition of Interface Description Languages (IDL).</td>
</tr>
<tr>
<td>Naming</td>
<td>Global unique names for interfaces and components.</td>
</tr>
<tr>
<td>Meta data</td>
<td>Information about components, interfaces, and their relationships; APIs to services providing such information.</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Interoperability</td>
<td>Communication and data exchange among components from different vendors, implemented in different languages.</td>
</tr>
<tr>
<td>Customization</td>
<td>Interfaces for customizing components. User-friendly customization tools will use these interfaces.</td>
</tr>
<tr>
<td>Composition</td>
<td>Interfaces and rules for combining components to create larger structures and for substituting and adding components to existing structures.</td>
</tr>
<tr>
<td>Evolution Support</td>
<td>Rules and services for replacing components or interfaces by newer versions.</td>
</tr>
<tr>
<td>Packaging and Deployment</td>
<td>Packaging implementation and resources needed for installing and configuring a component.</td>
</tr>
</tbody>
</table>

Next sub-sections discuss these elements.

### 3.2.1 Interfaces

An interface serves as a contract between the client and the component. Developing a session bean or entity bean within EJB 2.1 two interfaces have to be defined, namely: a home interface and a business interface. The clients first invoke the home interface. This interface represents lifecycle methods (create, remove and find\(^9\)) needed at the run-time phase of the EJBs. The business interface represents the business methods of the Enterprise JavaBean class, which are exposed to the client. Section 3.3.1 clarifies the purpose of these interfaces.

In addition, the interfaces have to be defined as local, remote or both. Clients that are running on the same Java Virtual Machine (JVM) as the Enterprise JavaBean use local interfaces. Clients that are not running on the same JVM must use remote interfaces. Applying a remote interface, Remote Method Invocation (RMI) is used for the communication. The EJB specification provides blueprints for building these interfaces. The home and business interface fill in the interface part of the component contract.

Within EJB 3.0, many enhancements have been made. A major benefit of EJB 3.0 is the simplified programming model. The developer only has to define the business interface for EJBs 3.0.

\(^9\) The method find is only used for entity beans.
3.2.2 Naming

The Java Naming and Directory Interface (JNDI) API provides naming and directory functionality. This mechanism is necessary to provide the different components and resources of a unique name and to find components and resources by this name. JNDI is also needed for composing the components. Using JNDI, a Java enterprise application can store and retrieve any type of named Java object. Because JNDI is independent of any specific implementations, applications can use JNDI to access multiple naming and directory services, including existing naming and directory services such as Lightweight Directory Access Protocol (LDAP) and Domain Name System (DNS) (Sun Microsystems, 2008).

3.2.3 Meta data

The deployment descriptor is used for putting meta data to the Enterprise JavaBean class. A deployment descriptor is an XML file that provides structural, assembly and deployment information. For example, the developer can put the following line in the deployment descriptor: `<session-type>Stateless</session-type>`.

With this line, a session bean is defined as stateless. The deployment descriptor provides the environment information of the component contract. In EJB 3.0, annotations are used to put meta data to the EJBs. An annotation is a special form of syntactic meta data that can be added to Java source code. With annotations, information about the Java code can be expressed in the Java code. For example, with the annotation `@stateless`, a session bean is defined as stateless. There are many annotations available, which makes the deployment descriptors redundant.

In addition, we can use interceptors. Interceptors are methods that are invoked automatically when the methods of a bean are invoked. Interceptors are often used for logging and security issues. Interceptors are a basic form of Aspect-Oriented Programming (AOP) (Sriganesh, Brose, & Silverman, 2006). AOP is a programming paradigm, which attempt to aid programmers in the separation of concerns (Wikipedia, 2008).

3.2.4 Interoperability

Interoperability means communication and data exchange between components from different vendors, implemented in different languages. Java RMI provides a RMI-IIOP bridge for connecting with CORBA applications. IIOP (Internet Inter-ORB Protocol) is a protocol that makes it possible for distributed programs written in different programming languages to communicate over the Internet. Within Java EE, the Java RMI-IIOP API is usually used instead of the Java RMI API.

---

10 CORBA is a component model defined by the Object Management Group (OMG).
A stateless session bean can also be exposed as a web service; this is called an endpoint interface. A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards (W3C, 2004). Using Web services the enterprise application can communicate with other applications regardless of location or platform.

In addition, the JVM provides interoperability as well, as the JVM guarantees platform independency.

### 3.2.5 Customization

In the definition of Heineman and Councill, customization means the availability of interfaces for customizing components. User-friendly customization tools use these interfaces. This is certainly true for the JavaBeans technology, but not for the Enterprise JavaBeans technology. Within this technology, there are only customization possibilities during deployment using the deployment descriptor.

### 3.2.6 Composition

Composition means combining components to create larger structures and also substituting and adding components to existing structures in order to achieve desired behaviour. The composition of the EJBs is performed during the design phase by method calls (Wang & Lau, 2006). Figure 12 shows a methodXXX of session bean A making a call to methodYYY of session bean B.

![Figure 12: Composition](image)

In EJB 2.1 the only way to lookup other objects was by using JNDI. In EJB 3.0 dependency injection (DI) can also be used; a powerful and easy to use mechanism for injecting references. This is a design pattern, also known as “inversion of control” (Fowler). The container is responsible for injecting the resources to the java code. Dependency injection is the inverse of JNDI. The benefit of DI is that the developer does not have to write the lookup code.
3.2.7 Evolution support

The Enterprise JavaBeans can be replaced by newer versions. This way the EJB technology supports evolution of enterprise applications.

3.2.8 Packaging and deployment

A Java enterprise application is packaged as a jar file with an .ear extension. EJB module and the Web module are part of this package. The EJB module (the Enterprise JavaBean classes, interface classes and deployment descriptor) is packaged as one or more jar files with the .jar extension. The Web module containing the web components is packaged as a jar file with the .war extension. The .ear file is the deployment file (Sriganesh, Brose, & Silverman, 2006). Figure 13 presents the different package and deployment files.

![Figure 13: Package and deployment](image)

3.3 EJB container

An EJB container is the implementation of the EJB component model. A component model implementation is the dedicated set of executable software elements required to support the execution of components that conform to the component model [4]. After the developer has built the Enterprise JavaBeans and has fulfilled all the requirements of the component contract, the developer deploys the components to the EJB container.

The container provides access to the necessary middleware services. The EJB uses an implicit approach to access the several services. In EJB 2.1, the developer uses a deployment descriptor to inform the container about the middleware services needs. Figure 14 shows the interaction between the container and deployment descriptor.

![Figure 14: Deployment descriptor](image)
In EJB 3.0, we can simple use annotations to inform the container about the middleware needs. We can also use the middleware services explicit by using the Java APIs. The EJB container executes the EJB components. Sub-section 3.3.1 presents an example of how the EJB container executes a stateless session bean.

### 3.3.1 Execution of a stateless session bean

We are going to build a stateless session bean with remote interfaces. First, we build this with EJB 2.1. To build a stateless session bean in EJB 2.1 we have to define a home interface, a business interface, the Enterprise JavaBean class and a deployment descriptor. To write the interfaces we have to extend interfaces, which have been described by the EJB specification. To define the remote home interface we have to extend the javax.ejb.EJBHome interface and for the remote business interface, we have to extend the javax.ejb.EJBObject interface. Within the home interface we must declare a method create() without arguments. Within the business interface, we declare the business methods we want to expose to the clients. To write the Enterprise JavaBean class we have to implement the javax.ejb.Session interface and its required methods. In the deployment descriptor, we define besides other things that the session bean is stateless.

If we deploy the component to the EJB container, the container generates the implementation of the home and the business interface: For the remote home interface, the container generates an EJBHomeObject and the business interface the container generates an EJBObj ect. Figure 15 shows the results (the objects with the bold border are generated by the EJB container):

The communication between the several objects is as follows: First, the client needs to locate the home object. The client uses JNDI to obtain this reference. Next, the client calls the method create() of the EJBHomeObject to create the EJBObj ect. A reference to the EJBObj ect is returned to the client. The client calls a business method to the EJB object. The client never invokes the method of an enterprise bean instance directly. The invocation is intercepted by the EJB container and then delegated to the session bean instance. By intercepting requests, the EJB container can
provide middleware services defined in the deployment descriptor implicitly. The EJBObeject returns the values obtained by the business call.

The EJB container manages the stateless session bean. The stateless session bean living within the EJB container needs to be created or destroyed when respectively client traffic increases or decreases. It provides a complete life cycle management for the stateless session bean. Figure 16 illustrates the lifecycle for a stateless session bean.

![Figure 16: Stateless session bean lifecycle](image)

To build the same stateless session bean and a remote interface within EJB 3.0, we only have to define a business interface and an Enterprise JavaBean class. We use an annotation to define the interface as remote and use an annotation within the Enterprise JavaBean class to define the class as a stateless session bean. The EJB container internally uses wrapper classes to provide the middleware services to the Enterprise JavaBeans (Sriganesh, Brose, & Silverman, 2006). Figure 17 shows the EJB 3.0 architecture of a stateless bean with remote interface.

![Figure 17: EJB 3.0 architecture](image)
This architecture is much simpler than the EJB 2.1 architecture. EJBs 3.0 are Plain Old Java Objects (POJO) and the interfaces are a Plain Old Java Interfaces (POJI). This means that the Enterprise JavaBean and interface classes are simple Java classes. They do not have to implement or extend several interfaces. Using EJB 3.0 makes development and testing more easily.

### 3.4 Difficulties of applying the EJB technology

This section argues that applying the EJB technology has its difficulties. Several beans are available: stateful session bean, stateless session bean and the message-driven bean. We need to apply the right bean for the right situation. For example, choosing a stateful session bean where a stateless session bean would be sufficient has dramatic impact on the performance of the Java enterprise application (Blankestijn, 2007). To apply the EJBs we also need to master the component model. This means we need to master interfaces, naming, meta-data, interoperability, customization, composition, evolution support, packaging and deployment of the EJBs.

Another difficulty are the several protocols available to access the Enterprise JavaBeans, namely HTTP, RMI-IIOP and SOAP. Different kinds of clients use these protocols. The client could be a Servlet, JSP or JSF using the HTTP protocol, or an Application client using the RMI-IIOP protocol or even a Web service client using the SOAP protocol. These multiple ways to access the Enterprise JavaBeans have its impact on the implementation of the EJBs and the interfaces. Knowledge and experience with the several clients is essential to make the right choices.

In addition, the provided middleware services reduce the complexity of developing a Java enterprise application, but applying these middleware services within the EJB components remains complex. The difficulties of applying the most used APIs (JPA, JMS, JTA and JAAS) are:

- **JPA**: We need to master the Entity Manager and JPQL for accessing the entities. In addition, we need to understand the object-relational mapping. This consists of mapping the tables, columns, keys, relations, sequences etc. We need to know how this mapping is done in order to make the right configurations such as lazy versus eager loading and cascading issues (Blankestijn, 2007).

- **JMS**: To apply JMS and the message-driven bean we need to understand the messaging concepts like the point-to-point message model, the publish-subscribe message model, queues and message filters.

- **JTA**: There are two ways to use the JTA within the EJBs: container-managed transaction (CMT) or bean-managed transaction (BMT). In CMT, the EJB container manages the transactions. The container starts a transaction before a business method is invoked, invokes the method, and depending on what happened during the method call, either commits or rolls back the managed transaction (Panda, Rahman, & Lane, 2007). This is the default mechanism of Java EE. This mechanism is enabled by using annotations or a
deployment descriptor. In most situations, this is sufficient. However, in specific cases more control of the transactions is needed. For example, if the clients need to manage the transactions instead of the container. For this purpose, BMT is used, which means that the transactions need to be explicitly managed using the JTA. This allows the developer to demarcate transactions in a manner that is independent of the default implementation of the Java application server. To choose between CMT and BMT and to make the right configurations it is essential to understand the complex transaction theory behind it. This theory consists of optimistic and pessimistic locking, local and distributed transactions, transactions scope, transaction attributes and the different isolation levels of transactions (Srganesh, Brose, & Silverman, 2006; Panda, Rahman, & Lane, 2007).

- **JAAS:** The Java security APIs span a wide range of areas, including cryptography, public key infrastructure, secure communication, authentication, and access control. Java security technology provides the developer with a comprehensive security framework for writing applications, and gives the user or administrator a set of tools to securely manage applications (Sun Microsystems, 2008). However, mastering this framework is difficult and time-consuming.

Applying the Java enterprise component technology is a complex task. We need a method to support us with the development of the tasks for instruction to master this technology. Chapter 4 presents such a method.
4 Ten steps to complex learning

In this chapter, we introduce the “Ten steps to complex learning” method.

4.1 Three persistent problems

The “Ten Steps” method uses a holistic design approach, which is the opposite of an atomistic one. This approach offers a solution for three persistent problems in the field of education, namely: compartmentalization, fragmentation, and the transfer paradox (van Merriënboer & Kirschner, 2007):

- **Compartmentalization** means separation of the whole into distinct parts or categories.
- **Fragmentation** is a process of breaking something into small and incomplete parts.
- The **transfer paradox** describes the following phenomenon: methods that work best for reaching isolated, specific objectives are often not the methods that work best for reaching integrated objectives and increasing transfer of learning.

Next sections discuss the ten steps.

4.2 The ten steps to complex learning

The “Ten Steps” method consists of the following ten steps:

1. Design learning tasks
2. Sequence task classes
3. Set performance objectives
4. Design supportive information
5. Analyze cognitive strategies
6. Analyze mental models
7. Design procedural information
8. Analyze cognitive rules
9. Analyze prerequisite knowledge
10. Design part-task practice

These steps can be applied in random order.

---

11 In an atomistic approach, complex contents and tasks are continually reduced to simpler elements.
Figure 18 illustrates the relationship between the ten steps (van Merriënboer & Kirschner, 2007).

![Diagram of the ten steps]

**Figure 18: A schematic overview of the ten steps**

### 4.3 Step 1: Design learning tasks

Step 1 is the design of the learning tasks. Learning tasks are authentic whole task experiences based on real life tasks that aim at the integration of skills, knowledge, and attitudes. The learning tasks are illustrated by the circles in Figure 18. The triangles in the circles, each time on different places, illustrate the variability between the different learning tasks. Variability is important to achieve transfer of learning. It is not possible that the students can perform a whole task on their own right from the beginning. Therefore, we provide the students task support. Figure 19 shows the difference between task support and the problem solving process.
With task support the given state, the goal state and the solutions are provided by the instructions. The problem solving process is what is going on in the mind of the student. The difference can be shown by an example of chess players; the initial state of chessboard is the given state. The sequence of moves represents the solution. The final state of the board is the goal state. To decide which move to perform is in the player’s head, which is the problem solving process. We need to provide task support for the learning tasks.

We start with a high level of support and guidance and end with tasks without support and guidance. Figure 20 illustrates the level of support indicated by the level of grey in the circles.

Offering students whole task practice for each learning task, integration instead of compartmentalization will be achieved, which solves the first persistent problem.

4.4 Step 2: Sequence tasks classes

Step 2 is sequencing the task classes. Task classes only differ in difficulty; the dotted boxes in Figure 18 represent the task classes. The different learning tasks are categorized in easy to difficult task classes. A task class is a category of several learning tasks. A task class consists of learning
tasks where the difficulty is the same. The easy to difficult task classes offering coordination instead of fragmentation solving the second persistent problem.

4.5 Step 3: Set performance objectives

Step 3 examines the performance objectives of the instructions. In this step, we analyze the necessary skills and categorize them as recurrent or non-recurrent. This categorization is important for the design of the supportive and procedural information.

4.6 Step 4, 5 and 6: Design supportive information

Step 7 is the design of supportive information. Supportive information explains how a domain is organized and how problems in that domain are approached. The supportive information is needed for the non-recurrent skills. The supportive information is specified per task class and is always available to students. The supportive information of a new task class builds on the information of the previous task class. All learning tasks of a task class can be performed with the same supportive information. Step 5 (Analyze cognitive strategies) and 6 (Analyze mental models) provide the basis for the design of the supportive information, in the case that it is not already available in existing learning material.

4.7 Step 7, 8 and 9: Design procedural information

Step 7 is the design of procedural information. Procedural information is prerequisite for learning and performing routine aspects of learning tasks (recurrent skills). It specifies exactly how to perform the routine aspects of the task and is best presented just in time; precisely when the student needs it. It is quickly faded as students gain more expertise. Step 8 (Analyze cognitive rules) and 9 provide the basis for the design of the procedural information. The results of step eight are if-then rules or procedures. Step 9 (Analyze prerequisite knowledge) analyzes the prerequisite knowledge necessary to perform these if-then rules or procedures.

4.8 Step 10: Design part-tasks practice

Step 10 is the design of part-task practice. Part-tasks practice items provided to students help them to reach a high level of automaticity for selected routine aspects of a task. Part-tasks practice typically provides huge amounts of repetition. Contrary to the other design steps, this step is not necessary. Only when a very high level of automaticity is required this step is performed. If the tasks provides supportive, procedural information and part-task practice (if necessary), the students will not be overwhelmed with the complexity of the task; transfer of learning will be achieved.
Part II: Development of tasks
5 Analysis and design

Next sections apply the “Ten Steps” method to the EJB technology. All the steps are performed, which result in a training blueprint. With this blueprint, we can actually develop the tasks for instructions. The steps are described in a linear way. In practice, we have used an iterative and incremental approach. Section 5.1 starts with the first step of the method, namely: the design of the learning tasks.

5.1 Step 1: Design learning tasks

To achieve transfer of learning the learning tasks must fulfil the following aspects: real life task, whole task practice, support and variability (van Merriënboer & Kirschner, 2007). Next subsections examine these aspects.

5.1.1 Real life task

The learning tasks have to be real life tasks. In real life, enterprise applications are built from requirements. Therefore, we provide the students requirements in each learning task. There is no one correct solution, more solutions are correct. This is characterising the real life task. In addition, we have to use a real-life task environment. We need an Integrated Development Environment (IDE) for building a Java enterprise application. The choice for NetBeans and corresponding Glassfish application server (NetBeans) is straightforward, because Sun Microsystems develop these tools, therefore the last specification of Java EE is always supported.

5.1.2 Whole task practice

An important aspect of the learning tasks is the whole task practice. Just applying the EJB technology is not a whole task in real life. The EJB technology is part of Java EE and the aim of Java EE is to build Java enterprise applications. Hence, the complete task is the development of a Java enterprise application. Therefore, in each learning task the students build a complete Java enterprise application. To guarantee that the students practice all the layers of the Java enterprise application we use the CRUD requirements. Each learning task needs to address at least one CRUD operation. During the development, the students perform the complete life cycle of component-based software systems consists, namely (De Cesare, Lycett, & D. Macredie, 2006; Ivica Crnkovic):

---

12 CRUD requirements are functional requirements. We focus in the learning tasks only on the functional requirements and not on the non-functional requirements.
• **Component qualification:** is the process for determining the suitability of a component for use within the intended final system.

• **Component adaption:** make the components suitable for the intended system.

• **System assembly:** is the integration of components through a well-defined infrastructure. Java EE provides this infrastructure.

• **System evolution:** Replacement, addition and modification of components.

### 5.1.3 Task support

It is not possible that the students can build a complete Java enterprise application on their own right from the beginning. We use the “completion strategy” (van Merriënboer & Kirschner, 2007). In this strategy, students first study a case study, then work on completion tasks and finally perform conventional tasks. For the case study, the instructions provide the given state, goal state and the complete solution. For the completion tasks the given and goal state are provided. For the solutions only a part is given. The students must complete the solutions themselves. For the conventional task, only the given state is provided. The goal state and the solution are not given. The students have to find the solutions all by themselves.

Table 2 shows the completion strategy.

*Table 2: Completion strategy*

<table>
<thead>
<tr>
<th></th>
<th>Case study</th>
<th>Completion</th>
<th>Conventional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Given state</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Goal state</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Solution</td>
<td>+</td>
<td>Complete</td>
<td>Find</td>
</tr>
</tbody>
</table>

To achieve transfer of learning we also have to vary between the learning tasks. Sub-section 5.1.4 explains this variability.
5.1.4 Variability

Variability means that the students get different requirements for each learning task. We use the CRUD requirements to vary between the learning tasks. In addition, we want the students to practice the different kinds of components (stateless, stateful and message-driven beans). Therefore, we need bean specific requirements that force students to use a message-driven bean or stateful/stateless session bean. We do not ask the students for example to build a stateful session bean, because that would not be a real life task. Instead of this, the students get requirements on which they can decide to use a stateful session bean.

Next step, number 2, categorizes the learning tasks.

5.2 Step 2: Sequence tasks classes

We have to decide how to differ between the task classes. We use the simplifying conditions method for designing easy-to-difficult task classes. To use this method, we have to find conditions on which we can categorize the tasks classes (van Merriënboer & Kirschner, 2007). The first idea was to separate the task classes on the layers of the enterprise application. For example, task class one handles the business layer, task class two the web layer etc. However, this is not a whole task in practice so we skipped this idea. Another idea was to separate the task classes using the CRUD requirements. The first task class addresses the “retrieve” functionality, the second “create” and “delete” and the third task class all the CRUD requirements. However, we need to vary between the requirements, so this was not a good idea either. The third idea was to separate on database tables, the first class handles enterprise applications based on one table, the second on two tables and so on. By using one table for the first task class, gave us the opportunity to let the students build all layers of a Java enterprise application without the complexity of multiple tables. Nevertheless, one or two database tables within an enterprise application are not realistic so we decide that each learning task has to consist of at least three tables. With this amount of classes, we can provide real life experience within a reasonable amount of time. The fourth idea was to focus on the middleware services; each task class handles some of the services. The problem with this idea was that enterprise applications are built from requirements. We cannot ask the students to build a BMT transaction for example. This would not be a realistic requirement.

The solution is straightforward. Looking at the architecture of a Java enterprise application it is simply to differ on the clients (See Figure 21). In this case, all the layers of the enterprise application are addressed. Task class one addresses the Web browser, task class two also includes the Application client and in task class three the Web service client is added. Figure 21 illustrates the three different task classes.
The Web browser uses the HTTP protocol to communicate with the Java application server, the Application client using the RMI-IIOP protocol to communicate and the Web service clients using the SOAP protocol. Table 3 presents the task classes looking at the communication protocols.

Table 3: Communication protocols of the task classes

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Task class 1</th>
<th>Task class 2</th>
<th>Task class 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HTTP</td>
<td>HTTP</td>
<td>HTTP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RMI-IIOP</td>
<td>RMI-IIOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SOAP</td>
</tr>
</tbody>
</table>

In each task class, the first learning task consists of a case study and the last learning task is a conventional task. In each learning task, the students build a complete Java enterprise application.

Section 5.3 addresses the objectives of the tasks.
5.3 Step 3: Set performance objectives

The main objective for the students is to develop a Java enterprise application. To perform this task, the students need several skills. A Java enterprise application consists of several layers namely: the client, web, business, persistence and data layer. These layers have to be implemented by the several Java EE parts (See Figure 6). Figure 22 presents the skill hierarchy, which organizes the several skills needed to develop a Java enterprise application.

The skill hierarchy shows all the necessary skills to develop a Java enterprise application. The relationship between “implementing the web layer” and “implementing the client layer” is transposable indicated by the double-headed dotted arrow. This means that these two skills can be performed in any desired order. In addition, we can easily omit one of the two clients. Each skill is a performance objective.

Figure 22: Skill hierarchy

The skill hierarchy shows all the necessary skills to develop a Java enterprise application. The relationship between “implementing the web layer” and “implementing the client layer” is transposable indicated by the double-headed dotted arrow. This means that these two skills can be performed in any desired order. In addition, we can easily omit one of the two clients. Each skill is a performance objective.
Table 4 classifies which objectives are recurrent and which are non-recurrent. The table also defines which objectives are not parts of the instructions.

Table 4: Classifying performance objectives

<table>
<thead>
<tr>
<th>Performance objectives</th>
<th>To be Taught</th>
<th>Not to be taught</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Non-recurrent</td>
<td>Recurrent</td>
</tr>
<tr>
<td>Developing a Java enterprise application</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Implementing the data layer</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Implementing the persistence layer</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Implementing the business layer</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Implementing the web layer</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Implementing the client layer</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Building an entity class</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Building Enterprise JavaBeans</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Building a stateless session bean</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Building a stateful session bean</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Building a message-driven bean</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Building a Servlet</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Building a Java Server Page</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Building a Java Server Face</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Building an Application Client</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

Explanation of the “not to be taught” objectives:
The skill “implementing the data layer” is not taught. This requires a complete different competency. Our goal is to teach the Java enterprise component technology. Therefore, implementing the data layer is not taught. The students get a predefined database. In addition, we exclude the building of Java Server Faces. The reason for excluding Java Server Faces is that this technology is very extensive. Teaching this does not contribute for teaching the Java enterprise component technology. As web components, we use Servlets and Java Server Pages; hence, we do not need the Java Server Face technology.

Explanation of the recurrent and non-recurrent objectives:
In practice all, the “implementing” skills are non-recurrent. Our goal is to teach the students component-oriented programming applying the EJB technology. Therefore, we classify the “implementing the business layer” skill as non-recurrent and the other skills as recurrent. This way the students can focus on the business layer of the enterprise application without omitting the other layers. The main skill “Developing a Java enterprise application” is always non-recurrent.

Implementing the business layer is the main objective of the instructions. Therefore, this skill gets special attention in relation to implementing the web layer for example. This is because the web layer is only used for performing the complete task; they are not taught intensively. On the
contrary to the EJBs, the implementation of the business layer, are the core of the Java enterprise component technology. For each kind of EJB, students need to learn all the standards of the component model (see section 3.2), namely: interfaces, naming, meta-data, customization, composition, interoperability, evolution support, packaging and deployment (Councill & Heineman, 2001). Table 5 shows the Enterprise JavaBeans objectives.

### Table 5: Objectives of the Enterprise JavaBeans

<table>
<thead>
<tr>
<th>Subjects</th>
<th>Objectives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interfaces</td>
<td>Use local, remote interface and web service interface.13</td>
</tr>
<tr>
<td>(only concerns the session beans)</td>
<td></td>
</tr>
<tr>
<td>Naming</td>
<td>Use the Java Naming and Directory Interface (JNDI).</td>
</tr>
<tr>
<td>Meta data</td>
<td>Use annotations.</td>
</tr>
<tr>
<td>Customization</td>
<td>Use the Deployment descriptor.</td>
</tr>
<tr>
<td>Composition</td>
<td>Use dependency Injection.</td>
</tr>
<tr>
<td>Evolution support</td>
<td>Replacing the components.</td>
</tr>
<tr>
<td>Packaging and deployment</td>
<td>Package the components into Jar files.</td>
</tr>
<tr>
<td></td>
<td>Deploy the components to the container.</td>
</tr>
</tbody>
</table>

Section 5.4 and 5.5 address the design of the necessary information to perform the learning tasks.

### 5.4 Step 4, 5 and 6: Design supportive information

The supportive information helps students to perform the non-recurrent skills. For this purpose, we use cognitive feedback. Students are asked to compare their own solutions with provided solutions. At the end of each task class, we put a feedback task. We also need to provide the students with general information on how to solve problems. Therefore, we perform step five “analyze cognitive strategies” and step six “analyze mental models”. The cognitive strategies (step five) are the design patterns. Design patterns are best practices to commonly occurring problems (Alur, Crupi, & Malks, 2003). These patterns are presented within the case studies. We use existing instructions to provide the mental models (Step 6). We choose to use the book Mastering Enterprise JavaBeans 3.0 of Sriganesh, R. P., Brose, G., & Silverman, M. (Sriganesh, Brose, & Silverman, 2006).

---

13 The Web service client is part of task class three and therefore not part of this thesis.
5.5 Step 7, 8 and 9: Design procedural information

The procedural information helps students to perform the recurrent skills. Step 8 “analyze cognitive rules” generates the necessary how to rules. For example, how to build a message-driven bean. To build these rules we need to know what the prerequisite knowledge is of the students. Step 9 “analyze prerequisite knowledge” is used for this purpose. In our situation the primary target are students studying computer science at university level. Therefore, the prerequisite knowledge is the Bachelor Computer Science of the Open Universiteit Nederland. In general, the students need the following basic knowledge to successfully fulfil the learning tasks: HTML, Java, SQL, Servlets Java Server Pages and knowledge of the Swing framework.

5.6 Step 10: Design part-tasks practice

With this step, we can reach a high level of automation of recurrent task aspects. We do not need automaticity of the skills: hence, this step is omitted.

5.7 The training blueprint

In this thesis, we develop task class one and two. The development of task class three is put forward for future work. The time for the students is limited, so we need to restrict the number of learning tasks. We design five learning tasks and in each task, the students receive different CRUD and bean specific requirements. Figure 23 shows schematically the five tasks in which the triangles indicating the variability between the tasks.

![Figure 23: Variability](image)

We decide to use three learning tasks for task class one en two for task class two. In each task, we need at least the stateless session bean and the retrieve functionality to get the information out of the database. The implementation of these requirements is demonstrated in task 1. In task 2, the students practice the update requirement and the implementation of the message-driven bean. The students have to build the stateless session beans and retrieve functionality completely on their own. Partial solutions for the implementation of the update requirement and Message driven bean are given. In task 3 the students practice the create requirements and the implementation of the stateful session bean. Partial solutions are given. We have decided to skip the conventional task in
this task class otherwise; it will take too much time for the students. Task 4 is part of a new task class in which the building of the application client is demonstrated. The students only implement the stateless session bean and the retrieve functionality in this task. Task 5 is a conventional task, which means that no support is given. The students have to be able to implement all the CRUD requirements and the stateless session bean using the Application client and the Web client. Figure 24 shows the design of the task classes including the learning tasks and the requirements for each task.

Now, we are able to complete the training blueprint. Table 6 shows the blueprint of task class one and Table 7 the blueprint of task class two.

**Table 6: Blueprint task class one**

<table>
<thead>
<tr>
<th>Task Class 1:</th>
<th>Task two</th>
<th>Task three</th>
</tr>
</thead>
<tbody>
<tr>
<td>Students are confronted with requirements to build a Java enterprise application with a web client.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Supportive information:**
Chapter 1,3,4,5,6,7,8 of the book Mastering Enterprise JavaBeans 3.0 of Sriganesh, R. P., Brose, G., & Silverman, M. (Sriganesh, Brose, & Silverman, 2006)

**Learning task 1.1: Case study**
Requirements:
- Retrieve
- Stateless session bean

The students have to build a Java enterprise application with a web client. Students receive the requirements and the database. They get the complete solution by a case study.

**The following is demonstrated in the case study:**
- Define a NetBeans project
- Defining the Persistence Unit
- Building entity classes
- Building a stateless session bean facade
- Building a Servlet
- Building a Java Server Page
- Calling an Enterprise JavaBean
- Run the Enterprise Application
Learning task 1.2: Completion
Requirements:
- Retrieve / Update
- Stateless session bean
- Message-driven bean

The students have to build a Java enterprise application with a web client. Students receive the requirements and the database. Partial solutions are given.

Procedural information
- Building a message-driven bean
- Message producer
- Implementing the update requirement

Learning task 1.3: Completion
Requirements:
- Retrieve / Create
- Stateless session bean
- Stateful session bean

The students have to build a Java enterprise application with a web client. Students receive the requirements and the database. Partial solutions are given.

Procedural information
- Implementing the create requirement

Supportive information: Feedback
Students receive the solutions of learning tasks 1.1, 1.2 and 1.3.

Table 7: Blueprint task class two

Task Class 2:
Students are confronted with requirements to build a Java enterprise application with a web client and application client.

Supportive information:

Learning task 2.1: Case study
Requirements:
- Retrieve
- Stateless session bean

The students have to build a Java enterprise application with an Application client. Students receive the requirements and the database. They get the complete solution by a case study.

The following is demonstrated in the case study:
- Building an Application client

Learning task 2.2: Conventional
Requirements:
- Create / Retrieve / Update / Delete
- Stateless session bean
The students have to build a Java enterprise application with a web client. Students receive the requirements and the database. No solutions are given.

Supportive information: Feedback
Students receive the solution of learning tasks 2.1 and 2.2.

With this blueprint, we are ready to develop the tasks. Chapter 6 and chapter 7 discuss respectively the development of the tasks one and task two.
6 Development of task part one

The “Ten Steps” method describes some general guidelines for developing the tasks for instructions, but there are no restrictions; we can use or build any learning environment. We choose to build a simple web environment so that the tasks can be easily integrated in the existing learning environments and be accessed by any computer using the Internet. This chapter focuses on the development of task class one. The development of task class two is addressed the next chapter.

6.1 Development of the task classes

We develop a Web page containing the two task classes and the several learning tasks. Figure 25 shows the first page of the web environment.

![Figure 25: Start page web environment](image)

In the development phase, the task classes are called assignments. Assignment one addresses “Developing web-based enterprise applications” and assignment two “Developing web-based enterprise applications with application client”. Assignment one consists of three tasks and assignment two of two tasks. The final task of each assignment is the feedback task (See section 5.4). Section 6.2 presents the first task of assignment one.
6.2 Task one: World enterprise application

In this task, the students have to build a World enterprise application. Figure 26 shows a screen-print of this task.

![Screen print of Task one World enterprise application](Image)

This task is a case study, therefore it consists of a given and goal state and it includes the complete solution (See sub-section 5.1.3). On the right hand site, we find the general information (See section 5.4). From each task, this information has to be reachable (van Merriënboer & Kirschner, 2007).

The task starts with an introduction, telling about the customer. This is necessary to provide the students a real-life task (See sub-section 5.1.1). The following introduction is given for task one:

Statistics Finland (www.stat.fi) operates administratively under the Ministry of Finance, but is fully and independently responsible for its activities, services and statistics. Statistics Finland has personnel of around 1,100, of whom 200 are employed as statistical interviewers. In addition to its head office in Helsinki, Statistics Finland has regional service offices in Turku, Tampere, Seinäjoki and Oulu. Statistics Finland combines collected data with its own expertise to produce statistics and information services for the needs of society and to develop national official statistics.
6.2.1 Requirements

The next section of this task is the requirements, which are in fact the given state. The following requirements are given for this task:

Statistics Finland needs an information web system that helps find out which languages are spoken in a given country or which cities are situated in a given country.

Statistics Finland provides a world.sql file containing the create table and the insert statements and an ERD diagram document containing the ERD diagram of the database. The data used in the world database is copyright Statistics Finland, http://www.stat.fi/worldinfigures.

The architecture of the system has to be multi-tier; the clients use a web browser, the application is running on a glassfish application server and the database on a MySQL server.

A world.sql file and an ERD diagram are provided. It is not necessary that the students design the database. The students can simply create the database by using the world.sql script (the database is extracted from the MySQL developer site (MySQL)). An ERD diagram is also provided to give the student more insight in the database. Figure 27 shows the ERD diagram of the World database (the ERD diagram is generated by MySQL Workbench).

![ERD diagram](image)

Figure 27: ERD diagram World

The table Country consists of information about countries of the world, the table City consists of information about the cities in those countries and the class CountryLanguage about languages spoken in each country.
6.2.2  Goal state

With the goal state, the students get an idea about the functionality and interfaces of how the final enterprise application should look like. The task provides the following goal state:

Figure 28 shows the first interface of the World enterprise application.

![Input interface](image)

Retrieval information of the selected country.

Choose Country
- AFG-Afghanistan
- NLD-Netherlands
- ANT-Argentina
- ALB-Albania
- DZA-Algeria
- ASM-American Samoa
- AMD-andorra
- AGO-Angola
- AUS-Australia
- ATG-Antigua and Barbuda

Language ○ City

Retrieve

Figure 28: Input interface

Figure 29 illustrates the output if a user of the application chooses as country “Afghanistan” and selects “language”.

![Language interface](image)

The spoken languages are:
- Baloch
- Dari
- Pashto
- Turkmenian
- Uzbek

Figure 29: Language interface

Figure 30 shows the output if the user chooses as country “Afghanistan” and selects “city”.

![City interface](image)

The cities are:
- Kabul
- Qandahar
- Herat
- Mazar-e-Sharif

Figure 30: City interface
6.2.3 Solutions

In this section of the task the complete case study of how to build the World enterprise application is given. Appendix A presents this case study, which the students have to study and perform.

6.3 Task two: Pet Store

In this task, the students have to build a Pet store application. Figure 26 shows a screen-print of task two.

Figure 31: Task two Pet Store

This task is a completion task which means that it consists of a given and goal state and it includes partial solutions. The task starts with an introduction. The following introduction is given for task two:

The Java BluePrints program defines the application programming model for the Java EE platform. It provides best practice guidelines and architectural recommendations for real-world application scenarios to enable developers to build portable, scalable, and robust applications using the Java EE technologies.

Java BluePrint has developed a example application named Java Pet Store (https://blueprints.dev.java.net/petstore/)
6.3.1 Requirements

The following requirements are given for task two:

The Java BluePrint program gives us the assignment to extend the example enterprise application with a web system in which the users can edit the contactinfo of a seller. After each edit, a confirmation email is send to the seller. Therefore, a correct email address is necessary.

The architecture of the system has to be multi-tier; the clients use a web browser, the application is running on a glassfish application server and the database on a MySQL server.

The database is provided. The Java Blueprint program provide a petstore.sql file containing the create table and the insert statements and an ERD document containing the ERD diagram of the database.

A petstore.sql file and an ERD diagram are provided. Figure 27 shows the ERD diagram of the Pet Store database.

Figure 32: ERD Diagram Pet Store
6.3.2 Goal state

This task provides the following goal state:

*Figure 33 shows the interface, which the user sees.*

![Seller contact info]

*Figure 33: Seller contact info*

*Figure 34 illustrates the output if a user chooses as seller.*

![Edit screen of a seller]

*Figure 34: Edit screen of a seller*
6.3.3 Solutions

This task provides the following partial solutions:
First, build the edit requirement. Therefore, build the database layer, persistence layer, business layer and web layer.

Next, build the email requirement. Email is asynchronous communication, therefore a message driven bean is most suitable. For example, in the case the mail server is down. The source code of this bean is given in the MDB document.

The control Servlet is the producer of the messages. Look in the message producer document for information about how to send the messages.

Appendix B presents the MDB document and Appendix C the Message producer document.

6.4 Task three: Web shop

In this task, the students have to build a Web shop application. Figure 26 shows a screen-print of task three.
This task is a completion task which means that it consists of a given and goal state and it includes partial solutions. The task starts with an introduction. The following introduction is given for task three:

A computer shop wants to sell its articles on the internet. Selling online has a number of advantages over selling by conventional methods, including (http://www.businesslink.gov.uk):

- Making savings in set-up and operational costs. You do not need to pay shop assistants, rent high-street premises, or answer many pre-sales queries.
- Reducing order-processing costs - customer orders can automatically come straight into your orders database from the website.
- Reaching a global audience, thereby increasing sales opportunities.
- Competing with larger businesses by being able to open 24 hours a day, seven days a week. Being able to receive payment more quickly from online transactions.
- Attracting customers who would not normally have investigated your type of high-street outlet.
- Improving your offerings using the data gathered by tracking customer purchases.
- Using your online shop as a catalogue for existing customers.

6.4.1 Requirements

The following requirements are given for task two:

We get the assignment to build a small piece of the webshop. We have to build a register page in which the users of the webshop can register themselves with a name and password. We have to build the catalog page, so the users can see the articles, which they can buy, and the possibility to add articles to a shopping cart.

The architecture of the system has to be multi-tier; the clients use a web browser, the application is running on a glassfish application server and the database on a MySQL server.

The database is provided by a webshop.sql file containing the create table and the insert statements and an ERD document containing the ERD diagram of the database.

A webshop.sql file and an ERD diagram are provided. Figure 27 shows the ERD diagram of the Pet store database.
6.4.2 Goal state

This task provides the following goal state:

Figure 37 shows the register page.

After a customer has been registered, he can starts shopping:
After the user adds for example product 1, 5, and 6 and selects “Show current shopping cart” he gets:

![Shopping cart](image)

### 6.4.3 Solutions

This task provides the following partial solution:

We use two Servlets. The first Servlet, the Login Servlet starts an HTTP session and shows the register page to the user. After the user has been registered, the login Servlet passes the control to the ControlServlet. The source code of the login Servlet is given in the Login document.

For the cart bean, we have to make use of a stateful session bean because each user has its own shopping cart.

Finish, this application by yourself.

Appendix D presents the login document.
6.5 Task four: feedback

Figure 40 shows a screen-print of task four.

![Figure 40: Task four Feedback](image)

This task is a feedback task. The source codes of the tasks are provided. The students can learn from these source codes if they compare their own solutions with these solutions.
7 Development of tasks part two

In this section, we present the development of the tasks of assignment two.

7.1 Task one: Library enterprise application

In this task, the students have to build a Library enterprise application. Figure 41 shows a screen-print of task one.

![Figure 41: Task one Library enterprise application](image)

This task is a case study, therefore it consists of a given and goal state and it includes the complete solution. On the right hand site, we find the general information for this assignment. The task starts with an introduction, telling about the customer. The following introduction is given for task one:
A library is a collection of information, sources, resources, books, and services, and the structure in which it is housed: it is organized for use and maintained by a public body, an institution, or a private individual. In the more traditional sense, a library is a collection of books. The term can mean the collection, the building that houses such a collection, or both (http://en.wikipedia.org/wiki/Library).

7.1.1 Requirements

The next section in the task is the requirements, which are in fact the given state. The following requirements are given for task one:

A library needs an information GUI client that helps find out the information (title, language, year and subject belonging to a given book description id).

The library provides a books.sql file containing the create table and the insert statements and an ERD document containing the ERD diagram of the database. (The database tables are extracted from web.cecs.pdx.edu/~lmd/cs386/LibraryDB.sql)

The architecture of the system has to be multi-tier; the clients use a GUI application, the application is running on a glassfish application server and the database on a MySQL server.

An ERD diagram is also provided to give the student more insight in the database. Figure 27 shows the ERD diagram of the Library database.

Figure 42: ERD diagram Library
7.1.2 Goal state

Next interface illustrates the goal of this task:

![Interface](image)

In this example an id of 164 is given. When the user clicks on the button "Find all" the information of this book is shown.

7.1.3 Solutions

In the solutions part the complete case study of how to build the Library enterprise application is given. Appendix E presents this case study.
7.2 Task two: BaMaS

In this task, the students have to build a bachelor-master system. Figure 26 shows a screen-print of the Web page of task two.

![Screen-print of the Web page of task two](image)

The following introduction is given for task one:

**BaMaS** ([http://www.bamas.nl](http://www.bamas.nl)) is the online information system for the links and connection programmes between bachelor and master programmes in the Netherlands. After completion of a bachelor programme at a university, you can proceed to a related master programme. In the case, you want to move from a bachelor to a non-related master, a connection programme will often be required. It is also possible to follow up a bachelor programme at a polytechnic with a related master programme at another university or a university. For this task, a connection programme is required. For each bachelor programme you can check to which masters it is linked and for which masters a connection programme is required. You can also select a master programme and then check which bachelors are linked with this master, with or without a connection programme. Apart from information on links and connection programmes, BaMaS also contains limited information on the institutions and their educational programmes. Detailed information on educational programmes, links and connection programmes is available on the websites of the respective institutions. The information on educational programmes, links and connection programmes is accumulated and presented. For each bachelor programme you can check to which masters it is linked and for which masters a connection programme is required. You can also select a master programme and then check which bachelors are linked with this master, with or without a connection programme. Apart from information on links and connection programmes, BaMaS also contains limited information on the institutions and their educational programmes. Detailed information is provided through direct links to the information sites of the respective institutions. The information
on educational programmes, links and connection programmes.

In this task, we are going the build a small piece of the BaMaS system.

### 7.2.1 Requirements

The following requirements are given for task one:

We get the assignment to build a web system in which the users can retrieve all the institutions. Just the name and the street have to be shown to the users. Next, they can select an institution and all the names of the bachelor educations of the institutions are shown. From a bachelor education, the user can retrieve the names of the masters connecting to this bachelor education.

In addition, the customer needs a CRUD GUI client for the institutions. Just the name, street, phone and code fields have to be administered.

The architecture of the system has to be multi-tier; the clients use a web browser, the application is running on a glassfish application server and the database on a MySQL server.

The database is provided. The customer has a `bamas.sql` file containing the create table and the insert statements and an ERD document containing the ERD diagram of the database.

![Figure 44: ERD diagram BaMaS](image)

---

Instructional design for Java enterprise component technology 83
We focus on these three important tables of the BaMaS database, namely:

- Institutions; the different institutions who providing the educations
- Educations; the different bachelor and master’s programmes.
- Connectionbachelors; for connecting the bachelors with the master’s.

The primary and foreign keys of the tables are:

<table>
<thead>
<tr>
<th>Table</th>
<th>Primary Key</th>
<th>Foreign Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Institutions</td>
<td>INS_ID</td>
<td></td>
</tr>
<tr>
<td>Education</td>
<td>EDU_UD</td>
<td>INS_CODE to institutions (CODE)</td>
</tr>
<tr>
<td>connectionbachelors</td>
<td>CBA_ID</td>
<td>EDU_ID to educations (EDU_ID)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EDU_EDU_ID_CONNECTS_FROM to educations (EDU_ID)</td>
</tr>
</tbody>
</table>

7.2.2  Goal state and Solutions

This task is a conventional task, therefore no goal state and no solution is provided. Because of this it is this task suitable as exam task.
7.3 Task three: Feedback

This task is a feedback task. The source codes of the tasks are provided. The students can learn from these source codes if they compare their own solutions with these solutions.
Part III: Conclusions
8 Conclusions and recommendations

Section 8.1 presents our conclusions and section 8.2 introduces recommendations for further work.

8.1 Conclusions

We have chosen to apply the “Ten Steps to complex learning” method for performing the analysis and design phase of the development of the tasks for instruction. We have applied this method to the Java enterprise component technology. This method offers a solution for three persistent problems in the field of education, namely: compartmentalization, fragmentation, and the transfer paradox (See section 4.1). These problems are precisely the problems we notice in the available tutorials in the field of the Java enterprise component technology. The tutorials offer skills or knowledge (compartmentalization). In addition, the skills are isolated (fragmentation). In this specific domain and for small tasks the tutorials seem efficient, but they do not allow students to reach integrated objectives (transfer paradox). By applying “Ten Steps to complex learning” method we have developed learning tasks which provide integration instead of compartmentalization of the knowledge and skills. Each learning task is a whole task practice confronting the student with all the skills and knowledge. The task classes and scaffolding in the learning tasks provide coordination instead of fragmentation. This will results in transfer of learning. Therefore, our first conclusion is that the “Ten Steps to complex learning” method is a suitable method for developing tasks for instruction.

The research question of this thesis is: Can the tasks for instructions be built in such a way that they cover the process of development of component-based enterprise applications when studying the elements of the Java enterprise component technology (See Section 1.2)? Before we answer this question, we first look at the blueprint, which is the result of applying the “Ten Steps to complex learning” method. This blueprint consists of several learning tasks. In each learning task, the whole process of developing a component-based enterprise application is addressed (See subsection 5.1.2). The different requirements in each learning task allow the students to study all of the EJB components and the several aspects of the EJB component model. Utilizing the EJB components and using the EJB component model is one of the difficulties of applying the Java enterprise component technology (See section 3.4). The several learning task are grouped into task classes. Each task class addresses a different client (See section 5.2). This gives the students the possibility to practice the impact of the several clients on the EJB components. This is another difficulty of applying the Java enterprise component technology (See section 3.4). The supportive information in the blueprint provides the students the necessary information about the EJB component, EJB component model and the middleware services. For other skills necessary to
build component-based enterprise application only procedural information is given. This means that students are performing skills necessary to build the component-based enterprise application, but the focus stays on the EJB technology.

Therefore, our second conclusion is that with this blueprint tasks for instructions can be developed in such way that the whole process of development of component-based applications is addressed and the separate elements of the Java enterprise component technology can be studied.

8.2 Recommendations for future work

In this thesis, we have developed task class one “Developing web-based enterprise applications” and task class two “Developing web-based enterprise applications with application client”. Task class three “Developing web-based enterprise applications with application and web service client” has not been developed during this research assignment. However, Web service clients play a significant role nowadays, especially within the currently popular Service Oriented Architecture (SOA) (Brown, Johnston, & Kelly, 2002). We recommend implementing this task class. In addition, we could think about other task classes such as Microsoft .NET or CORBA clients. The development of the third and other task classes is put forward for future work.

We have not performed the last two steps of the ADDIE model (See section 1.4), namely implementation and evaluation. Implementation is the process of integrating the tasks in a real context. The course “Component Based Development” of the Open Universiteit Nederland offers possibilities to try the tasks. For this purpose, we need to find a few students who are willing to complete the tasks for instructions. We need to specify on which conditions this will take place. There is also a possibility to implement the tasks as an open course at the Open Universiteit Nederland. We also need to evaluate the instructions. For this purpose, we can use the last task at the end of task class two.
References


http://www.ou.nl/eCache/DEF/36.html


http://repository.tudelft.nl/consumption/idcplg?IdcService=GET_FILE&RevisionSelectionMethod=latestReleased&dDocName=175130


http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/


http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#whatis


## Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>4C/ID</td>
<td>Four-Component Instructional Design</td>
</tr>
<tr>
<td>ADDIE</td>
<td>Analysis  Design Development Implementation Evaluation</td>
</tr>
<tr>
<td>AWT</td>
<td>Abstract Window Toolkit</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>BMT</td>
<td>Bean Managed Transaction</td>
</tr>
<tr>
<td>CELSTEC</td>
<td>Centre for Learning Science and Technologies</td>
</tr>
<tr>
<td>CBD</td>
<td>Component Based Development</td>
</tr>
<tr>
<td>COP</td>
<td>Component-Oriented Programming</td>
</tr>
<tr>
<td>CMT</td>
<td>Container Managed Transaction</td>
</tr>
<tr>
<td>CRUD</td>
<td>Create  Retrieve  Update and Delete</td>
</tr>
<tr>
<td>DI</td>
<td>Dependency Injection</td>
</tr>
<tr>
<td>DNS</td>
<td>Domain Name System</td>
</tr>
<tr>
<td>EAR</td>
<td>Enterprise ARchive</td>
</tr>
<tr>
<td>EJB</td>
<td>Enterprise JavaBean</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HTTP</td>
<td>HyperText Transfer Protocol</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>IIOP</td>
<td>Internet Inter-ORB Protocol</td>
</tr>
<tr>
<td>ISD</td>
<td>Instructional Systems Design</td>
</tr>
<tr>
<td>JAAS</td>
<td>Java Authentication and Authorization Service</td>
</tr>
<tr>
<td>JAR</td>
<td>Java Archive</td>
</tr>
<tr>
<td>Java EE</td>
<td>Java Enterprise Edition platform</td>
</tr>
<tr>
<td>Java SE</td>
<td>Java Standard Edition platform</td>
</tr>
<tr>
<td>JSP</td>
<td>Java Server Page</td>
</tr>
<tr>
<td>JSF</td>
<td>Java Server Face</td>
</tr>
<tr>
<td>JMS</td>
<td>Java Message Service</td>
</tr>
<tr>
<td>JNDI</td>
<td>Java Naming and Directory Interface</td>
</tr>
<tr>
<td>JPA</td>
<td>Java Persistence API</td>
</tr>
<tr>
<td>JPQL</td>
<td>Java Persistence Query Language</td>
</tr>
<tr>
<td>JTA</td>
<td>Java Transaction API</td>
</tr>
<tr>
<td>JVM</td>
<td>Java Virtual Machine</td>
</tr>
<tr>
<td>LDAP</td>
<td>Lightweight Directory Access Protocol</td>
</tr>
<tr>
<td>MOM</td>
<td>Message-Oriented Middleware</td>
</tr>
<tr>
<td>NELLL</td>
<td>Netherland Laboratory for Lifelong learning</td>
</tr>
<tr>
<td>OMG</td>
<td>Object Management Group</td>
</tr>
<tr>
<td>ORM</td>
<td>Object Relational Mapping</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modelling Language</td>
</tr>
<tr>
<td>OUNL</td>
<td>Open Universiteit Nederland</td>
</tr>
<tr>
<td>POJO</td>
<td>Plain Old Java Object</td>
</tr>
<tr>
<td>POJI</td>
<td>Plain Old Java Interfaces</td>
</tr>
<tr>
<td>RMI</td>
<td>Remote Method Invocation</td>
</tr>
<tr>
<td>SOA</td>
<td>Service Oriented Architecture</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium</td>
</tr>
<tr>
<td>WAR</td>
<td>Web Archive</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web services Description language</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
</tbody>
</table>
Appendix A  Case study World enterprise application

A.1  Modelling

A.1.1  Use Case diagram

Statistics Finland needs an information web system that helps find out which languages are spoken in a given country or which cities are situated in a given country. We translate these requirements into the following use-case diagram:

![Use-case diagram](image)

*Figure 46: Use-case diagram*

A.1.2  Design

We are going to design a Java enterprise application. A Java enterprise application consists of a client, web, business, persistence and data layer. We design a web-based enterprise application. This means the client layer consists of a web browser. The data layer consists of the given MySQL database. An Integrated Development Environment (IDE) can easily generate the persistence layer because the database has been given. We need to design the Web layer and Business layer. In order to retrieve the languages and cities three interfaces are needed; one interface for the input (input.jsp) or more specific for selecting the country, one interface for showing the languages (language.jsp) and one interface for showing the cities (cities.jsp). We apply the MVC pattern, this means we use a Java Server Page for each interface and a Servlet (ControlServlet) to control these pages.
The MVC pattern separates display code from flow control logic. Typically, MVC design pattern combines the use of both Servlets (controller) and JSPs (view). It takes advantage of the strengths of both technologies, using JSP to generate the presentation layer and Servlets to perform process-intensive tasks. Separating responsibilities among model, view, and controller objects reduces code duplication and makes enterprise applications easier to maintain. Figure 47 illustrates the MVC pattern.

![Figure 47: Model View Controller](image)

The controller is focused on receiving requests from the client (1), deciding what business logic has to be performed (2), and then delegate the responsibility for producing the next phase of the user interface to an appropriate view component (3). The view is used to create the user interface based on the data read from the model (4).

Next, we specify the Enterprise JavaBeans. We use session beans for the interaction with the Servlet. The enterprise application does not need to keep the state; therefore, a stateless session bean is sufficient. We apply the session bean façade pattern. In this pattern, a stateless session bean is used as a façade to hide access to the domain model of the Java enterprise application. Clients, in our case a Servlet, interact only with the session bean and never directly with the entities. Figure 48 demonstrates this pattern.

![Figure 48: Session bean façade](image)
The session bean provides one access point for the clients and hides the complexity of the domain model. With this pattern, the session bean acts as a façade for the domain model. This means that the Servlet only interacts with the session bean and never directly with the entities. We define a local interface for the session bean, because the Servlet and JSPs are running on the same Java application server.

Now, we are able to complete the component diagram for the World enterprise application. Figure 49 shows the component diagram.

![Component Diagram](image-url)

*Figure 49: Component diagram*
The entity classes are not components, entities are POJOs (Plain Old Java Objects). Figure 50 illustrates the communication between the several components for showing the input interface to the user in a sequence diagram.

![Sequence diagram](image)

**Figure 50: Sequence diagram**

### A.1.3 Architecture

The architecture of the system has to be multi-tier; the clients use a web browser, the application is running on a Glassfish application server and the database on a MySQL server. Figure 51 presents the deployment diagram:

![Deployment diagram](image)

**Figure 51: Deployment diagram**
Next sections present the implementation of the World enterprise application. We start with implementing the database layer.
A.2 Database Layer

A.2.1 Downloading & installing MySQL

We use a MySQL database as the database for the World enterprise application. Download MySQL Community Server from http://www.mysql.com the site of MySQL and install MySQL. For a new installation, we can select all the defaults. For the root password, we choose admin.

If you have a previous MySQL installation, you have to choose your old root password. If you had a previous installation and you forgot your password, you will encounter problems when you try to install. In that case, you have to reset the password before installing (http://dev.mysql.com/doc/refman/5.0/en/resetting-permissions.html).

A.2.2 Building the database

Start the MySQL Command Line Client. To build the database and load the World tables, we use the following commands.

Creating the database World:
create database world;

Using the database:
use world;

Creating the tables and inserting the rows by loading the database script World.sql:
source <path_name>\world.sql

We can use the following command to check if the tables are actually loaded:
show tables;

We get the following result:

+-----------------+
| Tables_in_world |
+-----------------+
| city            |
| country         |
| countrylanguage |
+-----------------+

The database layer of the Java Enterprise Application has been built. Now we are ready to build the Java enterprise application. We use NetBeans as Integrated Development Environment (IDE).
A.3 NetBeans

A.3.1 Installing NetBeans and JDK 6

Before we can install NetBeans, we first need to install the Java SE Development Kit (JDK). Choose JDK 1.6 on http://java.sun.com/javase/downloads/index.jsp. Next, we install NetBeans. In this case study version 6.7.1 of NetBeans is used. Download NetBeans IDE 6.7.1 from http://www.netbeans.org. Choose for the download bundle “Java”. Now we are ready to install NetBeans. During the installation, accept all the default options.

A.3.2 Define project

Before we can build the World enterprise application, we first need to define a project. To define an enterprise application project we perform the following steps:

2. Name the application EnterpriseApplicationWorld and specify a location for the project. Click Next.
3. Choose Java EE 5 version. Click Finish.

Two modules have been generated: an EJB module and a Web application module.
A.4 Persistence Layer

A.4.1 Database Connection

To build the persistence layer, we first need to make a connection with the database. Navigate to the tab Services. Choose database and right-click, we choose new connection. A form is presented, Figure 52 shows how to complete this form (the password for root is ADMIN, port 3306 is the default port):

Click OK, the connection will be established.

A.4.2 Persistence Unit

A Persistence Unit (PU) is required if we want to use Java Persistence API (JPA). JPA is a Java programming language framework that allows developers to manage relational data. Navigate to tab Projects and right-click on the project “EnterpriseApplicationWorld-ejb”, then perform the following steps:

1. Choose New > Other > Persistence > Persistence Unit. Click Next.
2. Choose Toplink (default). For the field data source, choose New data source. Figure 53 illustrates the screen we get. Give the JNDI14 Name the value jdbc/world and as database connection choose the world connection.

---

14 The Java Naming and Directory Interface (JNDI) is an API for directory service that allows clients to discover and lookup data and objects through a name.
Choose for Table Generation Strategy **None**. Click **Finish**.

The Persistence Unit has been defined in the persistence.xml file. The data source description is added to sun-resources.xml under the Server Resources node.

### A.4.3 Creating the Entity Classes

To create the entity classes right-click on the project “EnterpriseApplicationWorld-ejb”

1. Choose **New > Entity Classes from Database**.
2. Choose **jdbc/world** as Data Source.
3. Choose **add all** for creating entities for all the database tables. Click **Next**.
4. Give the package the name “**world**”. Click **Finish**.

There are four entity classes generated while we only have three tables. An extra entity class is generated because of the double primary key column of the table Countrylanguage. All of this is handled by the Toplink framework. The complete persistence layer is now generated.

Table 9 shows the source code of the City entity. The important parts of the code are explained in the right column:
**Table 9: Source code entity**

```java
package world;

import java.io.Serializable;
import java.persistence.Basic;
import java.persistence.Column;
import java.persistence.Entity;
import java.persistence.GeneratedValue;
import java.persistence.GenerationType;
import java.persistence.Id;
import java.persistence.NamedQueries;
import java.persistence.NamedQuery;
import java.persistence.Table;

@Entity
@Table(name = "city")
@NamedQueries({
    @NamedQuery(name = "City.findAll", query = "SELECT c FROM City c"),
    @NamedQuery(name = "City.findById", query = "SELECT c FROM City c WHERE c.id = :id"),
    @NamedQuery(name = "City.findByName", query = "SELECT c FROM City c WHERE c.name = :name"),
    @NamedQuery(name = "City.findByCountryCode", query = "SELECT c FROM City c WHERE c.countryCode = :countryCode"),
    @NamedQuery(name = "City.findByDistrict", query = "SELECT c FROM City c WHERE c.district = :district"),
    @NamedQuery(name = "City.findByPopulation", query = "SELECT c FROM City c WHERE c.population = :population")
})
public class City implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(name = "ID")
    private Integer id;

    @Basic(optional = false)
    @Column(name = "Name")
    private String name;

    @Basic(optional = false)
    @Column(name = "CountryCode")
    private String countryCode;

    @Basic(optional = false)
    @Column(name = "District")
    private String district;

    @Basic
    @Column(name = "Population")
    private int population;

    public City() {
    }

    public City(Integer id) {
        this.id = id;
    }

    public City(Integer id, String name, String countryCode, String district, int population) {
        this.id = id;
        this.name = name;
        this.countryCode = countryCode;
        this.district = district;
        this.population = population;
    }

    public Integer getId() {
        return id;
    }
}
```

**Table:**
- `@Entity` marks POJO as entity
- `@Table` specifies table mapping
- `@Id` Entity ID
- `@Column` column mappings, which names of the entity maps with the database table column names
- Get and set methods for the attributes
The persistence layer has been finished. Next section presents the implementation of the business layer.
**A.5 Business layer**

**A.5.1 Creating the Session Bean Façade**

We are now going to build a Session Bean façade. Right-click on the project “EnterpriseApplicationWorld-ejb”:

1. Choose **New > Other > Persistence > Session bean for Entity classes. Click Next.**
2. Choose **Add all** to select all the Entity Classes. Click **Next**.
3. Click **Finish**.

The session bean façades have been built and local interfaces have been defined. Local interface are used by clients that are running on the same Java EE server. Clients who are not running on the same Java EE server have to use the remote interface. In that case, Remote Method Invocation (RMI) is used for the communication. RMI has a negative effect on the performance. So if possible, use the local interface. The CountryFacade consists of the following code:

**Table 10: Source code session bean**

```java
package world;

import java.util.List;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

@Stateless
public class CountryFacade implements CountryFacadeLocal {
    @PersistenceContext
    private EntityManager em;

    public void create(Country country) {
        em.persist(country);
    }

    public void edit(Country country) {
        em.merge(country);
    }

    public void remove(Country country) {
        em.remove(em.merge(country));
    }

    public Country find(Object id) {
        return em.find(Country.class, id);
    }

    public List<Country> findAll() {
        return em.createQuery("select o from Country as o").getResultList();
    }
}
```

@Stateless: indicates it is a stateless session bean

@PersistenceContext: injects a instance of the Entity Manager

CRUD methods
A Session bean façade implements the entity manager. The key methods of this Entity Manager API are:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>persist(Object entity)</td>
<td>Insert the state of an entity into the database.</td>
</tr>
<tr>
<td>remove(Object entity)</td>
<td>Delete the entity state from the database.</td>
</tr>
<tr>
<td>merge(Object entity)</td>
<td>Synchronize the state of detached entity with the persistency context.</td>
</tr>
<tr>
<td>refresh(Object entity)</td>
<td>Reload the entity state from the database.</td>
</tr>
<tr>
<td>find(Class entityclass, Object pk)</td>
<td>Execute a simple Primary Key query.</td>
</tr>
<tr>
<td>createQuery(String query)</td>
<td>Create query instance using dynamic JPQL.</td>
</tr>
<tr>
<td>createNamedQuery(String queryname)</td>
<td>Create instance for a predefined query.</td>
</tr>
<tr>
<td>createNativeQuery(String query, Class resultclass)</td>
<td>Create instance for an SQL query.</td>
</tr>
<tr>
<td>flush()</td>
<td>Force synchronization of persistency context to database.</td>
</tr>
</tbody>
</table>

Within the persistence context of the Entity Manager, the entity instances and their lifecycle are managed. The persistence context consists of “managed” entities:

![Persistence context diagram](Figure 54: Persistence context)

Entities exist in one of four states: new, managed, detached, and removed. Figure 55 shows the entity life cycle.
- **New**: an entity is new if it has just been instantiated using the new operator, and it is not associated with a persistence context. It has no persistent representation in the database and no identifier value has been assigned.

- **Managed (persistent)**: a managed entity instance is an instance with a persistent identity that is currently associated with a persistence context.

- **Detached**: the entity instance is an instance with a persistent identity that is no longer associated with a persistence context, usually because the persistence context was closed or the instance was evicted from the context.

- **Removed**: a removed entity instance is an instance with a persistent identity, associated with a persistence context, but scheduled for removal from the database.

When an entity is first instantiated using Java new operator, its state is "new". When we invoke EntityManager.persist() on a "new", or "removed" Entity, it becomes "managed". When we invoke EntityManager.merge() on a "detached" Entity, a "managed" copy of the Entity is created. When we invoke EntityManager.remove() on a "managed" Entity, its new state becomes "removed".
We need to add the following method to CityFacade session bean:

```java
// Retrieve the cities of a country
public List<City> findCity(String code) {
    Query query = em.createQuery("SELECT object (a) FROM City a WHERE a.countryCode.code = :code");
    query.setParameter("code", code);
    return query.getResultList();
}
```

In addition, we need to import javax.persistence.Query.

The CityFacadeLocal interface needs to be changed to make the methods of the session available. The interface needs to look like:

```java
package world;

import java.util.List;
import javax.ejb.Local;

@Local
public interface CityFacadeLocal {
    void create(City city);
    void edit(City city);
    void remove(City city);
    City find(Object id);
    List<City> findAll();
    public List<City> findCity(String code);
}
```

We also need to add the following method to CountryLanguageFacade session bean:

```java
// Retrieve the spoken languages of a country
public List<Countrylanguage> findLanguage(String code) {
    Query query = em.createQuery("SELECT object(a) FROM Countrylanguage a WHERE a.countrylanguagePK.countryCode = :code");
    query.setParameter("code", code);
    return query.getResultList();
}
```

In addition, we need to import javax.persistence.Query.

Change also the interface of this session bean!
A.6 Web Layer

A.6.1 Creating the ControlServlet

The communication between the Servlets and JSP will be as follow:

First, we build the control Servlet. Right-click “EnterpriseApplicationWorld-war”:
1. Choose new > Servlet.
2. Give the Class Name the name ControlServlet and package web. Click Next.
3. Click Finish.
4. Navigate to the Servlet code and right-click somewhere in the code. In the menu, choose Insert code > Call Enterprise Bean > CountryFacade.
5. Referenced Interface: Local.

The following lines are added to the Servlet code:

```java
@EJB
private CountryFacadeLocal countryFacade;
```

Do the same for CountrylanguageFacade and CityFacade.

Now, the connection between the web layer and business layer is established.
Next, we replace the method `processRequest` of the Servlet with the following lines of code:

```java
protected void processRequest(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {
    String page = null;
    String country = request.getParameter("country");
    if (country == null) {
        // make a attribute to pass all the countries to the input.jsp
        request.setAttribute("countries", this.countryFacade.findAll());
        // forward the request to input.jsp
        page = "/input.jsp";
    } else {
        String information = request.getParameter("information");
        if ("language".equals(information)) {
            request.setAttribute("language",
                this.countrylanguageFacade.findLanguage(country));
            // forward the request to language.jsp
            page = "/language.jsp";
        } else if ("city".equals(information)) {
            request.setAttribute("city", this.cityFacade.findCity(country));
            // forward the request to city.jsp
            page = "/city.jsp";
        }
    }
    RequestDispatcher dispatcher =
        getServletContext().getRequestDispatcher(page);
    dispatcher.forward(request, response);
}
```

**Fix the necessary imports!!**

**A.6.2 Creating the JSPs**

Right-click “EnterpriseApplicationWorld-war”:

1. Choose **new > JSP**.
2. Give the JSP File Name the name **input**. Click **Finish**.

With a Java Server Page, we can easily integrate Java code into the HTML pages. Web designers can focus on the design. We can make use of the following JSP constructs:

<table>
<thead>
<tr>
<th>Table 12: JSP constructs</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;%! declarations %&gt;</code></td>
</tr>
<tr>
<td><code>&lt;%= expression %&gt;</code></td>
</tr>
<tr>
<td><code>&lt;% code fragment %&gt;</code></td>
</tr>
</tbody>
</table>
Replace the code with the following lines of code:

```html
<%@ page contentType="text/html" pageEncoding="UTF-8" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>JSP Page</title>
</head>
<body>
<h2>Retrieve information of the selected country. </h2>
<form action="/ControlServlet">
<b>Please, first select a Country </b>
<
<select name="country" size="10">
<% java.util.List country = (java.util.List)
request.getAttribute("countries");
for (java.util.Iterator it = country.iterator(); it.hasNext();) {
world.Country elem = (world.Country) it.next();
%d>
<option value="<%= elem.getCode() %>" > <%= elem.getName() %>
</option>
<%}%>
</select>
<br>
<br>
<input type="radio" name="information" value="language" CHECKED>Language
<input type="radio" name="information" value="city"> City <br>
<br>
<input type="submit" value="Retrieve" />
</form>
</body>
</html>

A.6.3 JSP Language

Build also a JSP with the name language and the following lines of code:

```html
<%@ page contentType="text/html" pageEncoding="UTF-8" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>JSP Page</title>
</head>
<body>
<h2>The spoken languages are:</h2>
<% java.util.List language = (java.util.List)
request.getAttribute("language");
if (language != null) {
for (java.util.Iterator it = language.iterator(); it.hasNext();) {
world.Countrylanguage elem = (world.Countrylanguage) it.next();
<%>
<br>
<%= elem.getCountrylanguagePK().getLanguage() %>
<%}%>
</body>
</html>
A.6.4 JSP City

Build also a JSP with the name city and the following lines of code:

```jsp
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>JSP Page</title>
</head>
<body>
<h2>The cities are:</h2>
<%
 java.util.List city = (java.util.List) request.getAttribute("city");
 if (city != null) {
 for (java.util.Iterator it = city.iterator(); it.hasNext();) {
 world.City elem = (world.City) it.next();
 %>
 <br>
<%= elem.getName() %>
%
 %}%
</body>
</html>
```
A.7 Run the application

Right-click on EnterpriseApplicationWorld:

1. Choose properties > run
2. Change the Relative URL property to ControlServlet

To run the enterprise application we perform the following steps:

1. Right-click the “EnterpriseApplicationWorld” project node and select Clean and Build to build the project.
2. Right-click the “EnterpriseApplicationWorld” project node and select Deploy to deploy the application to the application server (Glassfish).
3. Finally, right-click the “EnterpriseApplicationWorld” project node and this time select Run. This will open up the web browser and point to the ControlServlet.

If we encounter problems, we can try to restart the application server: tab services > servers > glassfish V2 > restart. Alternatively, we can try to undeploy the enterprise application:

For specific problems, we can choose View Server log.
Appendix B  Message-driven bean

Keep the name of the bean default which is NewMessage. Before we can define a message-driven bean, we need to add a destination. Choose for project destination and then add a new destination and choose the following:

Destination name = message
Destination type = queue

The source code of the MDB is:

```java
package petstore;
import java.util.Properties;
import javax.ejb.ActivationConfigProperty;
import javax.ejb.MessageDriven;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.ObjectMessage;
import javax.mail.Session;
import javax.mail.Message.RecipientType;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;

@MessageDriven(mappedName = "jms/message",
                activationConfig = {
                    @ActivationConfigProperty(propertyName = "acknowledgeMode",
                                                propertyValue = "Auto-acknowledge"),
                    @ActivationConfigProperty(propertyName = "destinationType",
                                                propertyValue = "javax.jms.Queue")
                })
public class NewMessageBean implements MessageListener {
    public NewMessageBean() {
    }

    public void onMessage(Message message) {
        // extract the email object and call the send method
        ObjectMessage objectMessage = (ObjectMessage) message;
        try {
            Sellercontactinfo seller = (Sellercontactinfo) objectMessage.getObject();
            sendEmail(seller);
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```
public void sendEmail(SellerContactInfo seller) {
    // attempt to send the email
    try {
        // put your own smtp server in here
        String host = "smtp.xxxxx.nl";
        String from = "noreply@ou.nl";
        String to = seller.getEmail();
        String emailSubject = "update";
        String emailText = seller.getFirstName() + "\n" +
        seller.getLastName() + "\n" + seller.getEmail();

        // Get system properties
        Properties props = System.getProperties();
        // Setup mail server
        props.put("mail.smtp.host", host);
        // Get session
        Session session = Session.getDefaultInstance(props, null);
        // Define message
        MimeMessage message = new MimeMessage(session);
        message.setFrom(new InternetAddress(from));
        message.addRecipient(RecipientType.TO, new InternetAddress(to));
        message.setSubject(emailSubject);
        message.setText(emailText);
        // Send message
        Transport.send(message);
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}

Before we can actually run this application we need to change a parameter. In our situation, the
JMS service is handled by localhost. Therefore, we have to perform the following:

1. Open the admin console of the Glassfish server (user = admin, default password
   = adminadmin)

2. Select: Configuration > Java Message Service node in the tree on the left side.
   Change the type from "Embedded" into "Local". The documentation says:
   "Choose LOCAL (the default for the server-config configuration) to access the JMS
   service on the local host. The JMS service is started and managed by the Application
   Server."

3. Restart Glassfish
Appendix C  Message producer

In the Servlet right-click:

Insert code > send JMS message > Message-driven bean
Click ok.

Two methods are added to the Servlet. The first method has to be changed, look at the code below.
The second method stays unchanged.

```java
private Message createJMSMessageForJmsMessage(Session session, Object messageData)
throws JMSException {
    Sellercontactinfo seller = (Sellercontactinfo) messageData;
    // here we create an object message, that will be sent in JMS message
    ObjectMessage message1 = session.createObjectMessage();
    message1.setObject(seller);
    return message1;
}

private void sendJMSMessageToMessage(Object messageData) throws JMSException {
    Connection connection = null;
    Session session = null;
    try {
        connection = messageFactory.createConnection();
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        MessageProducer messageProducer = session.createProducer(message);
        messageProducer.send(createJMSMessageForJmsMessage(session, messageData));
    } finally {
        if (session != null) {
            try {
                session.close();
            } catch (JMSException e) {
                Logger.getLogger(this.getClass().getName()).log(Level.WARNING, "Cannot close session", e);
            }
        }
        if (connection != null) {
            connection.close();
        }
    }
}
```

In our existing code we need to invoke sendJMSMessageToMessage
Appendix D  Login Servlet

The source code of the method processRequest of the login Servlet is:

```java
protected void processRequest(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    String name = request.getParameter("name");
    String password = request.getParameter("password");
    /**
     * Since a servlet is pooled and re-used for different user requests, the
     * servlet code does not store any information specific to
     * any user. Rather, we store a reference to the user's
     * Cart in the user's HttpSession object, which is
     * globally accessible to all servlets.
     */
    HttpSession session = request.getSession(true);
    if(session.isNew())
    {
        RequestDispatcher dispatcher = getServletContext().getRequestDispatcher("/user.jsp");
        dispatcher.forward(request, response);
    }
    else {
        Customers customerNew = new Customers();
        customerNew.setName(name);
        customerNew.setPassword(password);
        customersFacade.create(customerNew);
        session.setAttribute("cart", cartBean);
        ((CartBeanLocal) session.getAttribute("cart")).clear();
        RequestDispatcher dispatcher = getServletContext().getRequestDispatcher("/ControlServlet");
        dispatcher.forward(request, response);
    }
}
```

In the other Servlet and JSP we have to use the following line of code to put these components to the same HttpSession:

```java
HttpSession session = request.getSession(false);
```
Appendix E  Case study Library enterprise application

E.1  Database Layer

E.1.1  Building the database

We use a MySQL database as the database for the enterprise application. Start the MySQL Command Line Client. To build the database and load the book tables, we use the following commands:

Creating the database books:
create database books;

Using the database:
use books;

Creating the tables and inserting the rows by loading the database script librarydb.sql:
source <path_name>\books.sql

We can use the following command to check if the tables are actually loaded:
show tables;

We get the following result:

<table>
<thead>
<tr>
<th>Tables_in_books</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
</tr>
<tr>
<td>author</td>
</tr>
<tr>
<td>book</td>
</tr>
<tr>
<td>book_description</td>
</tr>
<tr>
<td>borrowrel</td>
</tr>
<tr>
<td>person</td>
</tr>
<tr>
<td>publisher</td>
</tr>
<tr>
<td>relauth</td>
</tr>
<tr>
<td>relpub</td>
</tr>
<tr>
<td>subject</td>
</tr>
</tbody>
</table>

The database layer of the Java Enterprise Application has been built. Now we are ready to build the enterprise application. We use NetBeans as Integrated Development Environment (IDE).
E.2 NetBeans

E.2.1 Define project

Before we can build the enterprise application, we first need to define a project. To define an enterprise application project we perform the following steps:

Creating the Enterprise Application Project:

1. Choose **File > New Project > Java EE > Enterprise Application**. Click **Next**.
2. Name the application **EnterpriseApplicationBooks** and specify a location for the project. Click **Next**.
3. Choose **Java EE 5** version.
4. Unselect **create Web application module** and select **create application client module**.
5. Click **Finish**.

Two modules have been generated: an EJB module and an application client module.
E.3 Persistence Layer

E.3.1 Database Connection

To build the persistence layer, we first need to make a connection with the database. Navigate to the tab Services. Choose database and right-click, we choose new connection. A form is presented, Figure 57 shows how to complete this form (the password for root is admin, port 3306 is the default port):

![Figure 57: Define the database connection](image)

Click Ok, the connection will be established.

E.3.2 Persistence Unit

A Persistence Unit (PU) is required if we want to use Java Persistence API (JPA). JPA is a Java programming language framework that allows developers to manage relational data. Navigate to tab Projects and right-click on the project “EnterpriseApplicationBooks-ejb”, and then perform the following steps:

3. Choose New > Other > Persistence > Persistence Unit. Click Next.
4. Choose Toplink (default). If we select data source, choose New data source. Select the books database as our data source and give it the JNDI name jdbc/books
5. Choose for Table Generation Strategy None. Click Finish.

The Persistence Unit is defined in the persistence.xml file. The data source description is added to sun-resources.xml under the Server Resources node.
8.3 Creating the Entity Classes

To create the entity classes right-click on the project “EnterpriseApplicationBooks-ejb”.

1. Choose New > Entity Classes from Database.
2. Choose jdbc/books as Data Source.
3. Choose Add All. Click Next.
4. Give the package the name books. Click Finish.

The complete persistence layer is generated.
E.4 Business layer

E.4.1 Creating the Session Bean Façade

We are now going to build a Session Bean façade. Right-click on the project “EnterpriseApplicationBooks-ejb”.

1. Choose New > Other > Persistence > Session bean for Entity classes. Click Next.
2. Choose books.BookDescription as Selected Entity Classes, books.Subject is also included because of the “include referenced classes”. Click Next.
4. Click Finish.

A remote facade for BookDescription and Subject are generated.
E.5 Client layer

E.5.1 Application client

First, we make a JFrame:

1. Choose `EnterpriseApplicationBooks-app-client` > `new` -> `JFrame form`
2. Give the JFrame the name `Books JFrame` and choose as Package `enterpriseapplicationbook`:
3. Click `Finish`.
4. On the project tree find `main.java class`. Right click and delete it. Click `OK`.

E.5.2 Design JFrame

In the design mode of the JFrame we can drag and drop components of the Palette on the JFrame. We put a label and textfield on the form for the columns we need. Pick in the Palette for "label" and put it on the frame. In the Properties of the label, we can change the text of the label. Do the same for textfield and remove the text of this field. We also add a button for the action. We get the following:

![Design JFrame](image)

In the properties palette under the button `code` we find the parameter `Variable name`. Here, we can changes the names of the variables to:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>jTextFieldId</td>
</tr>
<tr>
<td>Title</td>
<td>jTextFieldTitle</td>
</tr>
<tr>
<td>Language</td>
<td>jTextFieldLanguage</td>
</tr>
<tr>
<td>Year</td>
<td>jTextFieldYear</td>
</tr>
<tr>
<td>Subject</td>
<td>jTextFieldSubject</td>
</tr>
</tbody>
</table>

For the button, we have to add an event "actionPerformed"
E.5.3 Source JFrame

Navigate to the source of the JFrame and make a connection with the façade:

Insert code > call enterprise bean > BookDescriptionFacadeRemote > remote

The following is added to our code:

```java
@EJB
private static BookDescriptionFacadeRemote bookDescriptionFacade;
```

Next, put the following code to the button handler:

```java
private void JButtonFindActionPerformed(java.awt.event.ActionEvent evt) {
    if (!(jTextFieldId.getText()).equals("")) {
        try {
            // To clear up the error status label
            JLabelMessage.setText(null);
            Integer bookNumber = Integer.valueOf(jTextFieldId.getText());
            BookDescription description =
                    bookDescriptionFacade.find(bookNumber);
            jTextFieldTitle.setText(description.getTitle());
            jTextFieldLanguage.setText(description.getLanguage());
            jTextFieldYear.setText(Integer.toString(description.getYear()));
            jTextFieldSubject.setText(((Subject)description.getSubjectId()).getSubjectType());
        } catch (NumberFormatException ef) {
            JLabelMessage.setText("Wrong id format");
        } catch (NullPointerException en) {
            JLabelMessage.setText("No such id exists");
        }
    }
}
```

E.6 Run the application

Right-click on EnterpriseApplicationBooks-app-client:

1. choose properties > run
2. Change the Main class property to enterpriseapplicationbooks.BooksJFrame

To run the enterprise application we perform the following steps:

1. Right-click the “EnterpriseApplicationWorld” project node and select Clean and Build to build the project.
2. Right-click the “EnterpriseApplicationWorld” project node and select Deploy to deploy the application to the application server (Glassfish).

Finally, right-click the “EnterpriseApplicationWorld” project node and this time select Run.