Master Thesis

Enterprise JavaBeans

Evaluation of different EJB platforms

carried out at the

Institute of Software Technology

and Interactive Systems

Vienna University of Technology

under the guidance of

ao. Univ. Prof. Dr. Dipl.-Ing. Gerald Futschek

by

LY The Minh

Ungargasse 41

2700 Wiener Neustadt

Vienna, January 2005

Abstract

Enterprise JavaBeans (EJB) is a promising new component architecture that is based on the popular programming language Java. In these fast living times shorter development iterations, the internet and electronic commerce become more and more important aspects of the software business. As a result a new flexible and powerful software concept is needed. Therefore this thesis wants to give an overview of the concepts of EJB and alternative component models like XPCOM, CCM or COM+. The evaluation and comparison of three commercial EJB platforms (BeanTA, PowerTier and WebLogic) should illustrate the concept's advantages and disadvantages.

Kurzfassung

Enterprise JavaBeans (EJB) ist eine viel versprechende Komponentenarchitektur, die auf der beliebten Programmiersprache Java aufbaut. In dynamischen Zeiten wie diesen, in denen kürzere Entwicklungszeiten, das Internet und der elektronische Handel eine immer größer werdende Rolle spielen, ist der Bedarf nach einem vielseitigen und mächtigen Softwarekonzept eine natürliche Konsequenz. Deshalb möchte diese Arbeit einen Überblick über das Komponentenkonzept von EJB und auch alternativen Komponentenmodelle wie XPCOM, CCM oder COM+ geben. Durch eine Evaluierung und einen Vergleich von drei kommerziellen EJB Plattformen (BeanTA, PowerTier und Weblogic) sollen die Stärken und Schwächen des Konzeptes dargestellt und veranschaulicht werden.

Table of Contents

List of Listings

List of Figures

Acknowledgements

It takes more than one single person to write a thesis, because writing a thesis requires more than simply typing words and sentences. I would not have been able to write this thesis without the help of the following persons.

First of all I want to thank ao. Univ. Prof. Dr. Dipl.-Ing. **Gerald Futschek** and Ing. Mag. **Johann Kadlicsek** for their incredible patience and guidance during all the time it took me to finish this thesis. Their constant support and advices have been an important help for this thesis and the inspiring source for many passages.

Furthermore I want to thank Dipl.-Ing. **Ömer Karacan** for giving me the opportunity to write this thesis and to participate in the project at Siemens Austria. His permanent help and support enabled me to do all the necessary research and preparation work for this thesis.

I would also like to thank Dipl.-Ing. **Franz Müller** for his cooperation and teamwork. Many details and issues became much clearer after our numerous creative discussions and talks together. He contributed a lot to this team project and was a very helpful and reliable partner and friend to work with.

In addition I want to thank Dipl.-Ing. **Gottfried Gniesser**, Dipl.-Ing. **Stefan Huber** and Ing. Mag. **Walter Makovicka** for their comments and suggestions. They helped me to continuously improve the quality of the thesis.

Finally I want to thank my parents and all my friends who helped me focussing on writing this thesis. Their efforts and motivation kept me on track and allowed me to complete my university studies successfully. Especially my parents and their support in all those years are responsible for a major part of this success. Without them I would not have been able to achieve this goal.

Preface

This thesis is part of a project I participated at the Program and System Engineering department of Siemens Austria. For this project I worked together with another student from my university who has been also writing his master thesis at Siemens Austria. The purpose of the project was to get a deeper understanding and insight into new emerging technologies in the software world. At the time when the project started new component architectures and their related technologies promised to provide a new and better way to build distributed and reliable software.

In the project we researched the two main software frameworks that were introduced just recently that time: the .NET framework from Microsoft and the J2EE framework from Sun Microsystems. My partner and me we were together responsible for the implementing and analyzing the part with the J2EE framework and its server-side component model called Enterprise JavaBeans (EJB). Therefore we build an EJB test application with all the requirements needed for completing the project and our thesis. While my partner focused on design patterns and issues concerning building applications based on the EJB component architecture in his thesis, I concentrated on the evaluation and description of the different platforms that can be used for this kind of application.

In Chapter 1 I will give a short overview of the global context and motivation for this thesis. The main relevant questions for this chapter are ‘Why do we need new software architectures?’ and ‘What is the purpose of this thesis?’. This chapter will try to give appropriate answers to these questions. Chapter 2 will describe and explain the concepts and ideas behind the Enterprise JavaBeans framework. The following Chapter 3 will discuss some alternative technologies and frameworks that are based on component paradigm, too. Chapter 4 will introduce the platforms that we will use in our evaluation and give a brief description of their origins and current state of development at the time of testing.

In Chapter 5 I will describe briefly the design and architecture of the client application that we used for the testing and evaluation of the different EJB platforms. For a more detailed discussion on design issues of the test application and other software related aspects I recommend my partner’s master thesis work with the “Object-Oriented Analysis and Design with Enterprise JavaBeans” [Mue01].The following Chapter 6 will specify the computer hardware and software that we used for our evaluation environment.

Chapters 7 and 8 are the core chapters for this thesis. In Chapter 7 I will discuss the concept, the criteria and the software simulation tests for the evaluation. The main issue is the creation of a checklist that can be used to compare different platforms. Usually each platform has certain unique features and tweaks that can not be found on other platforms. This fact makes it really hard to compare different platforms. So it would be very useful to have some kind of list with the common specifications and features that can be used for making a comparison matrix for several platforms. The results for each platform and the conclusions that can be given from the evaluation data are summarized in Chapter 8.

Even though the project has been completed in December 2001 it took me a long time to finish the writing of this thesis due to my studies abroad and other occupations. But these past years also give me the ability to see if the optimistic promises and outsights have been fulfilled by the each of the technologies. In Chapter 9 I will talk about how the future plans looked like at that time and compare them to the current situation.

For this thesis I will assume that the reader will have some basic knowledge in software engineering and computer related vocabulary. As a consequence standard notions and terms that usually are public known and common in use will not be explained in detail here. A glossary with important terms and abbreviations can be found at the end of this thesis.

Chapter 1

Introduction

In general, before you can write your first line of application code you have to do a lot of analyzing, designing and conception. At least you have to do so if your application is expected to have more functionality than a simple 'Hello World' program. Therefore the first step of software engineering consists of identifying the requirements and analyzing the problem you want to use your application for. Then you can start to design the software and decide which architecture and functionalities it should have. Afterwards you usually start implementing the application and your business logic following the results of the previous conception phases. The last step of the software engineering process is the testing phase, in which possible errors in the software should be eliminated.

As we can see from this traditional software engineering process the conception testing phase does indeed take a lot of work and time. Time is a very valuable resource these days, where projects have to be finished with less time available and products has to be on the market almost with light speed. 'Time is money' is the most important consideration of many companies nowadays. Especially with the internet and its hype as the new medium for doing business in the recent years, competition among software producers has become far more fierce and global. So companies are looking for ways to reduce the necessary time to market and make the software development process more efficient.

Besides the increasing pressure to improve the software development process companies face another problem that arises with the increasing importance of the internet in the global economy. Companies usually already have an existing hardware and software infrastructure and unfortunately these infrastructures differ a lot from each other. So with the introduction and integration of new technologies into existing legacy systems companies always face the same problem. Either change the whole infrastructure or make the new infrastructure compatible with the old ones. Both decisions have severe drawbacks that have to be considered when making a decision. Changing the whole infrastructure takes a lot of money and time, especially during the migration phase. Besides existing hard- and software are often well tested and in use for a long time. Therefore changing the existing infrastructure also means to take the risk of using a new system that might not be as stable as the old systems, because it is not as well tested. On the other hand making the new systems compatible to the old might bring up hardware or software constraints that will reduce the advantages of the new system.

1.1. Component paradigm

In the very beginning of the computer era software was written as monolithic programs. This proved to be very inefficient and not really applicable for the real world, where problems tend to be more sophisticated and many objects interact with each other. So the next step was to take an object-oriented approach in software engineering where software can be better mapped to the problem domain in the real world. This change in the building of software made the applications more flexible, less error-prone and much better to handle. Each object in the real world has its virtual counterpart in the software program.

But when applications grow to a very, very big size, the number of objects within the application can become overwhelming for any single software developer to handle. A more abstract and simplified view is needed for keeping the application structure manageable. Therefore objects are grouped together by functionality, semantic or technical dependency or any other criteria to become bigger entities that are called components. Components can be seen as more coarse-grained objects in an object-oriented concept.

This component architecture also allows a better modular concept for any software application, because dependencies between components are usually kept to a minimum. A component can be easier replaced by another component which might implement a slightly different functionality or a better algorithm for the same functionality as long as the external interfaces remain the same. So changing parts of the application does not have so many side effects as there are when using a pure object-oriented approach. Furthermore components can be easily reused and integrated into other applications. These advantages make the component architecture so popular for large and middle sized commercial software applications.

1.2. The internet as marketplace

In the last few years the internet has become a very important and influential aspect of our every day’s life. The term internet and its services like e-mail or www (World Wide Web) have found a widespread popularity. The amount of people who communicate through e-mail is increasing rapidly, and the global penetration of internet has already reached a very impressive level. So it was no wonder that some people would try to integrate this new communication media and technology into other aspects of social life. Nowadays terms like e-commerce, e-business, e-government or e-learning are familiar expressions for everyone.

Software has become an important part of our businesses and transactions in a more and more digitalized world. So the internet has become a very important and crucial aspect for this kind of business software, too. Nowadays large applications can be distributed and deployed all over the world over world, or business applications from two different enterprises can communicate and interact with each other. Geographic location has become a transparent and minor issue through global networking and especially the internet. But the internet is not only a medium for transporting and exchanging data, it has also become a large marketplace for any company. Access to the internet is provided from a wide range of different devices. Obviously software with access to internet has a great potential but has to solve a lot of complex issues, too.

1.3. Application servers

Usually when you start implementing an application you will write a lot of code to handle the whole infrastructure. This part of the application is responsible for handling access to the database, managing transactions, monitoring security restrictions, handling network traffic, etc. This infrastructure code usually has nothing to do with the business logic of the application. Therefore you have to write almost the same infrastructure code for each application again and again. So one might ask why not reuse this kind of code? With the component architecture this task becomes even more desirable. The code for handling the infrastructure does not even have to be written by the company itself. This is exactly the idea of an application server. The application server provides an environment where any application can use the existing infrastructure services. The provider of the server platform will only focus on the implementation of those services while the provider and developer of the application can focus solely on the domain specific semantics or business logic of the application.

While there have been many products and attempts to manage the issues above, they usually only take care of one single issue. Frameworks like Java 2 Enterprise Edition (J2EE) or .NET promise to incorporate all those advantages into one single product. They claim to enable software developers to build distributed, modular and flexible applications with the focus on implementing pure business logic in less time than without the framework. Especially the simple and transparent integration of existing legacy systems into those frameworks is often brought as an argument for their use.

1.4. Purpose and goals

The main purpose of this thesis is to present a comparison of several specific EJB platforms and a summary of how suitable they are for further and more widespread use within the software development department of Siemens or in any other software project. The result of this work should be a two dimensional matrix which shows an overview of different EJB platforms about the most interesting features, characteristics and technical details. This matrix should enable a project manager who looks for the proper platform to make a qualified decision according to his project requirements and needs. Furthermore this thesis should give a detailed introduction into the EJB component architecture and its implementation in several commercial products.

This thesis is based on the EJB specification Version 1.1 because the Version 2.0 had not been released by the time this project started. The relevant changes and additions to later Version of the EJB specification will be discussed in Chapter 9.

.

Chapter 2

The EJB framework

In March 1998 Sun Microsystems[1] presented its answer to the new challenges of the business software developing community. A new component architecture model and framework called Enterprise JavaBeans (EJB) was meant to solve the problems that the software developers of business software had to worry about in the years before. This new component model was designed to enhance and ease the development of distributed business application for the e-commerce sector. At the same time it should provide a standardized method for the integration of legacy systems into the migration process of corporate software. EJB is part of the Java 2 Enterprise Edition (J2EE) and is based on the Java programming language.

2.1. EJB architecture

The following figure shows the architecture of a simple 3-tier EJB application.

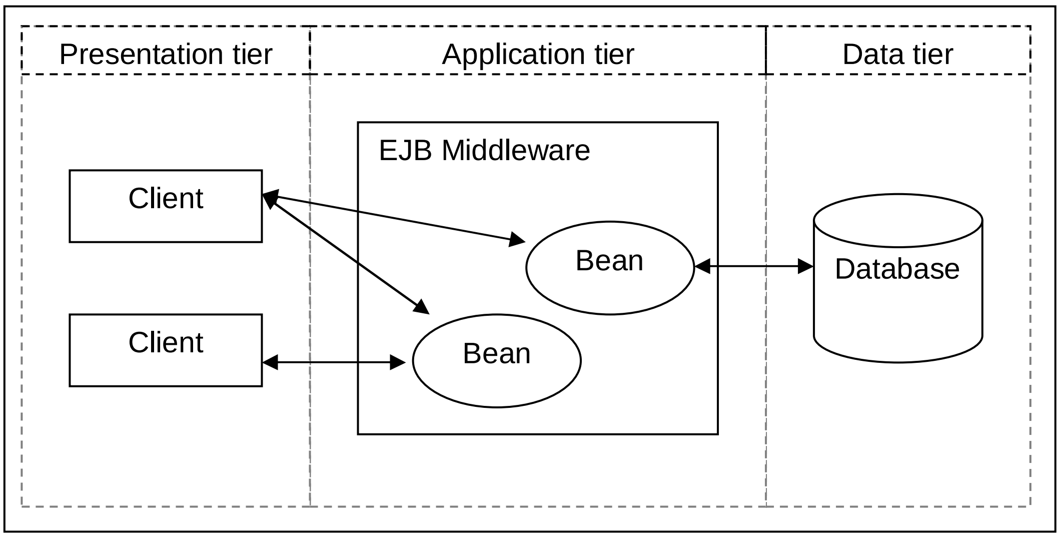


Figure 1 Example of a 3-tier EJB architecture

[1] Sun Microsystems, Inc. (see http://www.sun.com)

As we can see the in figure 1 the EJB concept is based on three distinctive tiers: The presentation tier, the application tier and the data tier. In the presentation tier the clients access the components from the application tier, which contains all the business logic for the application. The components are called beans in the diagram and reside in a runtime environment called EJB Container. The third tier contains all data for the application. In most cases this would be a database, but it can also be a legacy system which provides the data needed.

EJB Components

Each EJB component implements two interfaces: a home interface and a remote interface (see figure 2). The home interface is used by the client to locate a certain bean within the EJB Container and to get a reference to that bean. Furthermore the home interface also offers the methods to create or remove a certain bean within the EJB container. These are the method calls that control the life cycle of the bean.[2] The remote interface contains all the defined business methods that the client can invoke on the bean.

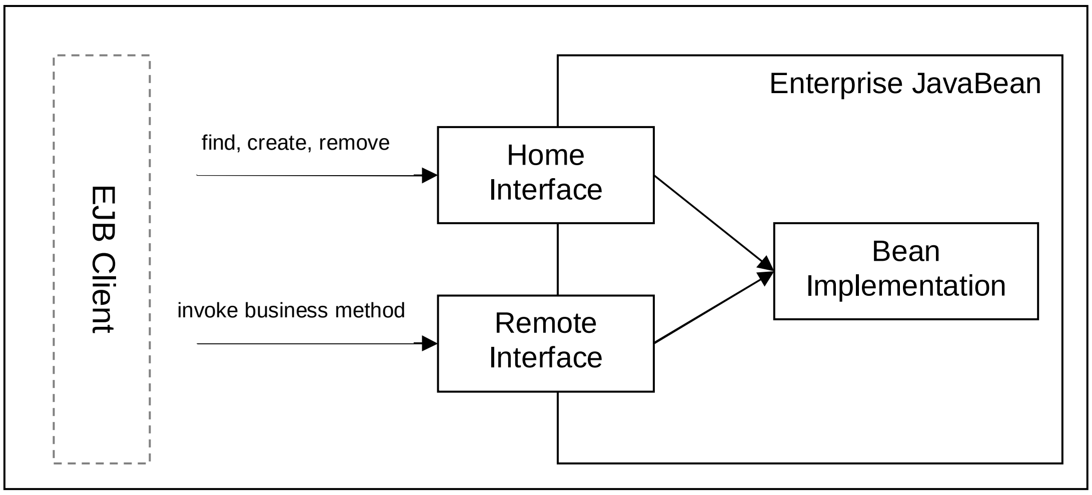


Figure 2 Enterprise JavaBean structure

There are primarily two different kinds of components within the EJB framework: entity beans and session beans. These two different types of components serve different purposes. While the Entity Bean type is meant for representing single data entities in the EJB concept, the Session Bean type is meant for implementing some kind of action or process abstraction within the framework.

[2] see following chapters 2.3 Entity Beans and 2.4 Session Beans

To get a better understanding of the different aspect of the framework and the architecture I want to describe the roles next that are specified in the EJB concept. These roles overview do also show which persons and what duties are involved in the building of an EJB application.

2.2. Roles in the EJB framework

There are six different roles specified in the EJB concept, each with distinct purpose and duties. With this kind of role specification each role can focus on its task and responsibilities.

- **Enterprise Bean Provider (Developer)**
  This person provides the enterprise bean, the component which contains all necessary implementation of the business logic to fulfill a certain task. Usually this person is a programmer who has a very good knowledge of how a specific task or action is processed.
- **Application Assembler**
  The Application Assembler takes several enterprise beans and combines them into one application. Usually this person does not need to know the implementation details of each bean, but should have a deeper understanding for the domain and area the application is used for. Furthermore the application assembler also integrates other non-EJB components into the application, like JSP (Java Server Pages) components.
- **Bean Deployer**
  The Bean Deployer is responsible for deploying the application into a given environment and server. The deployer should have a good knowledge of the EJB server and container into which the enterprise beans are deployed. Usually this person does not have to have programming knowledge because the deployment is done by graphical tools provided by the EJB server and container provider.
- **EJB Server Provider**
  The EJB Server Provider provides the services and infrastructure that an EJB container relies on. One EJB server can host several EJB containers which are the environments the enterprise beans live in. The EJB server is a generic platform which can be used to deploy any enterprise application.

- **EJB Container Provider**
  Every enterprise bean is deployed into a certain EJB container, which manages the life cycle for every enterprise bean and provide the infrastructure services that an enterprise bean can use. The EJB container, which encapsulates all deployed components, is an intermediate for the EJB server and the EJB components and provides the available interfaces for the EJB Bean Provider. Furthermore it is responsible for security and transaction issues, resource pooling and persistence handling.
- **System Administrator**
  The System Administrator is responsible to provide and maintain the necessary infrastructure for running the EJB server and to monitor the EJB server during runtime, usually with the usage of specific tools provided by the EJB Server Provider.

As we can see from the description for each role, the responsibilities are clearly stated for each role except for the EJB server and container provider. The EJB specification does not clearly define the dependencies and functionalities of these two components in the framework.[3] This fact and the close interaction between these two roles are the reasons why these two components are usually provided by the same vendor.

2.3. Entity Beans

Entity beans are components that represent persistent data in the EJB concept. All the data and information that the application uses is stored and handled in a third tier which is called data tier in figure 1. An entity bean refers to a partial view to that data tier and adds basic functionality that is closely linked to the data content. This data tier usually consists of some kind of large database system, in most cases a relational one. In this case a single entity bean can be interpreted as a record in such a database, e.g. a customer record in a contact database of a company. The entity bean as any other object in an object-oriented concept consists of its member attributes and methods.

[3] see [Sun99], p23: *"The current EJB architecture assumes that the EJB Server Provider and the EJB Container Provider roles are the same vendor. Therefore, it does not define any interface requirements for the EJB Server Provider."*

As I mentioned before each bean has a well defined life cycle. The life cycle for an entity bean is shown in figure 3. To get an instance of an entity bean the EJB container will explicitly instantiate a new EJB object and set the right context for this instance by calling the *setEntityContext* method to make it aware of the environment settings within the EJB container. After instantiation the entity bean becomes a member of a global pool which is managed by the EJB container. In this *Pooled* state the entity bean does not have any bean identity, which means that it does relate to any data in the data tier, it is a simple object instance. The number of instances in the pool depends on the EJB container and the available amount of memory resources. If the EJB container wants to reduce the number of instances in the pool it can call the *unsetEntityContext* method and remove the instance.

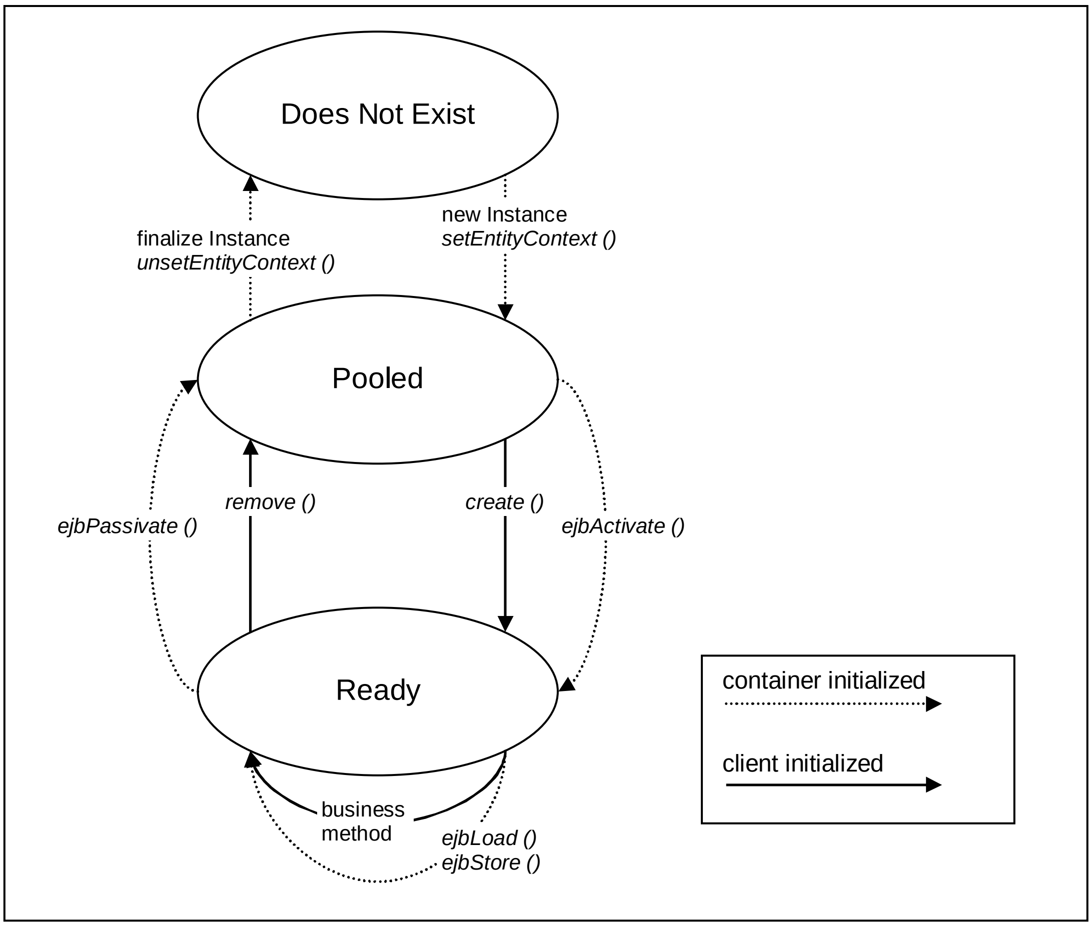


Figure 3 Life cycle of an entity bean

A client can insert new data into the data tier by calling the *create* method from the home interface of any EJB object provided from the EJB container. This will provide a bean identity for an instance from the pool and the instance will change to the

Ready status to indicate that it is ready for use. This instance will be the entity bean which will be returned for any client that wishes to access the data that the entity bean refers to or to call any business method from the bean's remote interface.

If the data that the entity bean represents is meant to be deleted, the *remove* method from the home interface can be called by the client. The bean identity will be removed from the bean instance and the data referring to that bean identity will be deleted from the data storage. In case of a relational database the related row in the database will be deleted. Then the bean instance returns to the pool of available instances and can be used for assigning another bean identity.

If the EJB container needs more memory resources, it can passivate some of the available entity beans (*ejbPassivate* method will be called). This means that the bean identity will be removed from an entity bean but the data still stays in the data storage. The order in which the entity beans will be passivated usually depends on the last client access for the specified bean. If the entity bean is accessed again by any client, it will be activated by reassigning the bean identity to an instance from the instance pool (*ejbActivate* method will be called). This mechanism allows the EJB container to dynamically react and adapt to the amount of available system memory.

Any invocation of a business method from the remote interface will not change the status of the entity bean. This is the same for the synchronization calls which will write/update the data to/from the data storage (call of *ejbStore/ejbLoad* method).

The EJB framework supports two types of persistence for entity beans.

- Container Managed Persistence (CMP)
- Bean Managed Persistence (BMP)

2.3.1. Container Managed Persistence

Entity beans with container managed persistence do not have to care about how and when data is persisted into the database. The container which manages the beans will handle all SQL statements and transactions for creation, update and deletion of data in the database. Usually a mapping tool is integrated in the EJB platform which will do the mapping of the object parameters to the fields in the database. According to the EJB specification the bean developer is responsible for declaring persistent fields of the bean class as either Java primitive or serializable types.

During the deployment of the bean into the container this mapping has to be specified in the deployment descriptor which is a XML file for the configuring the environment in the container. In the EJB specification, the XML deployment descriptor of a CMP bean provides *cmp-field* elements for identifying the persistent fields (container-managed fields) in the bean class. The *cmp-field* elements are used to differentiate between the fields that are written to the database and those that are not.

Managing the persistence this way enables the container to control the way how data is persisted and makes the application independent from the underlying data storage. The data storage can be replaced by another database or an existing system which provides the requested data.

2.3.2. Bean Managed Persistence

When using the Bean Managed Persistence the entity bean will handle the persistence by itself. The entity bean is responsible for running the SQL commands against the database and handling the data synchronization. Sometimes this kind of flexibility is useful when special handling for the persistence of the bean is required. For example if a legacy system is used as data provider for the EJB application, this mechanism allows the bean provider to access data and information in the legacy system. This enables the integration of existing systems to EJB applications or a soft migration from one system to another. The methods for managing and synchronizing the data of the entity bean with the data storage have to be implemented by the bean developer himself.

2.3.3. Primary Key Class

Each entity bean class has a primary key class that enables an EJB client to uniquely identify a certain entity bean. This primary key class usually consists of one or more persistent attributes from the entity bean. But it can be either any generic class, as long as it is serializable and unique for any entity bean instance.

2.4. Session Beans

Session beans are another type of component with different purpose than the entity beans. Session beans represent process and workflow entities in the component concept. Because EJB clients do not have any business logic, all the business logic for the application is modeled within session beans. Therefore an EJB client relies on the methods and functions that the session beans offer to perform a certain task. These tasks can be divided into two different categories according to their complexity and necessary data and information: complex and simple tasks. Complex tasks usually require several actions and consist of multiple subtasks that can be performed in different order. Examples for this kind of tasks are the processing of an online order or making a financial transaction with online banking. For this kind of process additional data and information often has to be kept for indicating the current status while performing different subtasks. Such tasks and processes are represented by stateful session beans. On the opposite hand simple tasks usually only consist of one single action and do not need such overhead; they are represented by stateless session beans. They usually provide general purpose and reusable services to any client. Some examples for such simple tasks are the conversion between different currencies or scales or the calculation of available seats in an online booking system for a theatre.

2.4.1. Stateful Session Bean

Stateful session beans have a conversational state and can only serve a single client during the whole client session. The information stored in this conversational state can be accessed by any invoked method of the remote interface. Figure 4 shows the life cycle for a stateful session bean.

When needed a stateful session bean can be created by calling the *create* method from its home interface. The session bean switches from the Does Not Exist status to the *Method Ready* status. In this status the session bean is ready for accepting and processing any business method calls from its remote interface. After each invocation of a business method it returns to the *Method Ready* status. Due to the circumstance that each session bean is uniquely assigned to a certain client, it is obvious that with an increasing number of clients the number of existing stateful session beans and their amount of occupied memory will rise, too.

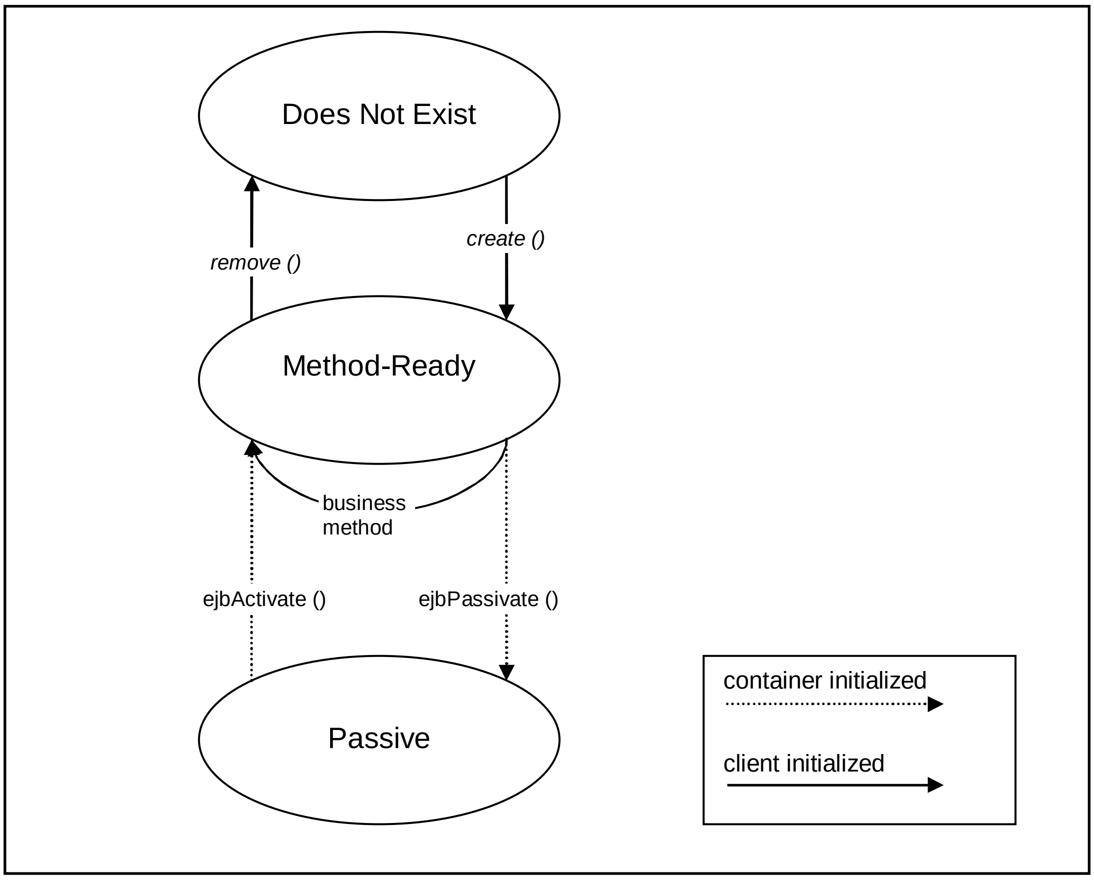


Figure 4 Life cycle of stateful session bean

In order to keep the memory usage within a reasonable bandwidth the same passivation-activation mechanism can be used for stateful session beans as it is done for entity beans. If the system needs more memory resources, the EJB container will call the *ejbPassivate* method and the data and information for not recently used session beans will be persisted to a secondary storage. As soon as the session bean is needed by its client the session bean instance will be restored with all its data and information. This whole mechanism as in the case with entity beans is fully transparent for the client; this means that the client usually does not know or care about the passivation or activation of any bean instances.

2.4.2. Stateless Session Bean

While stateful session beans serve a single client for an entire session, stateless session beans can serve several different clients because they do not have any conversational state and information. They are assigned to a certain client only for the scope of one method invocation. As a consequence to this fact they are much more scalable than stateful session beans. The EJB container will have a pool with ready instances for any stateless session bean class. The number of instances in this pool can be easily and independently managed by the EJB container according to the available memory and number of waiting requests.

The following figure 5 shows the life cycle for a stateless session bean. As we can see in this figure a stateless session bean can only have one of two different states. After being instantiated and initialized the instance will changed from the *Does Not Exist* status to the *Method-Ready Pool* status. This is done by the EJB container which manages the number of instances in the pool. Then each method invocation can be forwarded to any non-occupied instance of the pool. After completion of the method the instance becomes available for any other client requests.

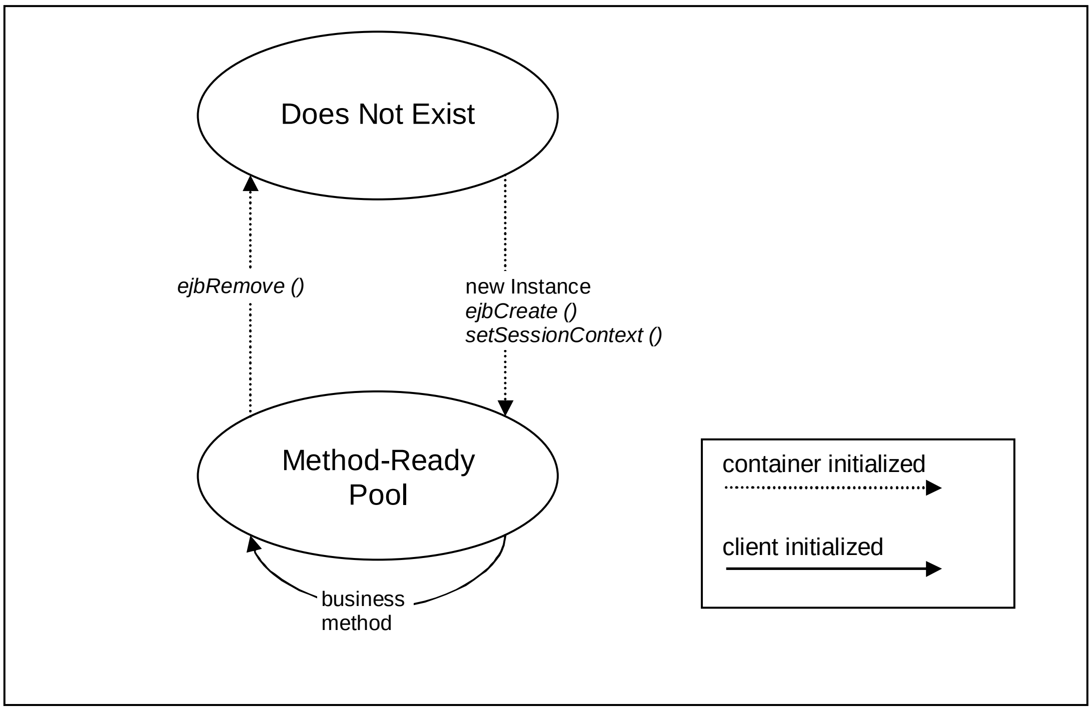


Figure 5 Life cycle of stateless session bean

In case the system memory resources are running low, the EJB container can free some memory by reducing the number of instances in the pool. Therefore the EJB container will call the *ejbRemove* method to destroy each session bean instance. For stateless session beans it makes no sense to passivate or activate them because they have no state information that could be stored to a secondary media.

2.5. EJB container

The EJB container is responsible for providing the necessary infrastructure services to the components that live in the container. These components contain the business logic for the application which is implemented by the bean developer. The options and settings for those infrastructure services are configured in the deployment descriptor for each component. With the help of this deployment descriptor the EJB container will generate additional helper objects and classes after the deployment of each component in the EJB container.

The generated object that is used on the client for the RMI (Remote Method Invocation) network communication is called object stub, while its counterpart on the server side is called object skeleton. Another helper object called EJB object is used for adding platform functionality. figure 6 illustrate how these objects work together to enable a smooth and proper operation of the deployed component. In this figure a remote method invocation of a business method is shown, and how the communication flow looks like in detail.

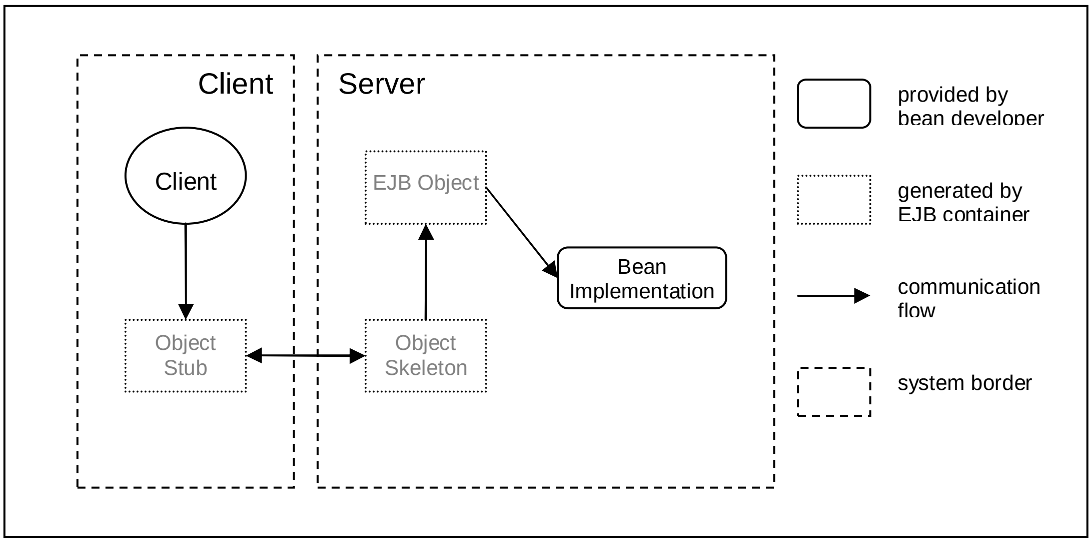


Figure 6 EJB object, object stub and skeleton

The client wants to invoke a certain method of the bean’s remote interface. This request is sent to the object stub which will forward it through the network to the proper object skeleton. The request then is sent to the EJB object which in turn will forward it to the actual method implementation of the bean. The EJB object, object stub and skeleton are all generated by the EJB container and invisible for the client.

The EJB object wraps the actual bean implementation provided by the bean developer. This gives the EJB container the possibility to add any relevant code and functionality to this deployed component, because each request and method invocation will be intercepted by the EJB object.

With this mechanism the business logic and basic system behavior of each component are totally separated from each other. In the EJB concept these parts are provided by different roles. So the two parties can work independently on their parts. The bean developer does not need to care about the basic system functionality for each bean, while the EJB container provider can implement a general algorithm for its infrastructure services.

2.6. Infrastructure services

In figure 7 a more detailed architecture of a sample EJB application is shown. As we can see from the figure the EJB container encapsulates the whole application tier with all the business logic. If the persistence is managed by the EJB container a CMP mapping tool will be used to access the data tier, other wise the entity beans will access the data tier directly. But no matter which kind of persistence is used the EJB container is responsible for triggering and monitoring the access to the data tier. On the other side the EJB container is able to intercept all communication that runs through the *Business System Interface*[4] as we have discussed in the chapter before.

With this kind of control the EJB container is able to integrate infrastructure services to all deployed components. The main infrastructure services that each EJB container provides are

- Concurrency
- Transactions
- Persistence
- Distributed Objects
- Naming
- Security

[4] layer between the business application and the client, because from the client's point of view the whole server-side business application appears a big system interface

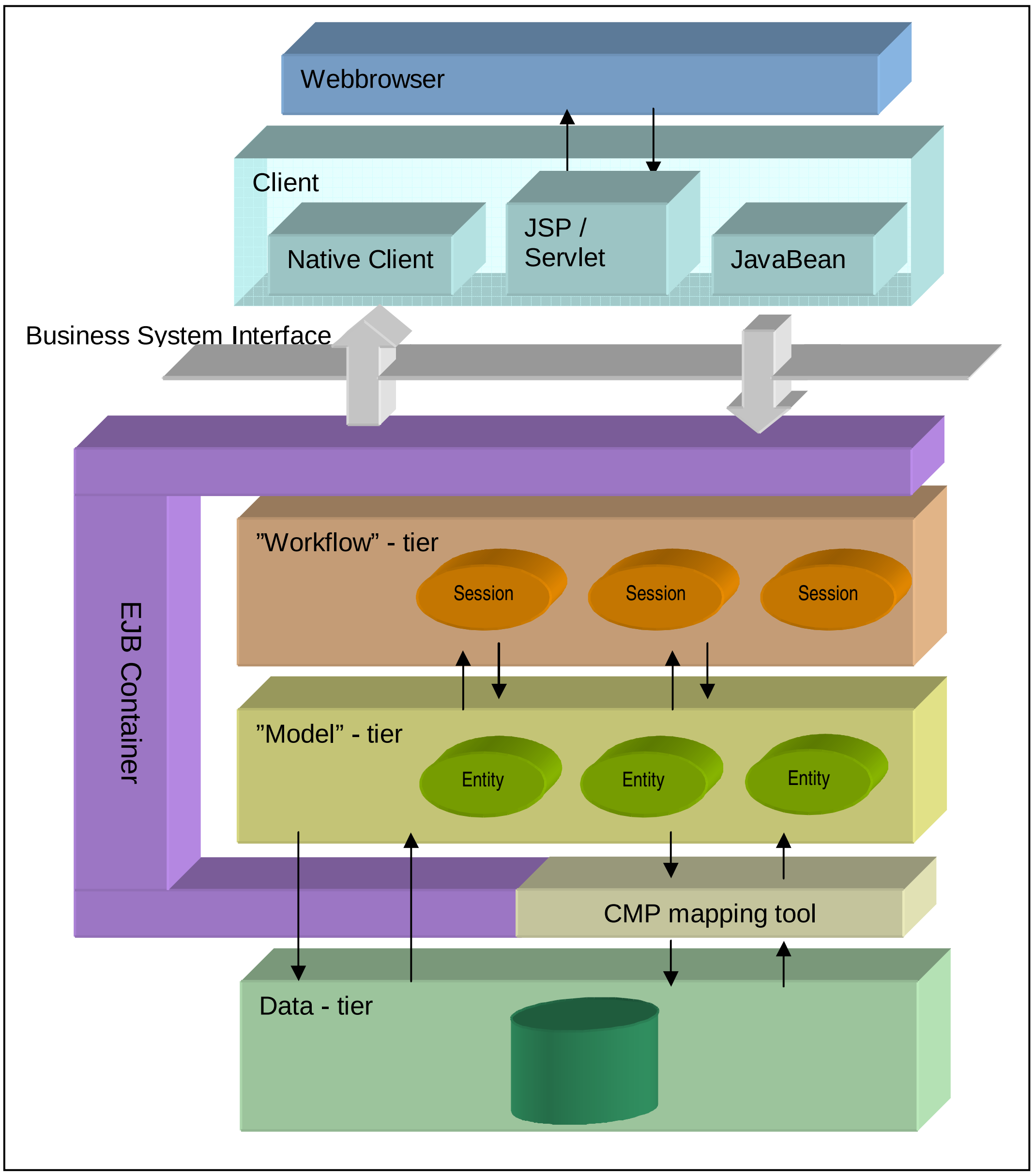


Figure 7 Sample EJB application architecture in detail

2.6.1. Concurrency

Due to their specification session beans (stateful and stateless) do not need to care about concurrency. Stateful session beans are accessed only by a single client so no concurrency will happen here. Stateless session beans are only accessed within the scope of a single method by each client, so there will not be any concurrent access here, too.

Entity beans can be indeed accessed by several clients at once. In the EJB architecture several clients can reference the same EJB object, but only one client can access the data of the entity bean at once. The handling for this is done by the EJB container automatically. Another important issue for concurrent access is reentrance. A bean X is called reentrant if it is allowed that bean X invokes a method on bean Y which in turn invokes a method on bean X. This would cause a loop back in the thread control, which is usually not wanted. But in some exceptional cases it might be a desired behavior. This is the reason why reentrance is possible for entity beans according to the EJB specification, but highly discouraged.

2.6.2. Transactions

For business applications the concept of transactions are a very important aspect. Similar to database transactions this concept will ensure that the system will not be left in an invalid state. Transactions in a business application span over several tasks that have to be completed together. If completion is not possible, all tasks have to be undone. For example an ordinary bank transfer consists of two tasks: a withdrawal from one account and a deposit to another account. If the deposit fails the withdrawal has to be undone, too. Otherwise the money would be lost from the system, which would produce an invalid system status.

The EJB container will monitor all tasks within a transaction automatically and ensure that all completed successfully. This kind of transaction control is called declarative, because this setting is declared in the deployment descriptor. Besides the transaction control can be transferred to the bean instance itself, if this is wanted by the EJB bean provider. This way the bean will have full control over the transaction handling.

2.6.3. Persistence

As we have seen in chapter 2.3 there are two different ways two store persistent data to a secondary storage. With CMP the data storing and loading is done by the EJB container, while with BMP all the handling is done by the bean itself. The three most common data storage for an EJB application are relational databases, object-oriented databases and legacy systems. Usually the EJB container will provide an integrated tool for mapping the data in the entity beans to the proper data in the data tier. In an object-relational mapping the fields of the entity bean are mapped to the fields in the database tables. But sometimes the mapping to a relational database can become quite complex because not all objects can be mapped easily to a relational database. Especially references between objects can be often a big problem. An object-oriented database provides a much better storage for such cases, because it allows a better and cleaner mapping to the database. Another popular way is to use a legacy system as data provider, especially when it is hard to migrate from the data from the legacy system to a database. The entity bean acts as an object wrapper for the legacy system data. This avoids an expensive system migration or extraction of data from the legacy system.

2.6.4. Distributed Objects

The EJB server provides an infrastructure where location transparency is achieved. Therefore the client does not have to care about where the objects actually reside in the network. The three most popular network protocols in use are the Java Remote Method Protocol (JRMP), the CORBA protocol and the DCOM protocol. For a Java client any of these protocols can be used, as long as the protocol maps to the Java RMI specification[5], because the EJB distributed interfaces are based on it. (The scenario is illustrated in figure 8)

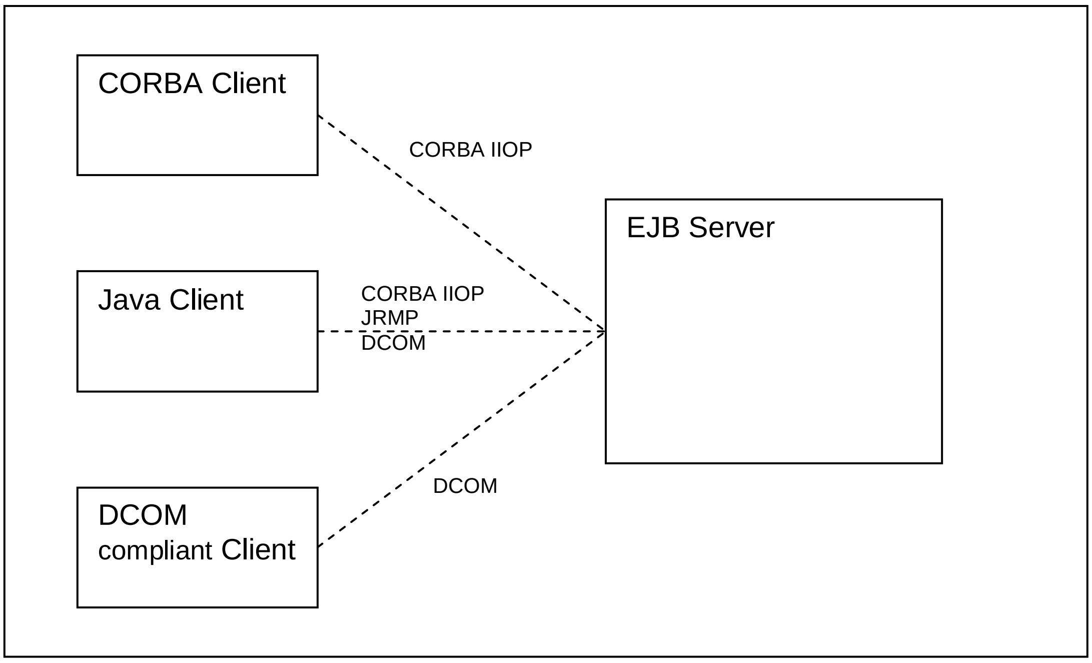


Figure 8 Access to an EJB server from different distributed clients

[5] Java RMI with the CORBA protocol is called Java RMI over CORBA IIOP (Internet Inter-ORB Protocol); Java RMI over DCOM is actually possible according to the EJB concept, but this would be a unrealistic scenario due to the better integration for the other protocols.

Clients written in other languages require the EJB server to offer support for the proper mapping to EJB for used protocol. If the client is written in a CORBA compliant language as C++, Smalltalk, Ada or COBOL the EJB server has to support an EJB-to-CORBA mapping as for example defined by Sun Microsystems [Sun99C]. A similar EJB-to-DCOM mapping would be required for clients written in any DCOM compliant language that want to access objects within an EJB server.

2.6.5. Naming

The naming service enables clients in the distributed environment to find other distributed objects. A naming service usually allows a client to perform two tasks: object lookup and object binding. With the object binding service a specific name can be assigned to a distributed object. So this bound object can be referenced by any client using the specified name. The object lookup service provides the interface for the client to connect to the distributed service and look for objects with a given name. A directory service is an advanced version of a naming service which allows distributed objects and other resources to be organized into hierarchical structures and adds more sophisticated management features to the whole system. Moreover metadata is usually available for describing the objects and resources. The EJB concept is based on the Java Naming and Directory Interface (JNDI), which has to be supported by every EJB server.

2.6.6. Security

Security is a very crucial and important issue to commercial business applications. The EJB architecture separates the source code handling the security from the source code that contains the business logic. The EJB specification encourages the bean developer to declare the security policy for a component instead of implementing it with in the bean. The EJB runtime is responsible for monitoring and implementing the declared security policy. This way the security policy can be easily adapted to any changes afterwards without changing the source code for the bean.

The EJB specification differs between three different security mechanisms that can be used for an EJB application: Authentication, Access Control and Secure Communication.

Authentication

Authentication is used to validate the identity of the user and avoid unauthorized access to the system. Authorization is usually done by requesting the user to login in to the system with a username and password, or by using other forms to prove his identity like ID cards or security certificates. Once the user has validated his identity to the system he gains access to the system and can use it.

Access Control

The Access Control mechanism will ensure that the user will only be able to access objects and resources in the system for which he has the proper permission. This is the only mechanism that is specified more in detail in the EJB specification, because the other two mechanisms are quite independent from the application and its business logic. Security roles and their permissions can be defined according to the security policy for the EJB application. These are specified during the deployment of the components into the EJB container who will monitor and control the access to the objects and their methods.

Secure Communication

Secure Communication between the server and the client can be achieved in two ways: by physical isolation or encryption. While physical isolation is a very expensive and not applicable to standard communication channels, encryption is a convenient way of protecting communication to be intercepted or manipulated from unauthorized persons. This is usually done by using SSL (Secure Socket Layer) for the communication between server and client.

Chapter 3

Alternative component models

EJB is not the only component concept and framework which tries to enhance the creation of distributed software applications. This chapter will introduce some alternative concepts to EJB based on the same component paradigm.

3.1. CORBA Component Model (CCM)

The Common Object Request Broker Architecture (CORBA) is a standardized specification for an open computing infrastructure in a distributed and heterogeneous environment. The specification is maintained and released by the Object Management Group (OMG)[6], a non-profit organization that produces and maintains computer industry specifications. To break the limitations with the earlier CORBA object model, the OMG decided to adopt the CORBA Component Model (CCM). Compared to the CORBA object model the CCM defines additional features and services that enable application developers to implement, manage, configure, and deploy components that rely on commonly used CORBA services, such as transaction, security, persistent state, and event notification services, in a standard environment. In addition, the CCM standard allows greater software reuse for servers and provides greater flexibility for dynamic configuration of CORBA applications.

The CCM will be included in the CORBA 3.0 specification. The CCM is a specification for creating server-side scalable, language-neutral, transactional, multi-user and secure enterprise-level applications. It provides a consistent component architecture framework for creating distributed n-tier middleware. Component written according to the CCM specification are called CORBA components.

---

[6] http://www.omg.org

The CCM architecture contains the following parts:

- CCM Containers
- CORBA components
- Portable Object Adapter (POA)
- Object Request Broker (ORB)
- CORBA object services like ORBA Transactions, CORBA Security, CORBA Persistence, CORBA Events, etc...

The defined roles in the CCM are very similar to the ones in the EJB role concept. Therefore the basic architecture is almost the same. Several CCM Containers can reside on one single server, each CCM container with its own deployed components.

3.1.1. CCM Containers

The CCM container acts as the interface between a CORBA component and the outside world. A CCM client never accesses a CORBA component directly. Any component access is done through container-generated methods which in turn invoke the component's methods. Depending upon the types of components that they can execute, CCM Containers may be divided into four categories:

- Service containers,
- Session containers,
- Entity containers, and
- Other containers

3.1.2. CCM Clients

The CORBA Naming interface (COSNaming) enables CCM Clients to find the CCM components they need and to create or obtain a CORBA object reference. A CORBA object reference is an abstract handle referring to an instance of a CORBA object. An object reference hides the location where the actual object resides and contains protocol information defined by the CORBA specification, as well as an opaque, vender-specific object key used to identify a servant that implements the object. Thus, existing component-unaware clients can invoke operations via an object reference to a component’s equivalent interface, which is the interface that identifies the component instance uniquely.

3.1.3. CCM Components

The CCM components are very similar to the components in the EJB architecture. CCM components contain the application logic for the CORBA application and are the basic actors in the CCM architecture. The CCM framework also implies a well-defined development cycle for CORBA components. In the first step component developers using CCM define the IDL interfaces that component implementations will support. Next, they implement components using tools supplied by CCM providers. The resulting component implementations can then be packaged into an assembly file, such as a shared library, a JAR file, or a DLL, and linked dynamically. Finally, a deployment mechanism supplied by a CCM provider is used to deploy the component in a CCM container that hosts component implementations by loading their assembly files. Thus, when the components execute in the server they are ready for processing client requests.

Figure 9 illustrate the structure of a CCM component as described in the specification. CCM components can offer support for different interfaces that are collectively called ports. These ports serve as interaction points with other components, CCM clients or objects within the application environment. There are four different types of ports each with a distinct purpose:

- Facet
- Receptacle
- Event Source
- Event Sink

Facets are interfaces that the component provides for external requests. The component has to encapsulate an implementation for each exposed facet interface (*Facet Implementations*). Interfaces that the component is able to facilitate and accept are called *Receptacles*. Interfaces that can emit or publish events are called *Event Sources* while their counterparts for accepting of processing of external events are called *Event Sinks*. In addition to these ports, all CCM components support inheritance and have additional *Attributes* that can be accessed. Inherited interfaces from other components are called *Supported Interfaces*. The Component reference is primarily used for identifying the component.

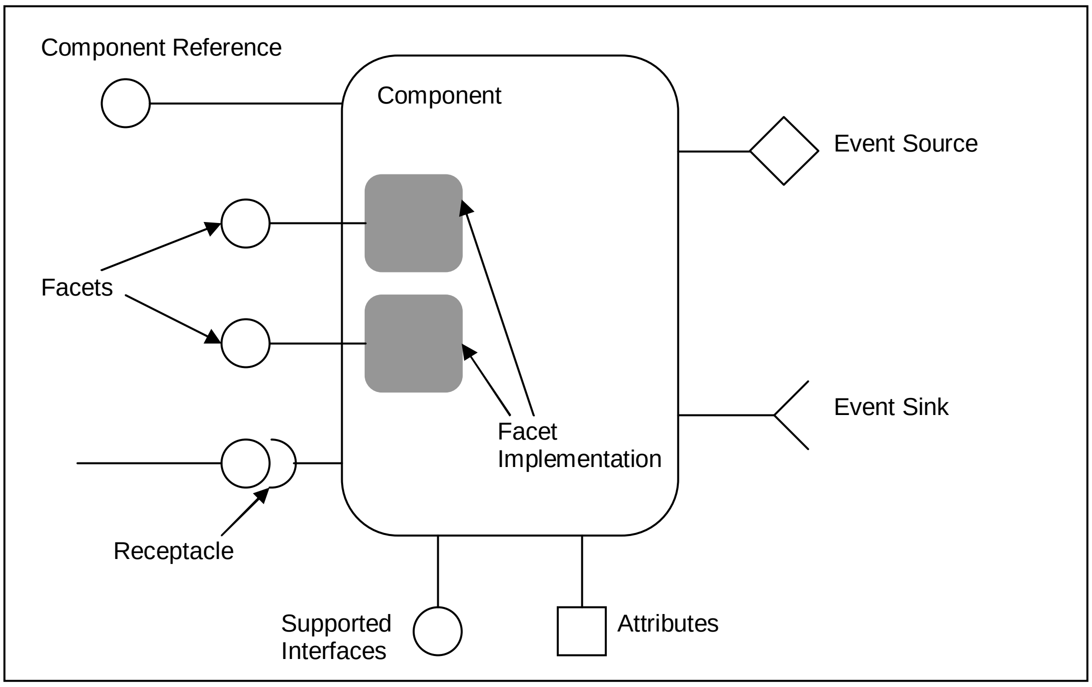


Figure 9 CCM component

There are four types of CCM components.

- Service components
- Session components
- Process components
- Entity components

Service Components

Each Service component is usually associated with one CCM Client and its lifetime is restricted to that of one single operation request (or a single method call). Each Service component is created and destroyed by the particular CCM Client that it is associated with. Service components do not survive a System shutdown.

Session Components

Each Session component is usually associated with one CCM Client. Each Session component is created and destroyed by the particular CCM Client that it is associated with. A Session component can either have states or they can be stateless. However, Session components do not survive a System shutdown. A Session component is very similar to a session bean in EJB.

CCM containers have a right to manage their pool of instances and they use the same mechanism as EJB containers. If additional resources are needed they may either use CORBA persistence or a user-defined persistence mechanism to passivate active instances to a persistent storage. When the instance becomes activated the state of the component instance is restored by swapping it in from persistent storage.

There are two types of Session Components.

- Stateless Session Components
- Stateful Session Components

Stateless Session Components

Like stateless session beans in EJB these types of components have no internal state. Since they do not have any states, they need not be passivated. Because of the fact that they are stateless, they can be pooled in to service multiple clients.

Stateful Session Components

Like stateful session beans in EJB these types of components possess internal states. Hence they need to handle persistence to be passivated and activated by the CCM container. These types of components can be saved and restored across client sessions.

Process Components

Process components represent process entities in this component architecture and they always have states. Each Process component may however be shared by multiple CCM Clients. Their states can be persisted and stored across multiple invocations. Hence they can survive System Shutdowns.

Entity Components

Entity components always have states. Each Entity component may however be shared by multiple CCM Clients. Their states can be persisted and stored across multiple invocations. Hence they can survive System Shutdowns. Each Entity component can be uniquely identified by its Primary Key. An Entity component is very similar to an entity bean in EJB.

One of the major differences between Process and Entity components are that while the Entity component has a Primary Key to uniquely be identified by the client, a Process component does not expose its identity to the client except through user-defined operations. While Entity Components are used to represent entities like customers or accounts, Process components represent business processes like applying for a loan or creating a work order, etc.

Persistence in Entity and Process components is of two types

- Container-managed persistence
- Component-managed persistence

Container-managed persistence

Here, the CCM container is responsible for saving the component's state. Since it is container-managed, the implementation is independent of the data source. Persistence is automatically handled by the container.

Component-managed persistence

Here, the Entity component is directly responsible for saving its own state. The container does not need to generate any database calls. Hence the implementation is less adaptable than the previous one as the persistence needs to be hard-coded into the component.

Portable Object Adapter

The POA allows programmers to construct servants that are portable between different ORB implementations. Portability is achieved by standardizing the skeletons classes produced by the IDL compiler, as well as the interactions between the servants and the Object Adapter.

Packaging and Deployment

CCM uses XML descriptors for specifying information about packaging and deployment just like EJB. However, additionally, CCM has an assembly descriptor, which contains metadata about how two or more CCM components are wired together.

3.2. Lotus Domino

Lotus Notes [Lot00] is one of the most popular groupware systems. The software is based on a simple three tier architecture, where the tiers are called levels. The same architecture which is shown in figure 10 is used for both client and server. Each software component belongs to one of the three levels:

- Client and server programs
- Notes Object Services (NOS)
- Databases and files

A component in the Domino architecture is a piece of compiled C or C++ code that is distributed as a dynamic link library (DLL). NOS are an example for such a component. A Notes application is the design of a Notes database (or complex application can design a whole set of individual databases that are linked to each other), which usually has the following definitions: the type of documents in the database, the way the documents can be indexed and viewed and the application logic which is written in one of four interpreted languages (Notes Formula Language, LotusScript, Java or JavaScript). A Notes database is a single file which stores the documents and application logic for the creation and modification of those documents.

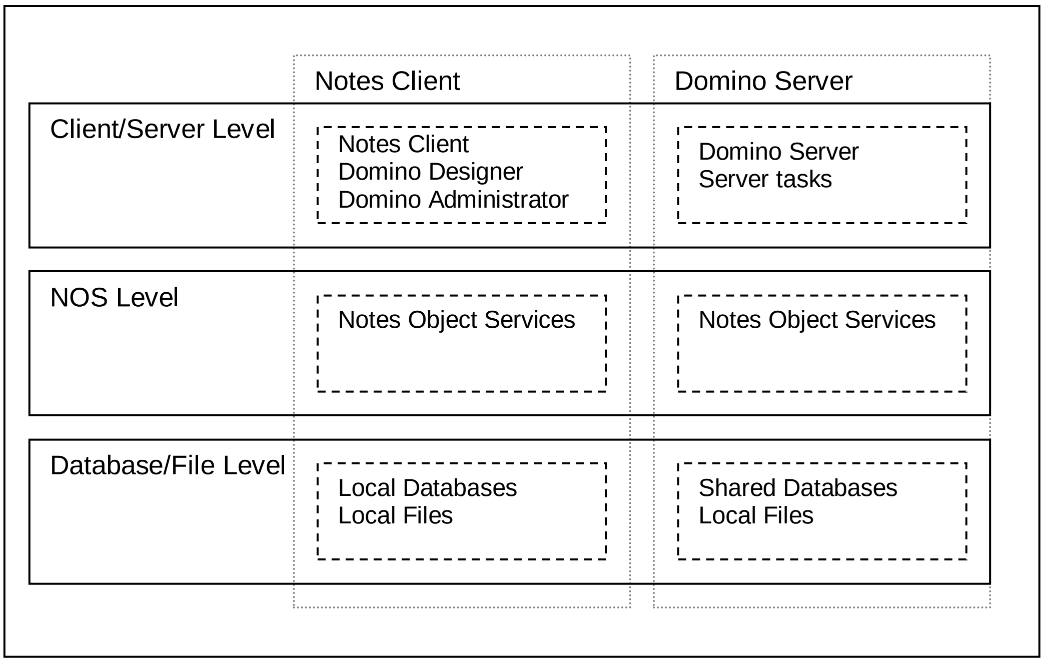


Figure 10 Domino three tier architecture

An overview of the Domino architecture and the relation between server and client computers is shown in figure 11. A Domino network usually consists of multiple servers (called *Domino Server*) and numerous clients (called *Notes Clients*). Notes Clients and Domino servers have a different portfolio of available programs that can run on the Client/Server level.

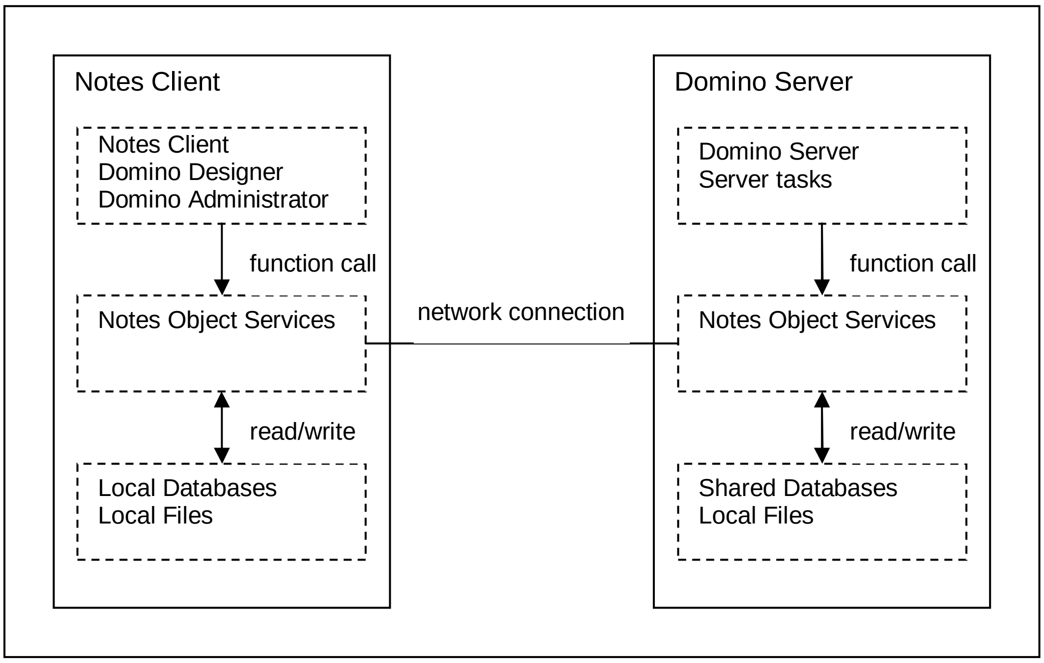


Figure 11 Domino architecture overview

Client and Server Programs

Both client and server programs use NOS to create, modify, read, and maintain databases and files. These are the main worker nodes in the network. The Notes Client, the Domino Designer, and the Domino Administrator which are installed on client computers allow interactive and GUI (Graphical User Interface) based access to databases and files local to the client computer and to shared databases.

Server programs

On server computers, the Domino Server program supports the connection between clients and the server and also manages a set of server tasks. These server tasks can either run time triggered database queries and updates or perform other actions like routing messages. In addition they provide connectivity for different types of clients like a Web browser or a CORBA client which want to connect to the server.

Notes Object Services (NOS)

The Notes Object Services is a set of portable C/C++ helper functions that allow the Notes user to create and access information in databases and files, compile and interpret formulas and scripts. They act as an interface to operating system services in a consistent, portable way. Using C-language callback functions, you can customize many NOS functions. Additional functionality can be easily added by integrating new NOS functions.

Databases and files

As we can see from figure 11 application data can be stored in two different storages: databases and files. Files are local. Databases can be either local or shared. Server computers have shared databases; client computers have local databases; and both have local files.

Shared databases can be accessed over the network by a program running on another computer. The Domino Server program is the only program in the framework that contains the logic to respond to incoming requests from another computer on the network for accessing a database. As a result to this limitation shared databases can only reside on Domino servers. Because NOS implements the logic that posts requests to access a shared database and because NOS runs on all client and server computers, programs running on any client or server computer can request access to a shared server database. The server may deny specific requests, however, if the requester lacks the proper access rights. When a program running on one computer accesses a shared database residing on another computer, the shared database is considered to be a remote database, with respect to the program accessing it.

A database or file is local if it can be accessed only by programs running on the same computer. Databases on client computers are local because client programs do not have the ability to accept incoming requests from other computers on the network. Client programs can only send requests for accessing a database to server computers. Therefore only programs running on a client computer can access databases on the client computer.

Databases contain most of the data in a Notes network, but some application data is kept in non-database files, such as ID files and the NOTES.INI file. These files exist on client and server computers and are always local because neither the client nor the server program contains the logic required to request or provide shared access to non-database files.

The Notes database

The Notes database is the cornerstone of Notes architecture. The majority of the Notes program is concerned with creating, maintaining, editing, viewing, accessing, copying, and replicating Notes databases. Each Notes database contains:

- A database header and other internal structures
- Notes, which fall into three categories: design elements, administrative notes, and documents
- Replication history (optional)
- Objects attached to notes (optional), e.g. file attachments
- The database header and other internal structures

The database header and other internal structures keep track of key database information, such as database creation time, and of notes and their attached objects.

The database header

The database header stores a time stamp that indicates when the database was first created or when it was last fixed-up. A database is fixed up, when notes that were corrupted as a result of a server crash are purged. This time stamp also serves as the database ID (DBID). In addition, the database header holds the unique replica ID, as well as links to the database replication history and to other internal structures that track database notes, attached objects, and free space in the database file.

Identifiers

Each note in a database has two identifiers: the note ID and the universal ID (UNID). The note ID is a 4-byte value that is assigned when the note is first created. Every database has a record relocation vector (RRV) table that maps a note’s note ID to the position of the note within the database file. This table simplifies relocating a note within a database. When a note changes location, the RRV table updates to reflect the new location.

The UNID is a 16-byte value that is assigned to the note when the note is first created. A UNID uniquely identifies a note relative to all other notes in the universe, except for special copies that have the identical UNID so that they can be identified as being the same note as the original one for special purposes. For example, when replicating, or synchronizing, the notes in replica databases.

Every database has a UNID table that maps the note UNID to its note ID, which in turn can be mapped through the database RRV table to the note’s position within the database file. UNIDs are used when replicating database notes and when replacing or refreshing a database design notes.

The named-object table maps names to associated notes and objects. For example, this table manages per-user views, which are also known as personal or private views, and per-user unread lists. The names assigned to these views and unread lists are composed, in part, of the user’s name.

A note in Lotus Notes

A note is a simple data structure that stores database design elements (forms, views, and so on), user-created data (documents), and administrative information, such as the database access control list (ACL). Because the same note data structure stores all these types of information, Notes requires only a single a set of NOS services to create, read, update, and replicate most of the information in a Notes database.

In figure 12 the logical structure of a note is illustrated. Each note has a small header followed by a list of variable-length items, which are also known as fields. The header holds general information about the note, including a value that indicates the note’s class - for example, document, form, or view - and its originator ID (OID). The OID contains the note’s unique, universal ID (UNID), which is essential for replication. Within the item list, each item has a name, attribute flags, a value, and a value type (e.g. text or number).

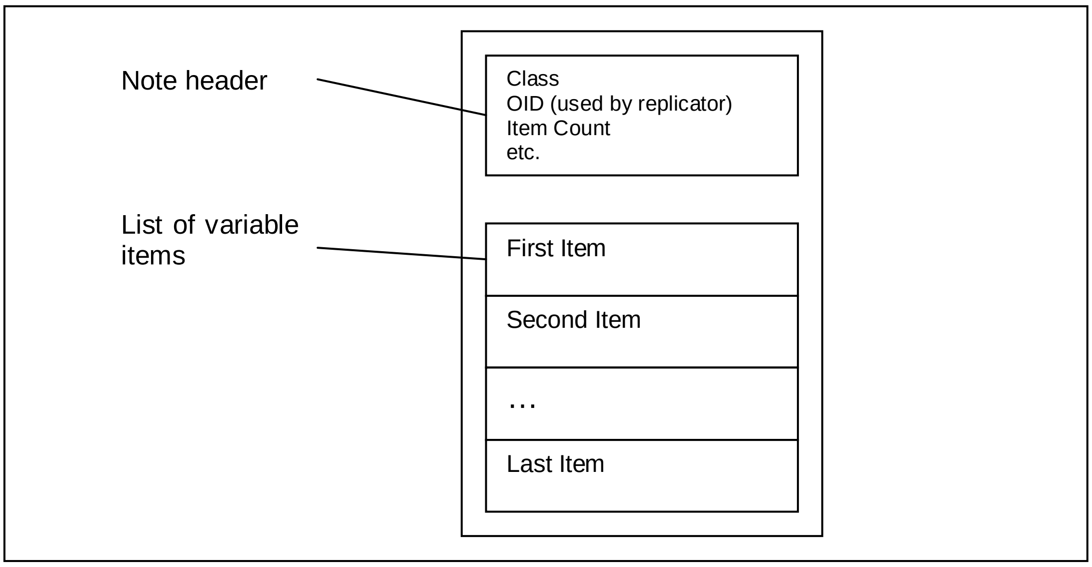


Figure 12 Logical structure of a note

Every note contains a set of items that is determined by the class of note. For example, all form notes contain the same set of items, although the item values differ from form note to form note. Similarly, all view notes contain the same set of items, although the item values differ from view note to view note. Document notes are different, however, because all documents do not contain the same set of items. Because the set of items in a document depends on the form used to create the document, two document notes may have vastly different item lists.

Data notes

Data notes, or documents, typically comprise the bulk of a Notes database. Each document can be associated with the form note that was used to create the document and that is used by default to view or modify the document. When presenting a document, Notes will use that defined form note to apply to the document. This approach provides more flexibility than does a model that tightly binds a document to one specific form. For example, although each document has a default form associated with it, alternative forms can be applied to the document so that the content is presented in many different ways. In addition, through the use of field values and/or the result of a formula computation, Notes can dynamically control which form to use. Both the Notes client and Web browsers support this unique late-binding model of presenting information.

Although forms and documents are usually stored separately, a document’s form may be stored within the document itself. The form is actually stored as a set of items that belong to the document. Storing a form this way makes it possible to copy a document from one database to another database that does not contain the form necessary to view and edit the document. This option, however, has a drawback in that if used too frequently, it can significantly increase the size of a database.

To support applications that categorize and subcategorize documents, individual data notes can be arranged hierarchically. A note can be a main note, a response to a main note, or a response to a response note. Up to 32 levels can be used to create a hierarchical structure for those notes. For example discussions are typically structured in threads that require a hierarchical order.

Administration notes

There are two types of notes that are created and managed by the database manager: the access control list note and the replication formula note. Each database has only one access control list note, which lists the access rights that various users, servers, and groups have to other notes in the database. Replication formula notes, which are optional, specify, on a server-by-server basis, which subset of notes to replicate when the database replicates with replicas stored on other servers.

Design-element notes

Design-element notes are created by the database designer, who can create elements that can be used to create forms. Forms are graphical interface for the user to interact with and view data in databases. These elements can include application code (like event handlers or software agents), graphics or other information that is used by the application developer.

3.3. Component Object Model plus (COM+)

Microsoft offers its own server-side component based software framework called .NET. The .NET framework is Microsoft's platform for distributed and internet-based applications like Web services or e-commerce portals. The server side component model for the .NET framework is COM+, which is an extension of Microsoft's Component Object Model (COM). This component concept has emerged from the Object Linking and Embedding (OLE) technology, which was driven by the idea to integrate and combine different documents together for enhanced and more powerful editing. This mechanism would allow different parts of a combined document to be edited with the proper associated application or editor and without having the user to think about how to those applications can interact with each other. A well-known example is an Excel spreadsheet which is integrated directly into a Word document. But the COM concept allowed even more than the OLE functionality, because it defined a standard mechanism of how different components can interact with each other and how they can access public methods from other components.

In the COM concept, objects (or classes) and their methods and associated data are compiled into binary executable modules, that are, in fact, files with a dynamic link library (DLL) or EXE file name suffix. A module can contain more than one class. With the emergence of distributed application and the World Wide Web applications the Distributed COM (DCOM) technology was introduced to the COM framework. DCOM allowed the components to reside somewhere within in a distributed environment, instead of having all components on the same computer. Then COM evolved into COM+ which was meant to provide an enhanced model that makes it relatively easy to create business applications that work well with the Microsoft Transaction Server (MTS) in a Windows NT or subsequent system.

Actually COM+ is a collection of operating system services and facilities for building scalable, distributed and secure application. It provides a wide range of different system services for application components, such as notifying them of significant events or ensuring they are authorized to run in the given environment. Due to the important role of COM for almost all Microsoft products, it is not surprising that COM+ services and technologies has become tightly integrated into the Windows 2000 and XP operating systems.

COM components and interfaces

In the COM concept a component is a piece of software that is self-describing. This means that it can be run with a mix of other components and each will be able to understand the capabilities and characteristics of the other components. Furthermore a new application can be usually built by reusing components which already exist and without having to compile the whole application. Another advantage is that it is quite easy to distribute different components of an application among different computers in a network. A component consists of one or more classes that describe objects and the methods or actions that can be performed on an object. A class (or coclass in COM+ terminology) has properties described in an interface (or cointerface). The class and its interface are language-neutral. Interfaces are described using Microsoft's Interface Definition Language (IDL).

In figure 13 the structure for a COM component is shown. The actual implementation is encapsulated within the component and only the proper interfaces that the component implements are exposed. A COM component usually implements several interfaces. Each interface has to be derived from the basic interface *IUnknown*, which offers three basic operations/methods: *QueryInterface*, *AddRef* and *Release*. The *QueryInterface* method can be used to check weather a given interface is supported by the component or not. If the given interface is supported by the component the proper interface pointer will be returned.

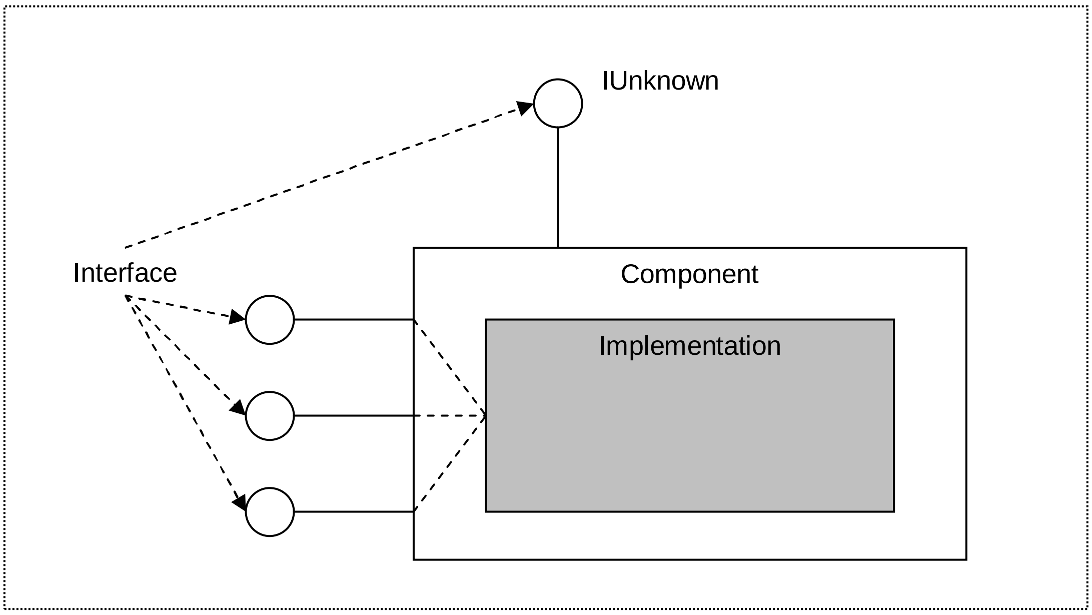


Figure 13 Structure of a COM component

The other two methods are used for the reference counting mechanism. Because all instances are instantiated or deleted by the COM+ framework, this mechanism is used to track if there are any existing references to the instance. If the instance is not referenced anymore it can be deleted safely.

Component instantiation

In the following figure the creation of a remote component instance is shown and how the request for a new component instance is processed in the framework.

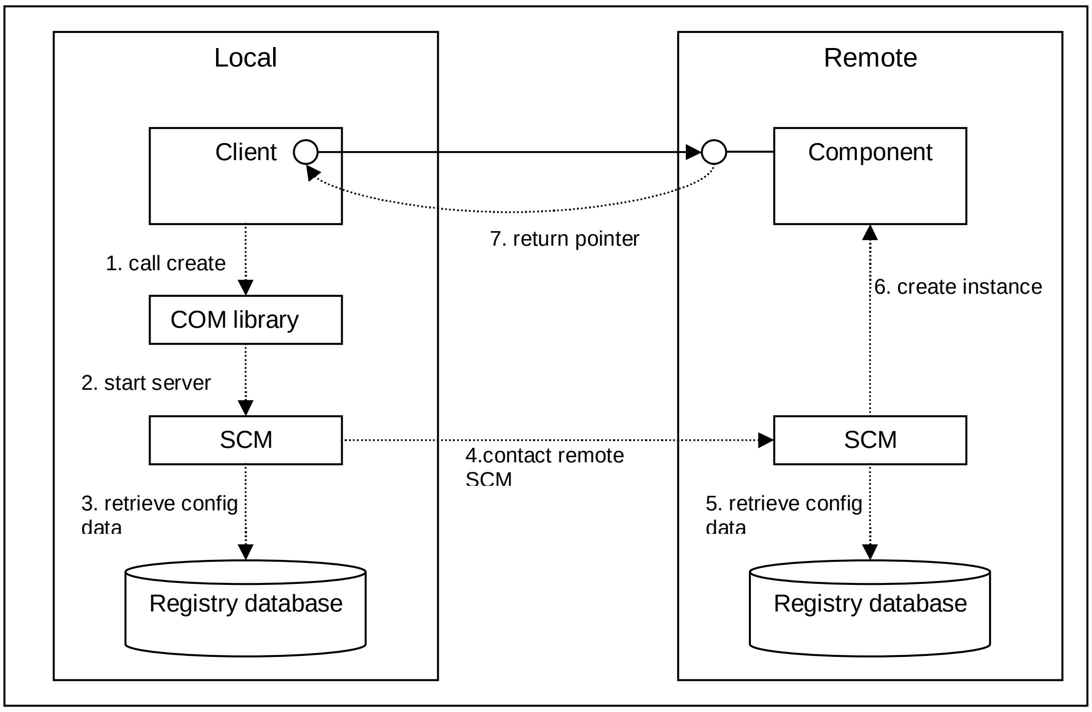


Figure 14 DCOM component instantiation

In the first step the local client will request the creation of a new instance of a component that exists on a remote computer. If the Service Control Manager (SCM) on the local computer is not already running it will be started by the COM library. Because the component does not exist on the local computer the local SCM will contact the SCM on a remote server to initiate the creation of a component instance. The remote SCM will retrieve the necessary configuration data for that component from its registry database. Then the remote SCM will create an instance for the requested component and return the interface pointer for the newly created instance to the local client. With this pointer the local client can access all methods and properties described in the associated interface.

Identifiers

Clients need the ability to find and refer to s specific component. To be able to uniquely refer to a certain component, components need some kind of identification mechanism. The COM+ framework uses two different identifiers for this purpose: an Interface Identifier (IID) to identify interfaces and a Class Identifier (CLSID) to identify classes. Both identifiers have to be so called Globally Unique Identifiers (GUID), which are 128-bit long, automatically generated character combinations.

System services

The COM+ framework provides different infrastructure services for COM components. These services include security, concurrency, persistence and life cycle management.

The security concept differs between two types of security checks: Activation Security and Call Security. Activation Security controls which users are allowed to start a server. The list of authorized users is stored in encrypted Access Control Lists (ACL). These lists allow a more detailed and fine grained configuration. The Call Security checks control the access to existing components and their interfaces. The access rights for this mechanism can be configured on system process or interface level. If they are set for a specific system process the check will be done when the system process starts. Otherwise the access rights can be managed on interface level by the developer within the client or component source code.

The concurrency concept is based on the mechanisms for threads of a 32-bit Windows operating system. Besides it offers compatibility to 16-bit operating systems and components by supporting the Apartment Model mechanism. With this mechanism the components can run in a Single-threaded Apartment (STA) or a Multi-threaded Apartment (MTA). A STA can serialize access to components that do not support Multi-threading. So these components can be used safely within a multithreaded environment, because all access requests are automatically pooled and controlled by the framework. Whereas components that run in a MTA are accessed directly, because they implement their own concurrency control.

The persistence concept in COM+ differs much from the persistence concept in EJB. COM+ does not offer any automatic persistence of data or mapping to databases. The component developer is responsible for writing the necessary information of a component to a persistent storage. The framework does only offer several interfaces that can be used by the developer for this purpose.

All components live in a component server which handles the life cycle for each component instance. These servers also make the location of a component transparent to any client. For a client it is irrelevant where the requested component actually is, because it does not matter if the component exists in the same computer or not. The component will be accessed by the client always in the same way. This location transparency is illustrated in figure 15. An interesting difference to EJB can be seen here. The server side object is called *object stub* in COM+ whereas in EJB the client side object is called *object stub*. The client side object in COM+ is called *object proxy*.

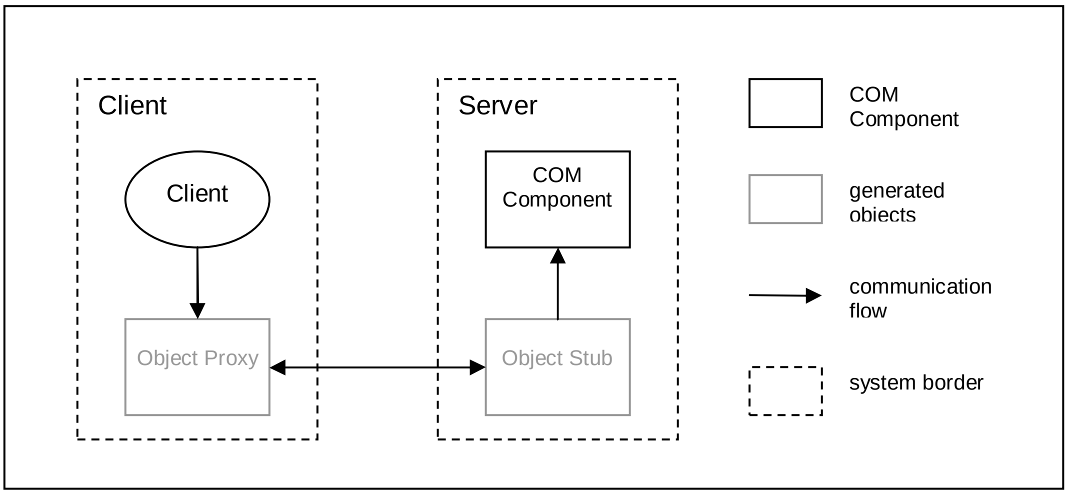


Figure 15 Object Proxy and Stub

Mono

The .NET framework is language independent but platform specific, which means that the components can be implemented in any programming language but they will only run on Windows platforms. An open source project called Mono[7] has re-implemented the .NET framework to provide support for multiple platforms especially Linux. The first release has been finished recently and is available for download on the project’s homepage.

[7] see http://www.mono-project.com

3.4. XPCom

The Cross Platform Component Object Module (XPCOM) is a framework which allows developers to break up monolithic software projects into smaller modularized pieces. These pieces are then assembled back together at runtime. XPCOM is part of the open source project Mozilla[8], which provides a freely available Web browser and Email client. But XPCOM can be used for other kinds of standalone applications, too.

With XPCOM it is possible to split up larger software projects into smaller pieces, that can be developed and build independently of one another. These pieces, known as components, are usually delivered in small, reusable binary libraries (a DLL on Windows, for example, or a DSO on UNIX), which can include one or more components. When there are two or more related components together in a binary library, the library is referred to as a module. In order to provide interoperability between components within an application, XPCOM separates the implementation of a component from the interface.

But XPCOM also provides several tools and libraries that enable the loading and manipulation of these components, services that help the developer write modular cross-platform code, and versioning support, so that components can be replaced or upgraded without breaking or having to recreate the application. Using XPCOM, developers create components that can be reused in different applications or that can be replaced to change the functionality of existing applications.

XPCOM not only supports component software development, it also provides much of the functionality that a development platform provides, such as:

- component management
- file abstraction
- object message passing
- memory management

[8] http://www.mozilla.org

Although it is in some ways structurally similar to Microsoft COM, XPCOM is designed to be used principally at the application level and under multiple operating systems like Windows and Linux.

Interfaces

Interfaces allow developers to encapsulate the implementation and inner workings of their software, and allow clients to ignore how things are made and just use that software. An interface can be seen as a contractual agreement between components and clients. Usually in component-based programming, a component guarantees that the interfaces it provides will be immutable. The clients will be able to access the same methods of a component via the interface because the clients are shielded from the inner workings of the component. Even if the implementation changes, e.g. across different component versions, the client code does not need to be changed. In this respect, interface-based programming is often referred to as programming by contract.

In XPCOM all interfaces are derived from the base interface nsISupports, which provides crucial functionality to all XPCOM components. The definition of the nsISupports interface is shown in Listing 1.

```
class nsISupports
{
  public:
    long QueryInterface (const nsIID & uuid, void *result) = 0;
    long AddRef (void) = 0;
    long Release (void) = 0;
};
```

Listing 1 nsISupports interface

The method *QueryInterface* is used to check if the component for which this method is invoked supports the interface given by its IID (Interface Identifier). If the interface is supported the proper result code is returned and the client can safely call the methods of the specified interface. The methods AddRef and Release are used to manage the reference count of the component during runtime. The components reference count and the ability to ask a component for the supported interfaces are necessary for solving two fundamental programming issues in XPCOM: Object Interface Discovery and Object Ownership.

Object Interface Discovery

In general components implement a set of different interfaces. Therefore a client will need a mechanism to check what interfaces are supported by a specific component. When a client wants to discover if an object supports a given interface, the client passes the IID assigned to that interface to the QueryInterface method of that object. The IID is used to uniquely identify an interface. If the object supports the requested interface, it adds a reference to itself and passes back a pointer to that interface. If the object does not support the interface an error is returned and the return value will be null.

XPIDL and Type Libraries

All public interfaces in XPCOM have to be defined in XPIDL (Cross Platform Interface Definition Language) syntax. XPIDL is a variant of the CORBA OMG Interface Definition Language (IDL), which allows you to specify methods, attributes and constants of a given interface, and also to define interface inheritance. There are some drawbacks to defining your interface using XPIDL. For example multiple inheritance is not supported. This means that a new interface cannot derive from more than one interface. Another limitation of interfaces in XPIDL is that method names have to be unique. Two methods with the same name that take different parameters are not allowed.

However, there are some major advantages that XPIDL provides. XPIDL allows you to generate type libraries, or typelibs, which are files with the extension .xpt. A type library is a binary representation of an interface or interfaces. When components are accessed from other languages than C++ they use the binary type library to access the interface, learn what methods it supports, and call those methods. This aspect of XPCOM is called XPConnect. XPConnect is the layer of XPCOM which provides access to XPCOM components from languages such as JavaScript. When a component is accessible from a language other than C++, such as JavaScript, its interface is said to be ‘reflected’ into that language. Every reflected interface must have a corresponding type library. Currently components can be implemented in C, C++, JavaScript, or Python, and there are efforts on the way to build XPCOM bindings for Ruby and Perl as well.

Object Ownership

Because components in XPCOM may implement any number of different interfaces, interfaces must be reference counted. This reference count is an integer inside the component that specifies how many clients are maintaining a reference to the component. This integer is incremented automatically when the client instantiates the component; over the course of the component's life, the reference count goes up and down, always staying above zero. At some point, all clients lose interest in the component, the reference count hits zero, and the component deletes itself.

XPCOM Identifiers

In addition to the IID interface identifier used to identify different interfaces, XPCOM uses two other very important identifiers to distinguish classes and components.

- Class Identifier (CID)
- Contract ID

A CID uniquely identifies a class or component in much the same way that an IID uniquely identifies an interface. The IID and CID are universally unique identifiers (UUID). A UUID is a unique, 128 bit number. A contract ID is a human readable string used to access and identify a component. Both CID and contract ID may be used to get a component from the component manager. Like a CID, a contract ID refers to an implementation rather than an interface, as an IID does. But a contract ID is not bound to any specific implementation, as the CID is, and is thus more general. Instead, a contract ID only specifies a given set of interfaces that it wants implemented, and any number of different CIDs may step in and fill that request.

Factories

Once code is broken up into components, client code typically uses the factory design pattern to create new instances of components. The factory design pattern is used to encapsulate object construction and initialization. The purpose of factories is to create objects without exposing clients to the implementations and initializations of those objects. The factory is the class that actually manages the creation of separate instances of a component for use. Another purpose for using the factory design pattern is that factories can easily handle singleton objects.

When clients use components, they typically instantiate a new object each time they need the functionality the component provides. This is the case when, for example, clients deal with files: each separate file is represented by a different object, and several file objects may be being used at any one time.

But there is also a kind of object known as a service, of which there is always only one copy (though there may be many services running at any one time). Each time a client wants to access the functionality provided by a service, they talk to the same instance of that service. When a user looks up a phone number in a company database, for example, probably that database is being represented by an "object" that is the same for all co-workers. If it weren't, the application would need to keep two copies of a large database in memory, for one thing, and there might also be inconsistencies between records as the copies diverged.

Providing this single point of access to functionality is what the singleton design pattern is for, and what services do in an application. If a factory creates an object that is supposed to be a singleton, then subsequent calls to the factory for the object should return the same object. Singleton objects are called Services in XPCOM.

In XPCOM, in addition to the component support and management, there are a number of services that help the developer write cross platform components. These services include a cross platform file abstraction which provides uniform and powerful access to files, directory services which maintain the location of application- and system-specific locations, memory management to ensure everyone uses the same memory allocator, and an event notification system that allows passing of simple messages.

Chapter 4

The evaluation platforms

Because of our limited resources and our goal to present a compact overview of our results and experiences, we decided to focus on three commercial EJB platforms:

- Bean Transactions 2.1
- PowerTier 6.54
- WebLogic 5.1.0

All three platforms are all based on the EJB specification 1.1 and have already been field-tested and offer several distinct features. Former versions of all three platforms have been used in projects at Siemens. So we had some experienced persons to work with.

4.1. Bean Transactions

The EJB platform Bean Transactions (BeanTA) is part of the OpenSEAS product family from Fujitsu Siemens Computers[9] (FSC). This product family consists of three products: Web Transactions, Biz Transactions and Bean Transactions. Web Transactions is a web server-based runtime for presenting any business object or information from an underlying application. It can format and convert the output data from the application to any suitable presentation form for the client (e.g. as HTML Website or Java Applet). Biz Transactions is an integration platform for defining business processes and integrating heterogeneous, existing applications into new systems. BeanTA is the EJB application server and platform that complete this product package. For our evaluation we only used the BeanTA platform.

The BeanTA platform is based on OpenUTM, a high end transaction monitor that offers sophisticated transaction management and monitoring. Because BeanTA

[9] http://www.fujitsu-siemens.com

does not include a CMP tool, a 3rd party tool called MPF/J from MicroDoc[10] is used as persistence framework for the CMP data mapping. The BeanTA software and the technical support through email and telephone have been provided by FSC during the whole time of our evaluation project.

4.2. PowerTier

The PowerTier application server is a product from Persistence Software[11]. This application server evolved from a former object-relational mapping tool. The version 6.54 of the software was provided to us by Persistence for our evaluation project. Technical support was provided via email.

4.3. WebLogic

The WebLogic Server is a product from the BEA Systems[12]. Because the version 6.x of the WebLogic Server did not support the EJB specification 1.1 correctly, we had to use the version 5.1.0 with service pack 10 for the evaluation. The software was downloaded from the homepage of BEA Systems. No technical support was provided during our project.

[10] http://www.microdoc.de

[11] http://www.persistence.com

[12] http://www.bea.com

Chapter 5

The EJB reference application

For testing and evaluating the different platforms we used a reference application that has been designed and built especially for this project. We tried to keep the application as simple as possible because our focus was on testing the platforms. On the other hand we wanted to have an application which included both simple and complex tasks, and which is of a domain where we had certain expert knowledge of. So we decided to write a bug managing application called BugTracker.

5.1. Architecture and Design

The BugTracker application is meant to be used to track and manage software bugs for different projects within a software company. In general software developers do not only have to car about bugs when developing a new software version. They are often faced with change requests that users want to be included in a future version of the software. Managing large software projects with many people is a very complex task for every project manager, especially when it comes to maintain the status of each bug or request in the software. Therefore usually this is done decentralized by an application which allows each authorized person to access a database to view or update the current status for a specific software bug or request. Because a software bug report is quite similar to a change request the term software issue will be used to refer to both.

The application will only be used within the scope of the software company. As a consequence a software issue can be submitted in two different ways. The software issue can be reported and submitted by a customer of the company or by some member of the project team. If a customer is the reporter of the software issue he will have to contact some person from the company (e.g. customer contact) who will enter the issue into the application’s database. The other way the issue will be reported and entered by the same person, who might be the software tester for that project. Therefore any access from external applications is not assumed within the scope for the BugTracker application. Any person that is related to the project is called Employee. The logical structure for the application is shown in.

For each person one or several addresses can be stored. Each address specifies a certain location related to the employee. This represents a typical data structure that is stored in company databases with personal records. The zip codes of the addresses are stored separately to serve as a pool for validating (i.e. to offer only valid values to choose from) and can be provided and maintained by an external system.

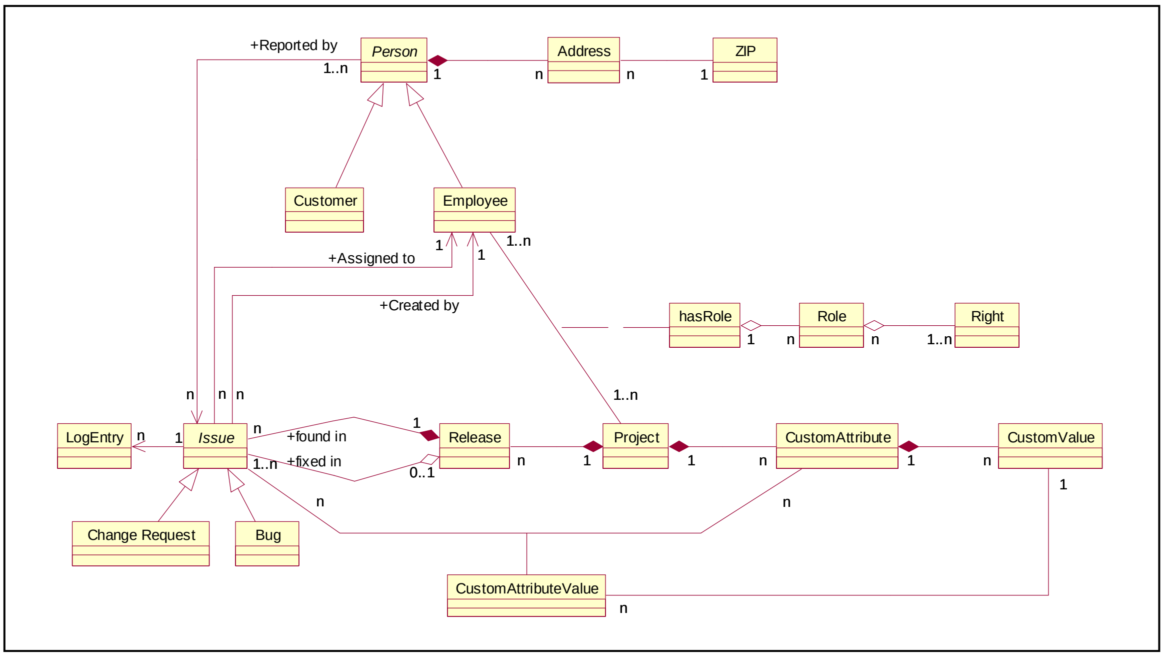


Figure 16 BugTracker Logical View

Employees working on projects can take several roles in the same project and also in several projects. Such roles can be customer contact, project contact (leader) or developer. The customer contact is responsible for entering the reported software issues into the system. The project contact is supervising a whole project, and assigns an open issue that has been released for implementing to a certain developer of the project. The developer is responsible for assigned issues and their implementation.

Each role has a set of rights stored separately to serve as a pool of available valid rights in the system. The rights specify which action can be performed by a specific role within the project, i.e. changing the state of an issue or the assignment to a developer. For each project one can define project specific customizable attributes as well as possible values for these attributes. These attributes belongs to each issue related to the project that has defined these attributes. So each issue is related to a set of *CustomAttributeValue* objects that contains the current values for these attributes of the issue.

As already mentioned above, there are two types of issues: bug reports and change requests. They only differ minimal, but are logically two different things. The main difference is the use of separate ID ranges for each type. An issue is assigned to a previously registered customer (who reported it), to a project (to which it belongs) and to a developer (who have to fix/implement it). It can also have a set of project specific attributes with currently assigned values. An issue once created will never be deleted; it just changes its state!

When a set of issues belonging to a project has passed the test phase, then a new software release for that project will be distributed. Each release should fix one software issue at least, otherwise it would not make sense to bring out a new release if there is no new feature or any bug fixed. Each bug or request is associated with the release it is found or requested in, and the release it is fixed in. Like any other application of this type every action taken on the issue is logged to provide a history of changes and tracing its life cycle. These log entries can not be modified after being inserted into the history.

Issue Life Cycle

A sample life cycle process for the issue could look like this: An issue is reported by a customer or project member and will get entered into the system. The project leader of the project will get notified of the arrival of a new issue. If the process regulation of the company requires special approval (voting by conference or email…) and/or there is need for more/additional information, the issue goes in the state “Voting” until a decision is taken: if the decision is negative, the issue is rejected and gets the state “Rejected”. Else the issue gets accepted (goes in the state “Accepted”) and finally gets properly analyzed. Using the results of the analysis, the project leader or any other company decision organ (e.g. Change

Control Board – CCB) can then decide if the issue will be finally fixed/implemented. If the decision is negative, the issue is rejected and gets the state “Rejected”, otherwise it goes in the “Ready” state – a pool of issues waiting for the assigned developer.

The developer takes an issue from the associated pool and opens it (the issue goes in the state “Open”) for fixing/implementing and marks it as implemented (state “Implemented”) when finished and ready for testing. This is the pool for the testing team who decides then if the issue is correctly fixed/implemented: if it is not, then the issue will be reopened – goes in the “Ready” state where it waits once again for the developer to open it (the associated developer can eventually be changed by e.g. the project leader), else the issue will be closed and ready to be included into the new release (by e.g. project leader).

5.2. Implementation

The application as described in the previous chapter has been implemented by me and my partner, using the Extreme Programming Method[13]. With this approach we could dynamically adapt our application design and had significantly shorter development iterations. This way we were also much more flexible during the implementation of the application when minor adaptations have to be done to the design and architecture.

In figure 17 the implementation view for the BugTracker application is shown. It shows the implementation classes for the related logical entities from figure 16. All logical classes have been either implemented as session beans or entity beans. The various design patterns and issues relevant to the architecture and implementation of the BugTracker application are discussed in the master thesis “Object-Oriented Analysis and Design with Enterprise JavaBeans” [Mue01].

[13] see [Bec00]

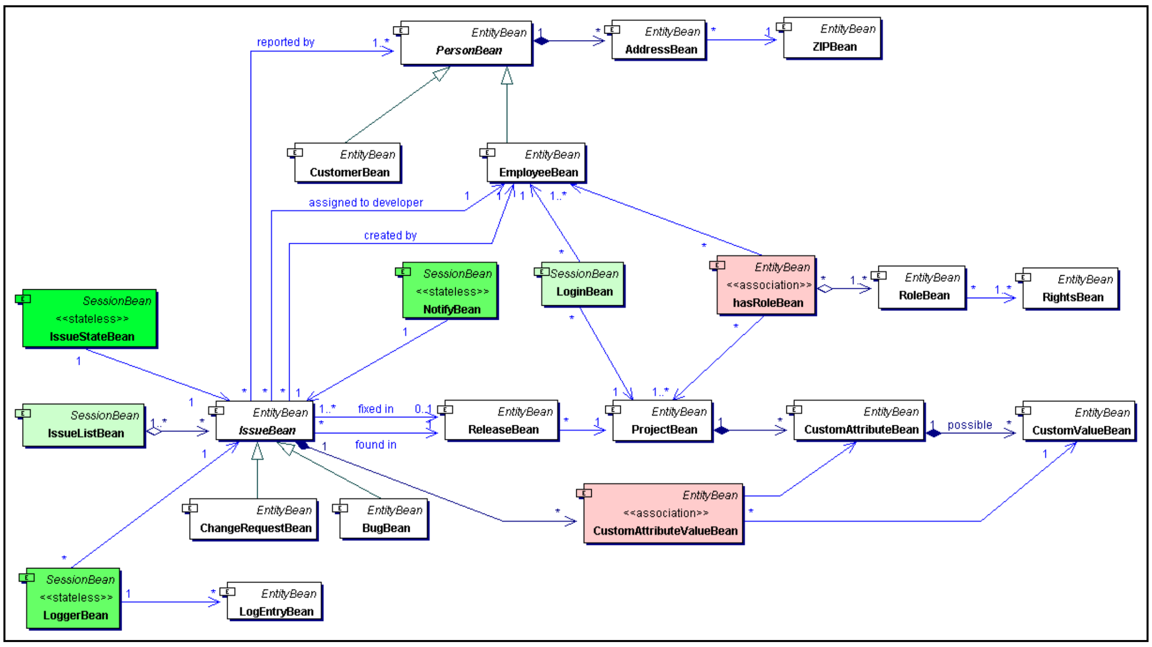


Figure 17 BugTacker Implementation View

Only the use cases required for our testing and evaluation have been implemented by us, because otherwise the implementation of the application would have taken too much time for realizing functionality that actually would never have been used. So we focused on several major use cases that we implemented and tested.

5.3. Application Client

The client for this application is written as stand-alone Java program with Java Swing as graphical toolkit. The architecture for the application client is shown in figure 18. The client is divided into two different layers to achieve a clean separation between graphical representation and business logic on the client side: the layer with the graphical user interface (GUI layer) and the layer with the client business logic (Client layer).

The GUI layer provides the graphical user interface and basic input verification. For example if a certain data format is expected for an input field in the user interface, the GUI layer is responsible for checking the format and giving feedback to the user if necessary. Other than that the GUI layer does not contain any other functionality. The whole graphic user interface is structured into several modules, so the client can be easily extended with additional modules and functionality.

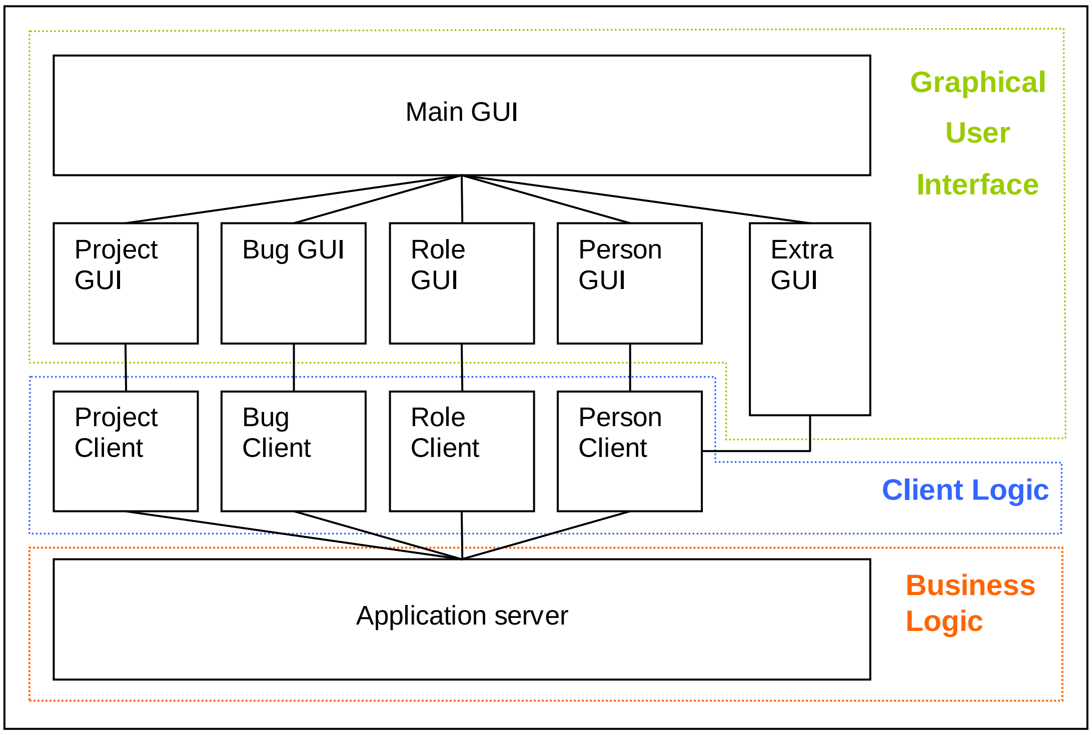


Figure 18 Architecture of application client

The *Main GUI* is the primary entry point for the graphical application. It provides the main window and controls the access to the other modules. Each module represents a certain use or test case. A module can be easily integrated into the application by adding a reference or entry into the Main GUI's menu list. Each of the GUI modules has access to a client class, which implements the business logic for that use or test case. These client classes encapsulate all the logic to connect to a remote server or to find a bean in the EJB container. Actually these clients can run without any graphical user interface. For example a small Java program can be written that access these clients to perform multiple tasks in the background, without the necessity to show a graphical user interface.

The client classes only contain the necessary code for finding and working with the bean components deployed on the application server. These bean components should provide all necessary business methods to perform the requested task. The client modules directly access the beans on the application server and invoke the business methods on the remote interface of the bean. During the performance tests for our evaluation all logged data will be stored in memory and written to a simple text file after the test run finishes.

Chapter 6

Specifying the evaluation environment

For the evaluation infrastructure architecture we decided to use a simple client server system consisting of one PC which acts as server computer and one PC which acts as client computer. At the same time both PCs were used as development workstations to modify and write the application source code.

6.1. Hardware configuration

We used the following hardware for our evaluation project:

- Server:
  Intel Pentium III, 800 MHz
  512 MB RAM
  10 GB Hard Disk
- Client:
  Intel Pentium, 266 MHz
  128 MB RAM
  5 GB Hard Disk
- Network connection:
  Cross over RJ-45 Patch cable

10 Mbit Ethernet cards

The server computer differs from the client computer by having much better equipment available. Both computers are connected using a simple cross over network cable. They are not connected to the Siemens company network to avoid any interference with the network traffic within the company and with the result for our performance tests. This isolated system configuration provides the necessary environment to run our tests equally on all platforms without any external disturbance.

6.2. Software configuration

The following Software was used during the project to implement and deploy the reference application:

- Server:
  MS Windows NT 4, Service Pack 5
  Informix Database Server
- Client:
  MS Windows NT 4, Service Pack 5
- Developer Tools:
  Togethersoft Together 5.02
  Borland JBuilder 4 Enterprise Edition

Together 5.02 has been used during the design and implementation of the server side application, JBuilder 4 has been used for writing the client application. The Together software supports the design of the application by providing integrated CASE tools that can be used to manage and generate basic source code that can be used for further implementation. If there are any changes in the design diagrams the related source code will automatically updated. The JBuilder provides certain tools that enhances and simplify the creation of EJB client applications.

6.2.1. Changing the database

First we intended to use the MS SQL Server 2000 as the underlying database for our tests. But during our attempts to integrate the MS SQL Server into the three platforms and the deployment process we had to realize that there are big difficulties when using the MS SQL Server in a Java based environment. Microsoft itself does not provide any JDBC driver for its database product. So we had to look for a third-party JDBC driver with free license that we can use for our test. Unfortunately each of the few free drivers we found, are limited in their functionality or configuration. Therefore we decided to switch to the Informix database which has its own JDBC driver. With a proper JDBC driver the integration of the database into the server platform was rather easy and simple for all three platforms.

Chapter 7

Evaluation

The evaluation of the three EJB application servers will consist of two parts. In the first part I will just compare and analyze some technical details and characteristics of each EJB platform. Therefore I will first list all criteria that I use for this evaluation to create a matrix which compare all three EJB platforms. In the second part I will do some performance testing based on different scenarios. The results for all three EJB platforms will be discussed and compared with each other.

7.1. Technical aspects

The technical evaluation criteria are listed in the following tables with a row for each criterion and five columns. The fist column contains a short abbreviation of the criterion. This abbreviation will be used in the final matrix to refer to this criterion. In the next column we will find a code letter that indicates what kind of evaluation will be performed on this criterion. The possible values for this column are:

- T - indicates that the criterion will be tested and/or measured within our test application
- A - indicates that the criterion will just be evaluated through a conceptual analysis of the platform

The third column includes a brief description of the test criterion. This description specifies what the criterion is about and what we have to consider when we evaluate this issue. The fourth column lists all the possible values that can be assigned to this criterion. For example if the criterion can only be evaluated with *yes* or *no*, these two values will be listed in this field. Finally in the last column we put a reference of the criterion to the related section in the EJB specification is placed. If the criterion is not covered by the EJB specification, this field in the last column will be left empty.

7.1.1. Basic Compliance to the EJB Specification

In this part of the list we will evaluate whether each platform complies with the EJB specification. According to the EJB specification each EJB platform has to provide a basic functionality. We will list these essential functionalities here within this section. Due to the fact that not every detail of the EJB component model has been documented in this specification, there can be misunderstandings in some parts of the specification. This is likely because that the authors of the specification have not been clear enough about some aspects of the implementation.

For example the life cycle of a Bean has to follow a certain life cycle model. I will ensure that these basic requirements are provided by the EJB platform vendor according to the EJB specification. If the criterion is fulfilled by the EJB platform the value for the criterion will be set to *ok* otherwise it will be set to *not ok*.

Criterion	Type	Description	Values	EJB 1.1
GenJNDI	T	Functionality of JNDI is verified by connecting to the EJB server via JNDI and retrieving a reference to the home object	ok not ok	5.2.1
GenHome	T	Functionality of the EJB Home interface is verified by testing methods for creating and removing instances	ok not ok	5.3
GenMeta	T	Functionality of the metadata facility is verified by obtaining a reference to an EJB MetaData object and testing this object's methods	ok not ok	5.3
GenObj	T	Functionality of the EJB Object interface implementation is verified by obtaining a reference to an EJB object and invoking all methods of the EJB Object interface on this object	ok not ok	5.4
GenSec	T	Basic security functionality of the EJB container is tested by invocation of methods with various security attributes	ok not ok	15.3 15.6
SesFulLifeCycle	T	The lifecycle of a stateful session bean is checked by a series of concurrent method invocations with and without a transaction context	ok not ok	6.6
SesLesLifeCycle	T	The lifecycle of a stateless session bean is checked by a series of concurrent method invocations with and without a transaction context	ok not ok	6.8
EntLifeCycle	T	The support of the EJB container for the lifecycle of entity beans is checked by a series of method invocations on a set of entity beans	ok not ok	9.1.4

7.1.2. Clustering and Load Balancing

Clustering is one way to make a system highly scalable. Clustering means to group several containers together in a way the load can be distributed among the different containers. With this mechanism they will appear as a single entity. It is not part of the EJB specification. So this is a proprietary feature of the platform and should be listed in the additional feature section. I decided to add it here though, because clustering does heavily influence the scalability.

Load Balancing is another very important concept for improving the scalability of a computer system. Usually a computer system does not have a constant rate of requests for processing. There are periods which require more performance than usual and periods which do not have the need for such a high demand of performance. During these peak times with a higher demand for performance the requests can be distributed to several machines so that each machine just has to process a part of the requests. There are different mechanisms to implement this feature; primarily based on the algorithm for the request distribution.

Criterion	Type	Description	Values	EJB 1.1
ClusAvail	A	This criterion indicates whether the EJB platform is supporting clustering or not (grouping several EJB containers together)	yes no	-
LoadBalAvail	A	This criterion indicates whether the EJB platform is supporting automatic load balancing or not (grouping several EJB containers together)	yes no	-

7.1.3. Activation / Passivation

The activation / passivation process is an interesting mechanism provided by the EJB model to deal with a large amount of Enterprise Beans, especially stateful Session Beans. In case the application server requires more resources it can acquire new resources by passivation of some beans in the container that have not been used for a specific time. During this passivation process the instance’s identity is automatically serialized and saved to a secondary storage and the bean is put back to the instance pool, where it can be used for the creation of other bean instances. If the passivated bean instance is needed the container will de-serialize the saved instance identity from the secondary storage and assign it to a new bean instance from the instance pool.

These criteria will test if the activation / passivation mechanism works properly and ensure that the EJB platform behaves in the way it is described in the EJB specification. Furthermore I will evaluate if the activation /passivation mechanism can be customized or not. Usually this happens by adjusting the number of beans when the EJB platform should start with passivation of inactive beans.

Criterion	Type	Description	Values	EJB 1.1
PassSesFul	T	This criterion indicates whether the passivation for stateful session beans works properly as described in the EJB specification	ok not ok	6.4.1
ActivSesFul	T	This criterion indicates whether the activation for stateful session beans works properly as described in the EJB specification	ok not ok	6.4.1
PassEntity	T	This criterion indicates whether the passivation for entity beans works properly as described in the EJB specification	ok not ok	9.1.5
ActivEntity	T	This criterion indicates whether the passivation for entity beans works properly as described in the EJB specification	ok not ok	9.1.5
PasActivCustom	A	This criterion indicates whether the activation passivation mechanism can be customized by the user	yes no	-

7.1.4. Transactions

The transaction management is a very crucial criterion of an application server. Especially business applications often require the ability of managing a large number of transactions at once. Transactions ensure that the manipulated data that the computer system relies on stays consistent and correct. Therefore operations on persistent data are usually running in a transactional context. The EJB specification is specifying 6 different categories of transactions for EJB components, which are listed in the following table. Additionally the *2PhaseCom* criterion will specify if the EJB platform supports transactional 2-phase commits or not. A 2-phase commit transaction is a transaction that spawns over two processes or systems. This feature is not part of the EJB specification, but it is a very powerful feature especially when the EJB platform is used in connections with legacy systems. Therefore I will check if the EJB platform provides this feature or not.

Criterion	Type	Description	Values	EJB 1.1
TxNotSupported	T	Functionality of the TX_NOT_SUPPORTED transactional attribute is verified	ok not ok	11.4.1 11.6.2
TxSupports	T	Functionality of the TX_SUPPORTS transactional attribute is verified	ok not ok	11.4.1 11.6.2
TxRequired	T	Functionality of the TX_REQUIRED transactional attribute is verified	ok not ok	11.4.1 11.6.2
TxRequiresNew	T	Functionality of the TX_REQUIRES_NEW transactional attribute is verified	ok not ok	11.4.1 11.6.2
TxMandatory	T	Functionality of the TX_MANDATORY transactional attribute is verified	ok not ok	11.4.1 11.6.2
TxNever	T	Functionality of the TX_NEVER transactional attribute is verified	ok not ok	11.4.1 11.6.2
2PhaseCom	A	This criterion indicates whether the EJB platform supports 2-phase commits of transactions or not	yes no	-

7.1.5. Handling of exceptions

Besides the functionalities that the different platforms should have, it is also very important to know how robust a platform is. One main concept and feature of the Java programming language is the use and handling of exceptions. This concept can avoid many problems that occur when something is going wrong. So within this section I will evaluate how exceptions are handled for different test scenarios. The responsibilities of an EJB platform provider are described in the EJB specification. Furthermore I will check if the EJB platform supports recovering of stateful session beans or not. Stateful session beans can sometimes contain important information that would be lost after a server crash, because session beans are not persistent components according to the EJB specification. But some platforms offer sophisticated caching for those stateful session beans, so that the information can be recovered after the EJB server crashed. The more robust a server is, the better availability can be guaranteed.

Criterion	Type	Description	Values	EJB 1.1
EjbExepGen	T	General exception handling support of the EJB container is checked by a series of method invocations causing various kinds of exceptions	ok not ok	-
EjbExepEntity	T	Exception handling support of the EJB container is checked by a series of method invocations causing various kinds of exceptions. This test is performed for an entity bean	ok not ok	12.3

EjbExepSesFul	T	Exception handling support of the EJB container is checked by a series of method invocations causing various kinds of exceptions. This test is performed for an stateful session bean	ok not ok	12.3
EjbExepSesLes	T	Exception handling support of the EJB container is checked by a series of method invocations causing various kinds of exceptions. This test is performed for an stateless session bean	ok not ok	12.3
RecSesBean	A	This criterion indicates whether the EJB platform supports the recovery of stateful session beans after a server crash or not	yes no	-

7.1.6. Persistence

The persistence mechanism in the EJB model allows two ways of making data persistent. One is the concept of Container Managed Persistence (CMP), where the EJB container is responsible for making the data persistent to any kind of storage system, e.g. a database or a legacy system. The alternative way is to implement the persistence mechanism in the Entity Bean itself (BMP). So the responsibility for the data persistence lies within the scope of the Bean provider. Because the first approach is the one which will be usually used and provided by the EJB platform I will check the proper working of the CMP mechanism within this section. Another criterion that will be evaluated is the data mapping mechanism of the EJB platform. Some EJB platforms use their own integrated mapping mechanism, other EJB platform use a 3rd party product. This can be a relevant issue when the data mapping should be replaced by another mechanism. A 3rd party product would offer more flexibility but might be not that well integrated into the EJB platform.

Criterion	Type	Description	Values	EJB 1.1
PerCmpFindKey	T	The functionality of the *findByPrimaryKey* method provided by the EJB container for CMP entity beans will be verified	ok not ok	9.4
PerCmp	T	The correct and proper working of the CMP mechanism is verified by a series of method invocations on CMP entity beans	ok not ok	9.3
PerMapMech	A	This criterion indicates how the data is mapped to the data storage. Does the EJB platform use its own integrated mapping tool or a 3rd party product?	Integrated 3rd party	-

7.1.7. Security

One aspect that the bean provider usually does not have to care about when implementing the bean code is the security management. They security can be managed by the application server and the security policy can be set up at deploy time. If the EJB platform supports Hot Security Administration the security settings can be changed and applied while the server is running. The EJB specification does not require Hot Security Administration, but it is a very comfortable feature, because the server does not have to be restarted when changing the security settings. The *SecurAdminLevel* criterion specifies the level of security that the EJB platform offers. Unfortunately this issue is not well specified in the EJB specification, so that usually security settings are applied to a whole bean component. So EJB platforms offer the support to make more fine grained settings for each bean component, so that the custom security settings can be applied to single methods of each bean component.

Criterion	Type	Description	Values	EJB 1.1
SecurMech	T	The correct and proper working of the security mechanism provided by the EJB container will be verified by a series of method invocations with different security roles	ok not ok	15.6
SecurAdminLevel	A	This criterion indicates the level of security that can be used for the bean components	bean method	-
HotSecurAdmin	A	This criterion indicates if it is possible to change and apply the security settings and rights while the server is running	yes no	-

7.1.8. Availability

High availability is a precondition for every successful e-commerce application. Expressions like “24x7 availability” are commonly used and always asked for among all enterprises that have to deal with e-commerce. So a criterion that has to be evaluated, too, is the grade of availability that each application server provides by using different techniques. Hot Deployment is a feature that allows bean components to be deployed into the EJB container without restarting the server. The automatic restart of the server after a server crash can be very useful, too. Another interesting feature is the use of multiple Java Virtual Machines (JVM). A EJB platform that is based on this feature will run each EJB container in a separate JVM, so the crash of one EJB container would not influence the other EJB containers. This can significantly increase the availability of a working system.

Criterion	Type	Description	Values	EJB 1.1
HotDeploy	A	This criterion indicates if it is possible to deploy bean components into the EJB container while the server is running	yes no	-
AutoRestart	A	This criterion indicates if the EJB platform supports automatic restarting of the server after a servercrash	yes no	-
JvmMech	A	This criterion indicates if the EJB platform uses a single JVM for all EJB containers or not	SingleJVM MultiJVM	-

7.1.9. General Specifications

The following table lists some essential basic information about each EJB platform, like version number and provider of the EJB platform. The necessary version of the Java Development Kit (JDK) and the minimum system hardware requirements for running the server are listed, too. If no information about the minimum requirement can be provided *n/a* (not available) will be used. The criterion *SubEjbVer* specifies which version of the EJB specification is supported by the EJB platform. The possible values for this criterion are only 1.0 or 1.1. The field *Remarks* contains some information about important issue that the user should know about when working with the platform.

Criterion	Type	Description	Values	EJB 1.1
ProductVer	A	This criterion indicates the version of the EJB platform that we used for our evaluation.	-	-
Provider	A	This criterion specifies the provider of the EJB platform		
ReqJdkVer	A	This criterion indicates the version of the JDK required for using the EJB platform	1.2 1.3	-
ReqCpu	A	This criterion indicates the required CPU clock frequency for running the EJB server	-	-
ReqRam	A	This criterion indicates the required amount of system memory needed for running the EJB server	-	
SupEjbVer	A	This criterion indicates the version of the EJB specification that is supported by the EJB platform	1.0 1.1	-
Remarks	A	This field specifies information about different issues that occur up during the evaluation	-	

7.1.10. Supported OS platforms

This section lists all the OS platforms that are supported by the EJB platform.

Criterion	Type	Description	Values	EJB 1.1
Windows NT	A	This criterion indicates support for Windows NT	yes no	-
Windows 2000	A	This criterion indicates support for Windows 2000	yes no	-
Sun Solaris	A	This criterion indicates support for Sun Solaris	yes no	-
HP-UX	A	This criterion indicates support for HP-UX	yes no	-
IBM AIX	A	This criterion indicates support for IBM AIX	yes no	-
Linux	A	This criterion indicates support for Linux	yes no	-
Reliant UNIX	A	This criterion indicates support for Reliant UNIX	yes no	-
OS/390	A	This criterion indicates support for OS/390	yes no	-
OS/400	A	This criterion indicates support for OS/400	yes no	-
TRU64 UNIX	A	This criterion indicates support for TRU64 UNIX	yes no	-
IRIX	A	This criterion indicates support for IRIX	yes no	-

7.1.11. Usability

In this section we will evaluate how user friendly the administration of the application server is for the user. Furthermore which support the application server provides for the developing or testing of an EJB application. Finally we will rate the installation and set up routine of each application server, too.

Criterion	Type	Description	Values	EJB 1.1
UsaAdminTools	T	This criterion indicates the grade of usability for the administration tools provided by the EJB platform	good sufficient bad	-
UsaDevTools	T	This criterion indicates the grade of usability for the administration tools provided by the EJB platform	good sufficient bad	-
UsaInstall	T	This criterion indicates the grade of usability for the installation of the EJB platform	good sufficient bad	-
UsaSerDebug	T	This criterion indicates whether the EJB platform offers support for debugging the EJB server or not	yes no	-

7.2. Test scenarios

We will have three different scenarios for our evaluation project. Each scenario will be discussed in the following subchapters. These scenarios can be divided into two groups. The first one is just testing the Bug Tracker application itself. After that is done we can run our load tests against the server and measure the time it takes to complete the different method invocations. The database is restored to its initial state each time before testing one scenario.

7.2.1. Testing the basic functionality of the Bug Tracker application

With this series of tests we will ensure that the Bug Tracker application is functioning properly. Besides the different method invocations according to the use cases will verify that the server is working in the intended way. Therefore we will simulate normal use cases through the GUI of the Bug Tracker application and check then if the data and inputs were processed and stored in the right way. The right functionality is a requirement before running the load tests against the server.

7.2.2. Creation and manipulation of person records (entity beans)

This test scenario will simulate some load tests for the use with entity beans. First we will create a series of data records, measuring the time it takes to insert the data record into the database. To get more accurate time intervals we will perform several method invocations in a loop and then calculate the average time for each method invocation. For example 5 entity beans are created in a loop and the time it takes to complete that loop will be used to calculate the average time for each creation of an entity bean. Next we will measure the time it takes to get a reference to a certain entity bean identified by its primary key.

After these two tests we will compare the efficiency of retrieving data from the entity bean through single method invocations and through the use of a value object on each server. The single method invocations will require several network calls, while the value object will only require one. The time it takes to get the whole data record is measured in this test. Finally each entity bean will be removed from the database by removing the entity bean. The time it takes to complete this method invocation is measured and logged.

7.2.3. Creation and manipulation of data by using a stateful session bean

In this scenario we will measure the time it takes to create a project data record through the use of a stateful session bean. The time that is measured includes the creation of a stateful session object and its assignment to the client, the time it takes to create a project entity bean and the time to update the necessary state information in the stateful session bean. Similar to the former test process we do create several data records at once and then calculate the average time for each session bean. Furthermore we will check the time it takes for the client to get a value object for the desired data record in the database via the stateful session bean. How long does it take from the method invocation on the client side to receive the value object from the server?

Chapter 8

Results and conclusions

This chapter contains the results for our evaluation project. In the following chapter the results for evaluating the technical aspects of all three evaluation platforms are shown in a matrix. The next chapter presents some figures and statistics for the time measurement during the performance tests. The BugTracker application was used for both, the performance tests and the tests to verify several technical aspects.

8.1. Technical Aspects

The following table shows the results for the technical evaluation for the three EJB platforms in the project:

Criterion	PowerTier	WebLogic	BeanTA
Basic compliance to the EJB specification			
GenJNDI	ok	ok	ok
GenHome	ok	ok	ok
GenMeta	ok	ok	ok
GenObj	ok	ok	ok
GenSec	ok	ok	ok
SesFulLifeCycle	ok	ok	ok
SesLesLifeCycle	ok	ok	ok
EntLifeCycle	ok	ok	ok
Clustering and Load balancing			
ClusAvail	no	no	yes
LoadBalAvail	no	yes	yes
Activation / Passivation			
PassSesFul	ok	ok	ok
ActivSesFul	ok	ok	ok

PassEntity	ok	ok	ok
ActivEntity	ok	ok	ok
PasActivCustom	no	yes	yes
Transactions			
TxNotSupported	ok	ok	ok
TxSupports	ok	ok	ok
TxRequired	ok	ok	ok
TxRequiresNew	ok	ok	ok
TxMandatory	ok	ok	ok
TxNever	ok	ok	ok
2PhaseCom	no	no	yes
Handling of exceptions			
EjbExepGen	ok	ok	ok
EjbExepEntity	ok	ok	ok
EjbExepSesFul	ok	ok	ok
EjbExepSesLes	ok	ok	ok
RecSesBean	no	no	yes
Persistence			
PerCmpFindKey	ok	ok	ok
PerCmp	ok	ok	ok
PerMapMech	Integrated	Integrated	3rd party
Security			
SecurMech	ok	ok	ok
SecurAdminLevel	bean	method	method
HotSecurAdmin	no	no	yes
Availability			
HotDeploy	no	yes	yes
AutoRestart	no	no	yes
JvmMech	SingleJVM	SingleJVM	MultiJVM
General specification			
ProductVer	6.54	5.1.0 SP10	2.0A
Provider	Persistence	BEA Systems	Fujitsu Siemens
ReqJdkVer	1.3	1.3	1.3
ReqCpu	233 MHz	n/a	250 MHz
ReqRam	128 MB	64 MB	256 MB
SupEjbVer	1.1	1.1	1.1

Remarks	propriety O/R mapping	-	-
Supported OS platforms			
Windows NT	yes	yes	yes
Windows 2000	yes	yes	yes
Sun Solaris	yes	yes	yes
HP-UX	yes	yes	yes
IBM AIX	yes	yes	no
Linux	yes	yes	yes
Reliant UNIX	no	yes	yes
OS/390	no	yes	no
OS/400	no	yes	no
TRU64 UNIX	no	yes	no
IRIX	no	yes	no
Usability			
UsaAdminTools	sufficient	good	good
UsaDevTools	good	sufficient	good
UsaInstall	sufficient	good	good
UsaSerDebug	yes	yes	yes

8.2. Performance Tests

According to the test scenarios described in chapter 7 the BugTracker application was used to measure the time it took to perform the requested tasks. For the performance tests with entity beans I chose to do three test series with different number of entity beans. Due to the hardware limitations the first test series have been done with 2000 beans, the second one with 4000 beans and the third one with 10000 beans. These numbers should represent realistic figures for a simple real world scenario. Comparing the average time for each test series should allow me to draw conclusions for the scalability for each EJB platform. In the following subchapter only the test series with 2000 and 10000 beans are illustrated. The test series with 4000 beans is discussed in the overview figure. Because the number of stateful session beans will in general be much lower than the number active entity beans, the performance tests for stateful session bean have been done in two test series with 1000 and 2000 beans.

8.2.1. Creation of entity beans

In these test series a given amount of CMP entity beans are created through in a loop in the client program. Figure 19 shows the time needed for the creation of each bean in the test series with 2000 beans. As we can see from this figure the BeanTA platform requires a lot more time to insert a new data record into the database. The cause for this result might be the overhead generated by using a 3rd party product for the data mapping. This support requires a more complex integration of the mapping tool than in case of an integrated mapping tool, where a much better optimization can be achieved. The peaks in each data row are indicators for the caching algorithm (including activation/passivation policy) of each platform. WebLogic seems to have a better caching mechanism than PowerTier or BeanTA.

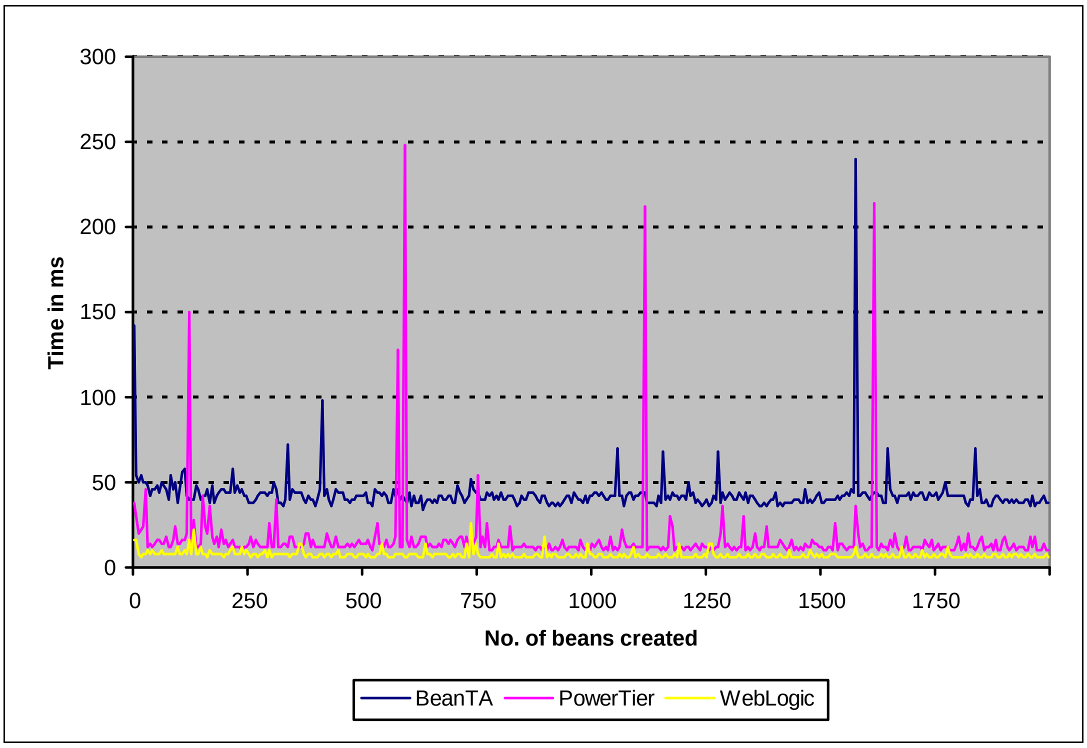


Figure 19 Creation of entity beans (2000 records)

The next figure 20 illustrates the test series with 10000 entity beans. In this figure the scalability for each platform can be seen. While the PowerTier graph shows an obvious upward trend, the WebLogic and BeanTA graph do not change much. However the BeanTA graph has much more peak values than the other two graphs. Figure 21 shows a summary of all three test series where we can see that WebLogic scales much better than the other two EJB platforms.

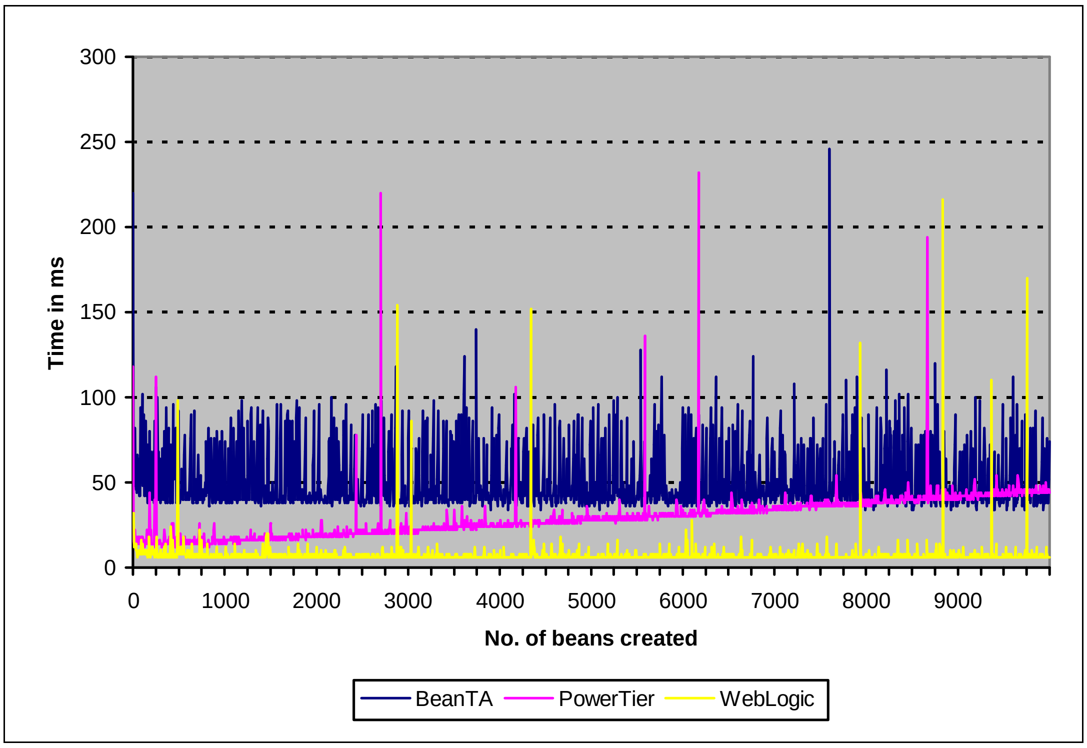
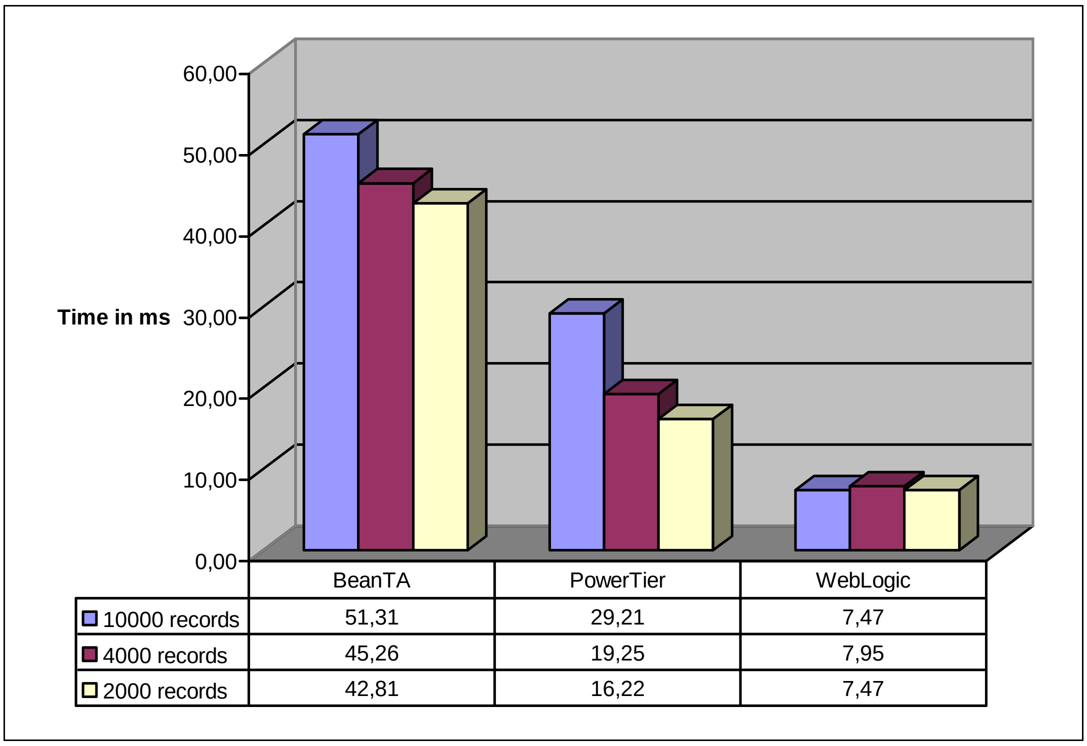


Figure 20 Creation of entity beans (10000 records)



	BeanTA	PowerTier	WebLogic
🟦 10000 records	51,31	29,21	7,47
🟥 4000 records	45,26	19,25	7,95
🟨 2000 records	42,81	16,22	7,47

Figure 21 Average time for creating an entity bean

8.2.2. Finding entity beans

The following test series show the time it took to find a specific entity bean depending on the number of existing entity beans in the EJB container. Figure 22 shows the figure for the test series with 2000 beans. Here again the BeanTA platform requires more time to find a certain entity bean due to the more complex integration of the data mapping tool.

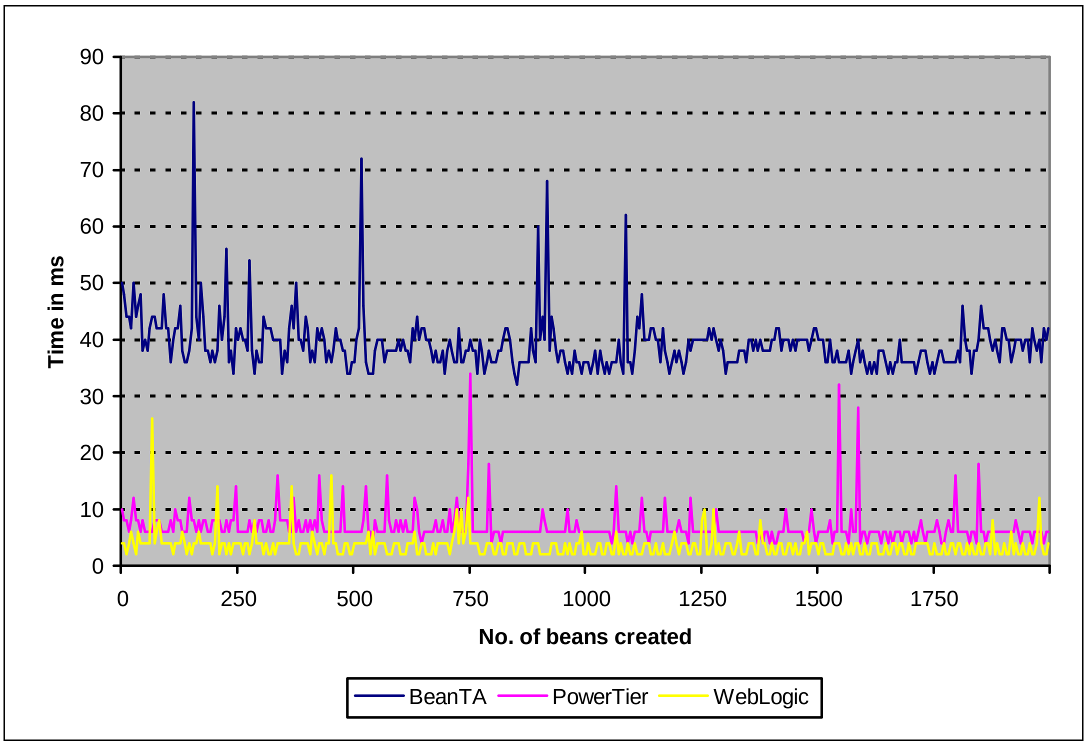


Figure 22 Finding entity bean (2000 records)

In figure 23 the test series we can see that the BeanTA and PowerTier platforms do both need increasingly more time to find an entity bean when the number of existing entity beans rises. While again the BeanTA graph fluctuate a lot the graphs for the WebLogic and PowerTier platforms are quite straight. In figure 24 the average time is shown for all three platforms and a summary for all three test series. The conclusions are here the same as in test series with 10000 beans: WebLogic achieves to keep a constant time for performing the search in all three test series whereas the performance of the other two platforms is depending on the amount of existing entity beans.

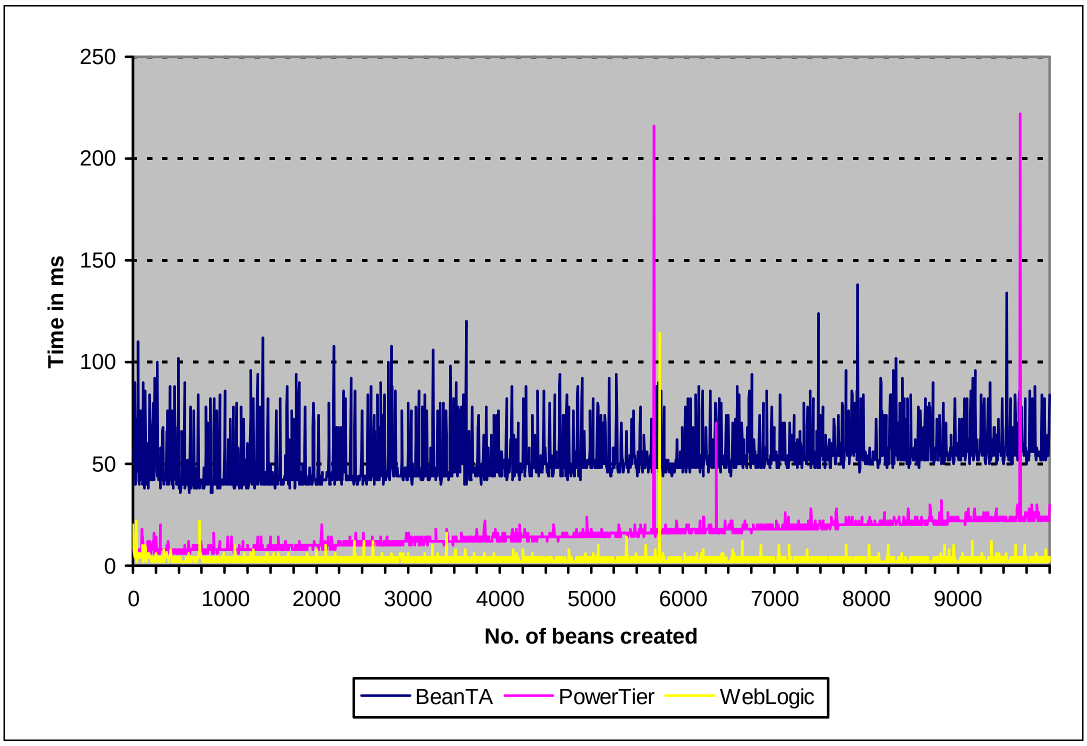
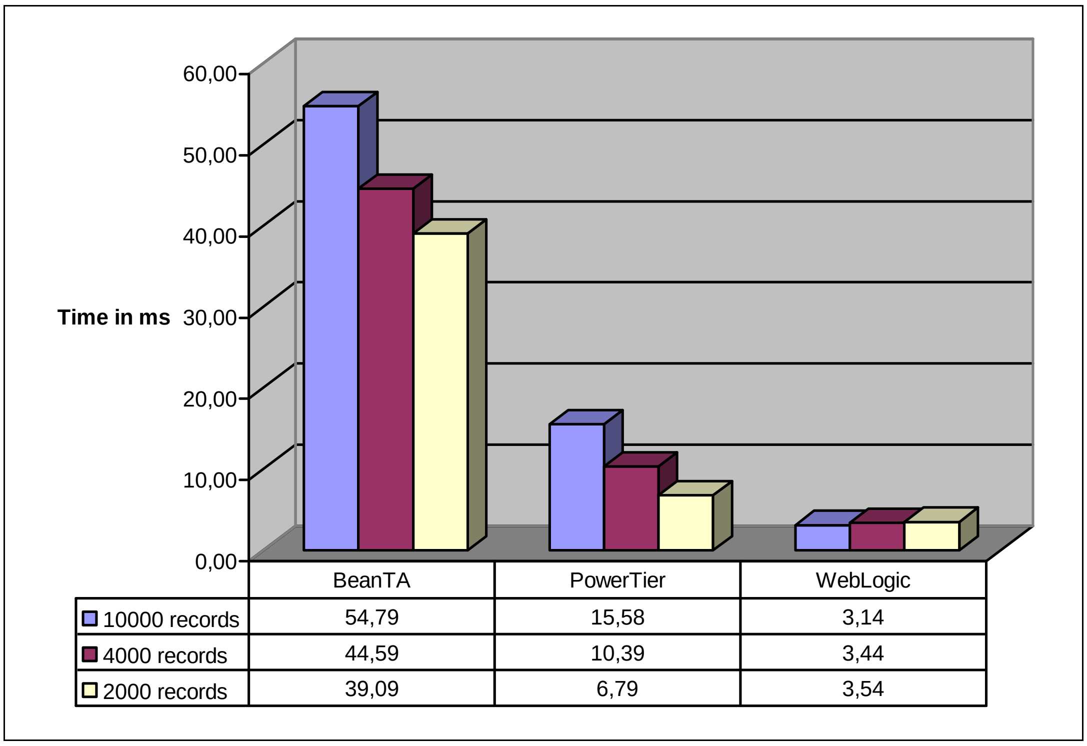


Figure 23 Finding entity beans (10000 records)



	BeanTA	PowerTier	WebLogic
🟦 10000 records	54,79	15,58	3,14
🟥 4000 records	44,59	10,39	3,44
🟨 2000 records	39,09	6,79	3,54

Figure 24 Average time for finding an entity bean

8.2.3. Removing entity beans

In these test series the performance of deleting data from the database is evaluated. An entity bean represents a data record in the data storage, so removing an entity bean will also delete the correspondent data record in the data storage. Figure 25 shows the illustration for the test series with 2000 existing entity beans in the EJB container. In this test the BeanTA performance is much better than in the other test series with entity beans.

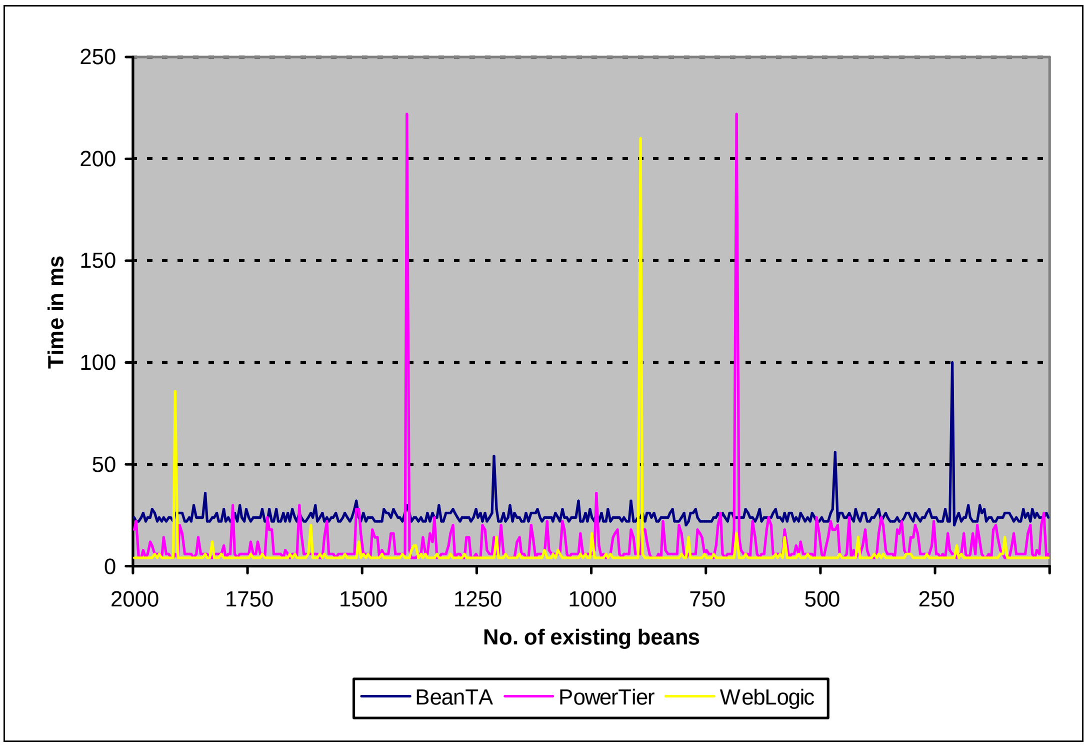


Figure 25 Removing entity beans (2000 records)

The next figure 26 shows the results for the test series with 10000 entity beans. The illustration shows some common characteristics among all three platforms:

- All three platforms need less time to delete an entity bean when the number of existing entity beans is decreasing, too.
- All three graphs seem to fluctuate a lot, especially at the beginning of the test. These masses of peaks indicate that a lot of caching is done in the background during the deletion of the entity bean.
- At the beginning of the test the performance of all three platforms is almost the same. But PowerTier and WebLogic scale down better than the BeanTA platform.

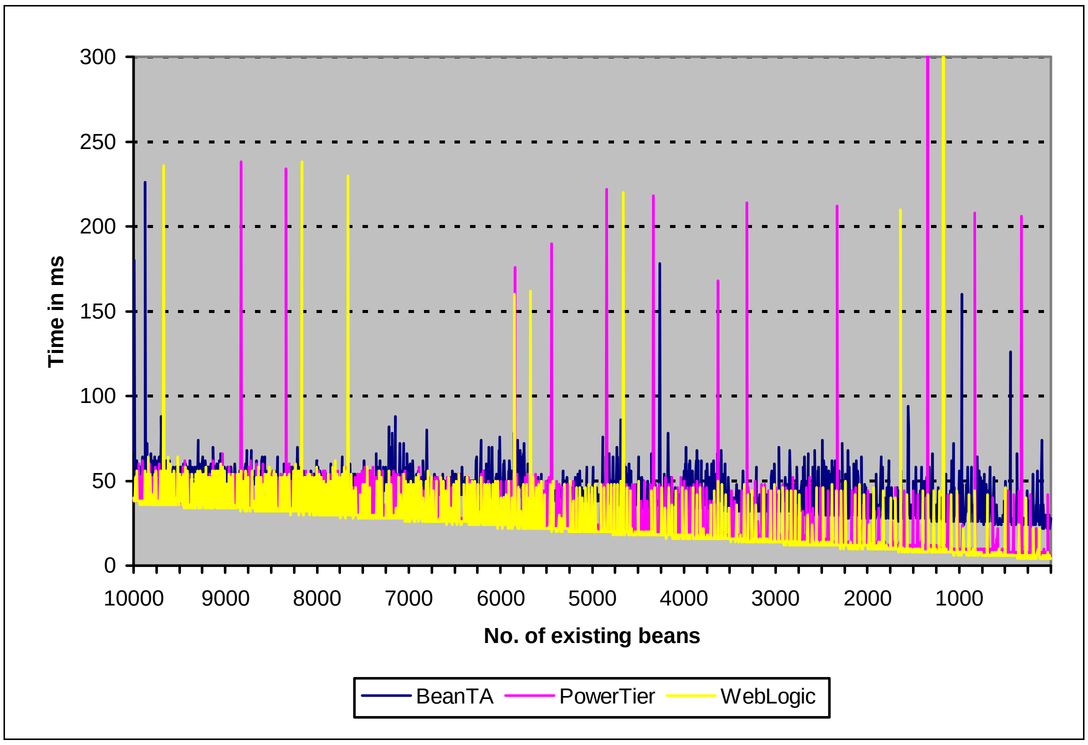
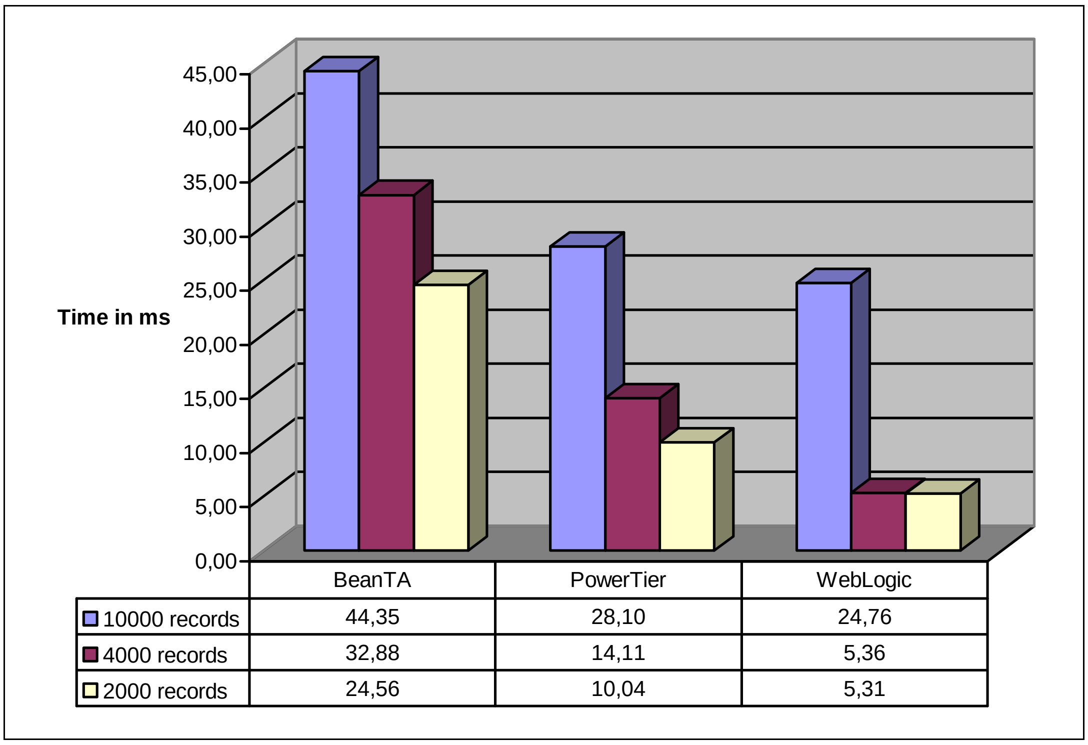


Figure 26 Removing entity beans (10000 records)

Figure 27 shows the average time for the three test series. The performance of all three platforms is much closer than in the other tests.

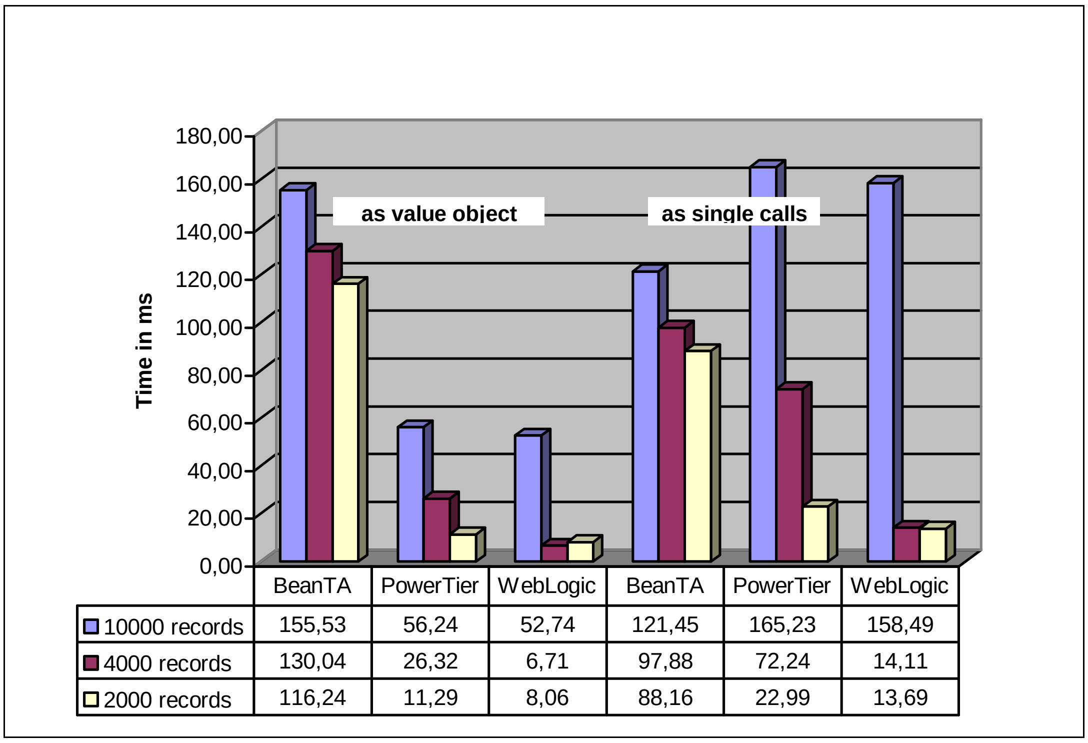


	BeanTA	PowerTier	WebLogic
🟦 10000 records	44,35	28,10	24,76
🟥 4000 records	32,88	14,11	5,36
🟨 2000 records	24,56	10,04	5,31

Figure 27 Average time for removing an entity bean

8.2.4. Retrieving data from an entity bean

Figure 28 is showing the average times for retrieving data from an entity bean during each test series. In this test we compare two different concepts of getting the data in a distributed application. In one case we use a value object to encapsulate all data. So the network calls are minimized to one single client request and one server reply. This concept provides some overhead in managing and creation of the value object. In the other case all data is retrieved through a separate call from the client requesting the specified data field. This might end up in a lot of network calls. So the value object seems to be the better solution concept for strongly fragmented data that is changed quite often, whereas the single call concept is a better alternative for very static data, where only partial data information is requested. In our test with entity beans we used data records that contain four data elements. So the value object in our test has to encapsulate four data fields. In the case of single calls we have to send four network calls to receive the requested data. The time in our test is measured from the moment when the request is sent to the server till the moment where we have the proper value object received from the server or the properties filled with the data from the single calls approach.



	BeanTA	PowerTier	WebLogic	BeanTA	PowerTier	WebLogic
🟦 10000 records	155,53	56,24	52,74	121,45	165,23	158,49
🟥 4000 records	130,04	26,32	6,71	97,88	72,24	14,11
🟨 2000 records	116,24	11,29	8,06	88,16	22,99	13,69

Figure 28 Average time it takes to retrieve data from an entity bean

The interesting result of our test is that on the PowerTier and the WebLogic platform the value object approach provides a better performance, whereas on the BeanTA platform the single calls are much faster than retrieving the data via the value object approach. It seems that the performance on BeanTA is much better without the overhead of handling the value object. Another interesting result of this test is that BeanTA is much faster than the other two platforms when it comes to retrieve the data via single calls under high load. In the test series with 10000 entity beans BeanTA is providing better performance than the other two EJB platforms. WebLogic scales very well in the lower level, but in the higher level it seems to have some problems with the scalability.

8.2.5. Creating stateful session beans

In the next test series the creation of a data record via stateful session beans is evaluated and how the performance for this task is depending on the number of already existing session beans in the EJB container. Figure 29 shows the first test series with the creation of 1000 session beans.

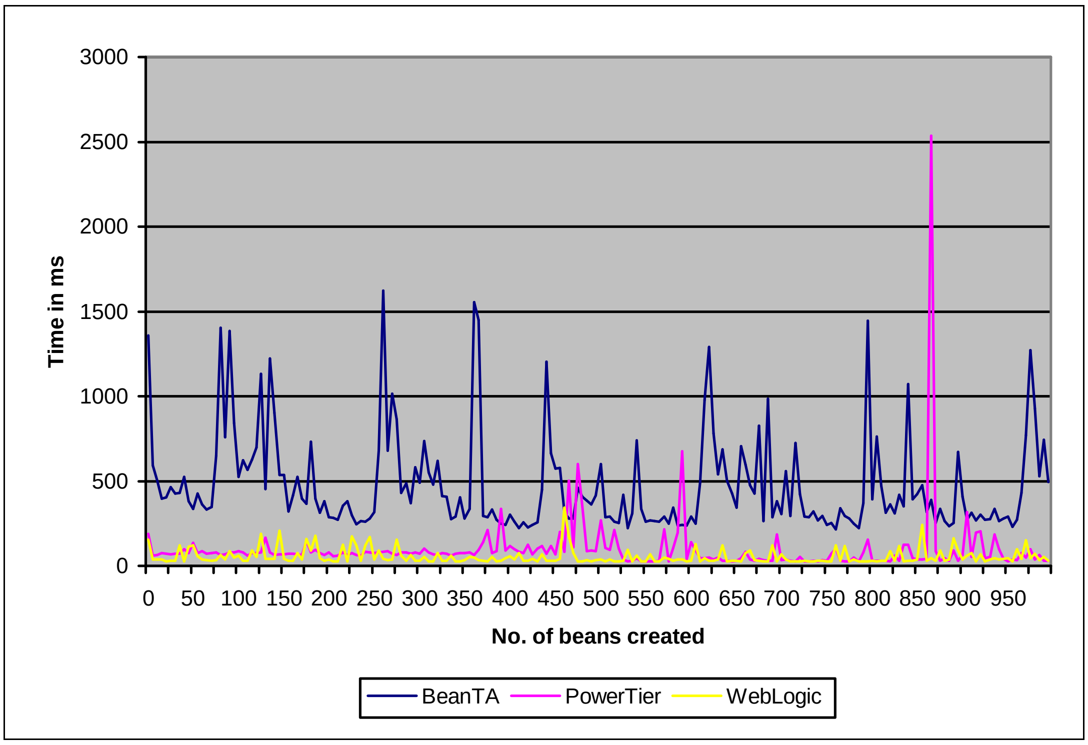


Figure 29 Creating a data record via stateful session bean (1000 beans)

The performance of the WebLogic and PowerTier platforms is again much better than the performance of the BeanTA platform. Here again the integration of a 3rd party mapping tool seems to be the cause for this performance gap. Another aspect might have influence on this result, too. Due to the fact that it has evolved from a high end transaction monitor, the BeanTA platform also incorporates a very sophisticated transaction management and monitoring. In general all three platforms need more time to generate a data record via stateful session bean than directly via entity bean, because the creation via session bean requires more internal status information, more method invocations and causes a larger overhead to handle.

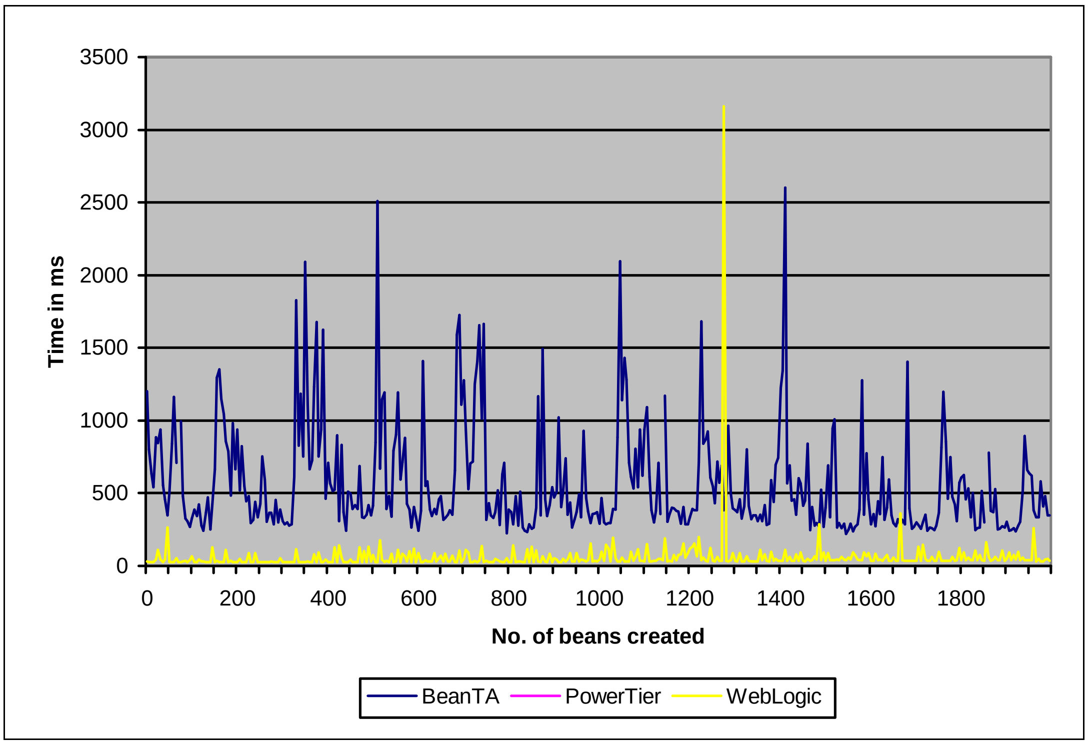
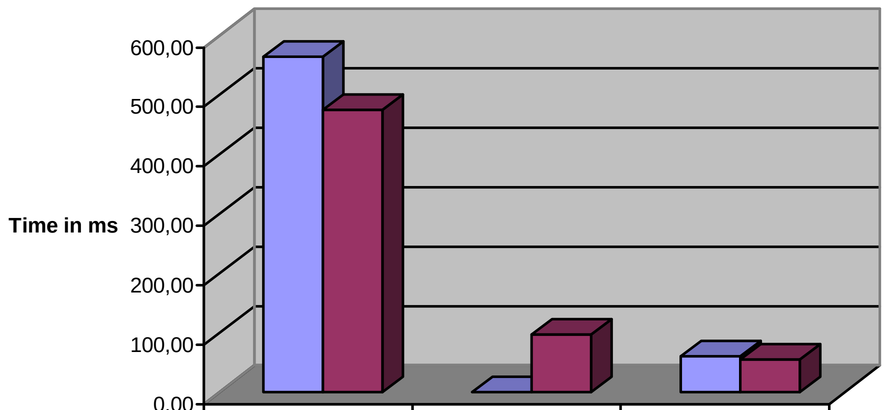


Figure 30 Creating stateful session beans (2000 beans)

Figure 30 shows the test series for 2000 session beans for the BeanTA and WebLogic platform. The PowerTier graph is missing because the data for the PowerTier graph could not be retrieved due to a bug in the PowerTier platform which causes the server to crash undetermined before it can create all 2000 session beans. The bug was fixed only after the end of the testing phase of this project. That is why the PowerTier data could not be considered anymore for this test series. A summary of those two test series is shown in figure 31. The number of existing stateful session beans does not seem to influence the time it take to create a data record in the database as we can see from the graphs in figure 29 and figure 30.

These graphs do not show an upward trend or pattern. The higher values for the average time in the summary figure are caused by the numerous peak values in the graph. The overhead caused by the more complex and linked handling for the data record seems to outweigh the cost caused by the growing number of instances.



	BeanTA	PowerTier	WebLogic
▨ 2000 beans	564,89	0,00	61,04
▨ 1000 beans	475,90	97,50	55,17

Figure 31 Average time for creation of a data record via stateful session bean

The results of these test series encourage the use of entity beans on the client side. Some design patterns for EJB application recommend that the client should only work with session beans, because that would provide a safer and clearer approach for an EJB application. The resulting overhead in forwarding a method call would be not that bad. But the results of these test series show that this would happen at the cost of reduced application performance. Creating data records directly with the use of entity beans is 10 times faster than using stateful session beans to do this on the server side. In an application where data records are frequently added and modified, this design decision can result in a huge performance gap.

8.2.6. Retrieving data from a stateful session bean

In these test series a value object is retrieved via the use of stateful session beans. The session bean acts on behalf of the client to get the requested data which is encapsulated in a value object. Figure 32 shows the result of the test series with 1000 session beans. The graphs for these test series seem to be within a certain bandwidth. The peak values that we can see in the graphs for the BeanTA and PowerTier platforms seem to occur quite randomly.

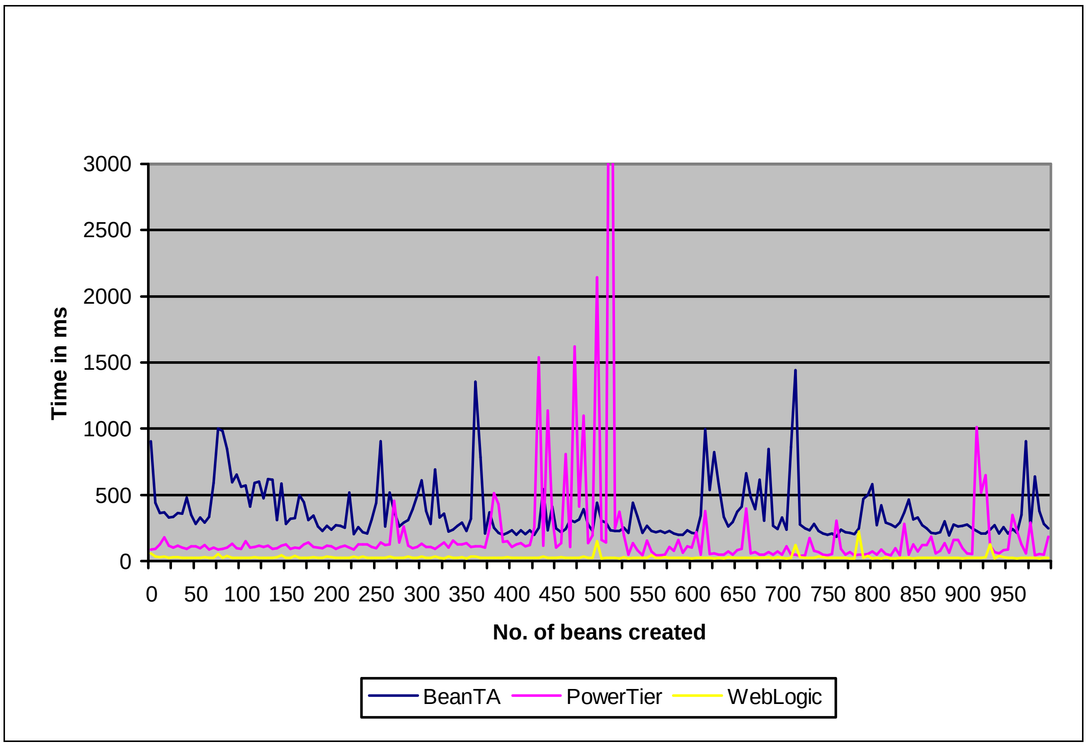


Figure 32 Retrieving a value object from the stateful session bean (1000 beans)

The following figure 33 shows the graphical illustration for the test series with 2000 session beans. Due to the reason mentioned in the last subchapter, the graph for the PowerTier platform is missing in this figure, too. The graphs here do not indicate any upward tendency or pattern. While the graph for the WebLogic platform stays quite constant the graph for the BeanTA platform shows a more random behavior with many peak values. In figure 34 a summary of the two test series is shown and the average time value for each test series. When we compare the average value in this figure with the average value, we can see again that the direct access to the entity bean does make a big difference in the performance for this task.

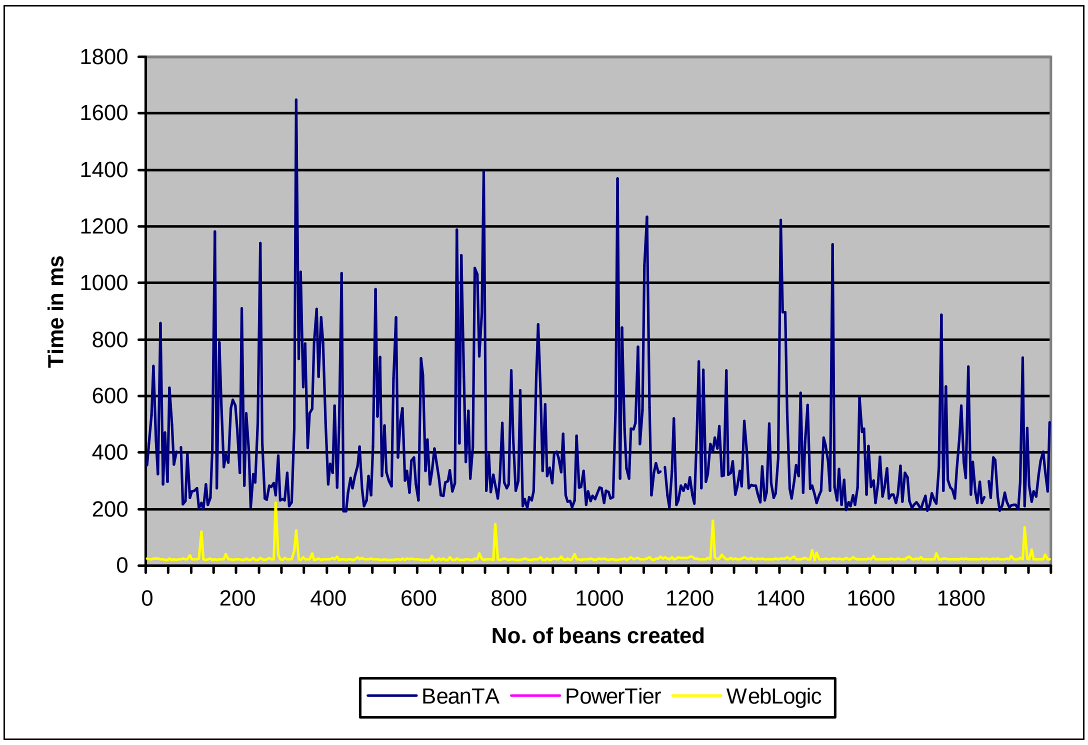
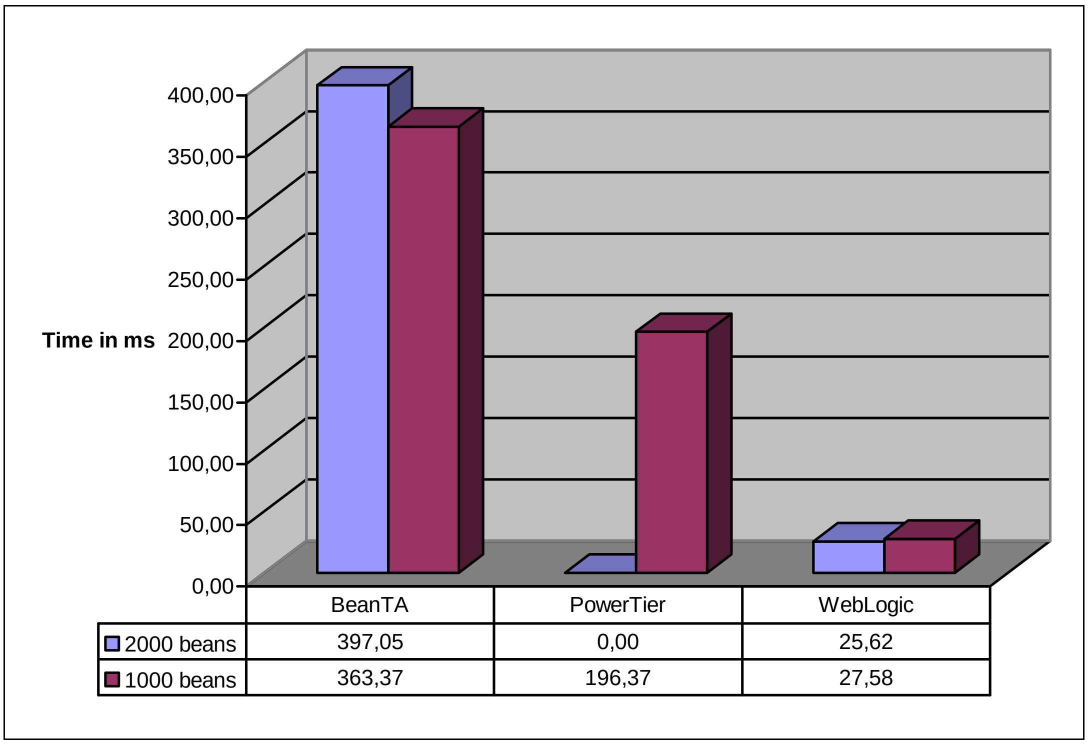


Figure 33 Retrieving a value object from the stateful session bean (2000 beans)



	BeanTA	PowerTier	WebLogic
2000 beans	397,05	0,00	25,62
1000 beans	363,37	196,37	27,58

Figure 34 Average time for retrieving a value object from a stateful session bean

8.3. Summary

All three EJB platforms differ in their list of feature and performance. When you start working with each platform it is quite obvious where each platform has its advantages. While the BeanTA platform offers very good usability by providing a lot of excellent graphical support and configuration tools and business critical features like Hot Deployment, Multi JVM architecture or sophisticated transaction management, its performance is not that good. As we have seen in the performance tests WebLogic offers much better performance and a good web-based interface for managing the server configuration. PowerTier offers a similar good performance and good O/R mapping but the fact, that most configuration and deployment tasks have to be done with command line tools, makes it less suitable to non-experts[14]. So the right platform choice will depend on the preferences of the developer. If transactions and high availability are main concerns, then a platform like BeanTA would be a proper choice. But if the performance of the application is the crucial issue then WebLogic would be a possible candidate.

The performance of an EJB application also depends on a good application design. On the one hand if we compare the results for the creating data records via stateful session beans and via entity beans and the results for retrieving data and information via stateful session beans and via entity beans we can see that clients working directly with entity beans have much better performance. However, on the other hand clients should be designed as thin as possible to achieve great flexibility and portability of applications. Besides the client application should not implement any business logic. All the necessary business logic should be provided by the server components. So if all these aspects are taken into consideration when designing an EJB application, we can state the following rules:

- When only the data from a single entity bean should be accessed or modified, the entity bean should provide the proper business logic and the client should directly work with the entity bean.
- When multiple entity beans have to be used for performing a specific task or the process consists of several more complex tasks, the business logic should be implemented within a session bean.

[14] PowerTier 6.54 does offer a basic graphical interface for the server configuration and deployment, but these tools do only provide a few basic functions and still have to mature.

With these considerations the client of an EJB application can be kept lightweight and flexible without losing much performance. Another example for the influence of good application design on the application’s performance is the use of value objects to reduce network traffic and improve the data request. As we have seen in the test series of subchapter 8.2.4 the use of value objects can bring significant performance advantage.

Despite the fact that all platforms are based on the same specification and every application should be easily deployed on another platform without changing the application, it is often quite difficult to achieve this portability. Because every platform usually provides distinctive features and special options, it is not that easy to have applications running on two different platforms without changing the source code. For example PowerTier offers a very powerful mapping tool which assists in the creation of CMP entity beans. But since PowerTier has a proprietary object-relational mechanism for better performance, these CMP beans created with PowerTier’s mapping tool can not be deployed on other platforms. This makes it very hard to write standard components and applications that can be deployed on any EJB platform, especially when you use proprietary platform features.

One main problem of the EJB specification is that some details are not completely specified. So there is much space for wrong interpretations and misunderstandings. Every vendor will interpret these details in the way it is advantageous for his product. A better and more detailed EJB specification should avoid these problems in the future. In the next chapter the future perspectives for the EJB specification will be discussed more in detail.

Chapter 9

Future perspectives

Software architecture that is based on components is a very flexible and powerful concept that will surely become a major architecture for future software projects. Due to the growing support for the necessary software infrastructure and platforms and the technical advantages of this concept it is getting more and more popular among software vendors and developers. However the development of the last few years did not keep up to the high expectations that some people had in these component frameworks.

In this chapter I want to give a short overview of changes that have occurred and introduced with newer versions of the EJB specification. Since the fact that several years has passed since the completion of the evaluation and this writing, some new versions of the EJB specification has been already released. With EJB 2.0 some substantial changes have been made to the EJB specification. A new persistence model for CMP entity beans is the most important one. But there are also interesting new features like a new query language for the finder methods and a new type of enterprise bean. The EJB 2.1 specification focused primarily on a better support for Web Services by providing new APIs for developers. The draft for EJB 3.0 does also promise some interesting new features like Dependency Injection, component inheritance or the mapping of an entity bean to multiple tables.

In the following chapters I want to give a short overview over the changes and innovations of the EJB 2.0 specification. Many points and issues that have been criticized in the EJB 1.1 specification has been improved and adopted in EJB 2.0. The main change in EJB 2.0 has been done to the persistence model of the CMP entity beans. As part of the new CMP persistence model a new query language has been defined: the EJB Query Language (EJB QL). Furthermore a new type of bean called MessageDrivenBean completes the list of new features in EJB 2.0.

9.1. Container Managed Persistence

The new CMP model is radically different from the old CMP model because it introduces an entirely new participant: the persistence manager. The persistence manager is responsible for mapping the entity bean to the database based on a new bean-persistence manager contract called the abstract persistence schema. In addition, the persistence manager is responsible for implementing and executing find methods based on EJB QL.

The container vendor or a vendor that specializes in persistence to a particular database system may provide the persistence manager. The main idea is to separate the mechanism used to manage bean relationships and persistence from the EJB container, which is responsible for managing security, transactions, and resources. The separation of these responsibilities allows different persistence managers to work with different containers. It also allows entity beans to become more portable across different EJB platforms as well as persistence managers. The persistence manager is not, however, limited to a relational database. In general persistence managers may also be developed for object databases as well as legacy and Enterprise Resource Planning (ERP) systems such as SAP.

One of the main drawbacks and most criticized aspect of the old CMP model was the missing support to model object relationships. The new CMP model is far more flexible than the previous model, allowing entities to model complex object graphs while providing for more portability across containers. Entity beans can define dependent classes that will be handled properly by the CMP mechanism. This feature greatly enhances and simplifies the creation of CMP entity beans.

EJB 2.1 added some minor improvements to the CMP persistence model introduced by EJB 2.0. With EJB 3.0 this new concept is extended by more powerful features. With the new EJB specification entity beans can now get their data from multiple tables in the database. In addition inheritance of entity beans can be modeled and used with the new CMP persistence model.

9.2. The EJB Query Language

In EJB 1.1 the database queries for the finder methods had to be specified in the deployment descriptor. Usually each EJB platform uses its own slight different syntax. So these statements always have to be adapted to the new platform where the beans should be deployed. EJB 2.0 introduces a new query language called EJB Query Language (EJB QL) for this purpose.

EJB QL is based on SQL-92 and specifies how the finder methods which are defined in the home interface should be implemented. Developers using an EJB 2.0 implementation will be freed from writing finder methods. Instead, the container provider's tools will be responsible for generating the finder methods based on the EJB QL expressions defined for the bean. With EJB 3.0 the persistence manager is responsible for generating these correct implementations. With this approach CMP entity beans become more portable and easier to deploy.

9.3. MessageDrivenBean

In EJB 1.1 the communication between beans and clients is done through remote procedure calls (RPC). A typical example would be a client who invokes a method of another remote bean somewhere on a server. But until the method invocation returns, the client is blocked; it must wait for the method invocation to end before it can execute the next instruction. Such a system is called synchronous whereas messaging systems are asynchronous. With a messaging system like Java Messaging Service (JMS) a client can send a message without having to wait for a reply. The JMS client executes the send operation and moves on to the next instruction. The message might eventually be forwarded to one client or many, none of which need to reply.

With EJB 1.1 support for the JMS API has already been integrated into the EJB framework. JMS lets you send messages from one JMS client to another through a messaging service, sometimes called a message broker or router. A message is an object in JMS, which has two parts: a header and a message body. The header is composed of routing information and metadata about the message. The message body carries the application data or payload. There are several message types depending on the type of payload, including message types that carry simple text (TextMessage), serializable objects (ObjectMessage), a collection of properties (MapMessage), byte streams (BytesMessage), primitive values streams (StreamMessage), or no payload (Message).

With EJB 2.0 a new type of enterprise bean has been introduced: the MessageDrivenBean (message bean). The message bean is designed to handle asynchronous JMS messages. The message bean is a complete enterprise bean just like the session and entity beans, but there are some important differences. A message bean does not have a remote or home interface. That is because the message bean is not an RPC component. It does not have business methods that are invoked by EJB clients. A message bean listens to a virtual message channel (topic or queue) and consumes messages delivered by other JMS clients to that channel.

Message beans are composed of a bean class that implements the *MessageDrivenBean* interface and a XML deployment descriptor. Message beans are similar to stateless session beans in that both beans maintain no state between requests. Message Beans are therefore stateless but, like stateless session beans, they may have instance variables, which are maintained throughout the bean instances' life.

As a final note on the message bean, it is important to understand that incoming messages do not have to be produced by other enterprise beans in order for the message bean to consume them. Message beans can consume messages from any topic or queue which are provided by a JMS-compliant vendor. Messages consumed by message beans may have come from other beans (session, entity, or message beans), non-EJB Java applications, or even non-Java applications that use a JMS-compliant vendor. A legacy application might, for example, use IBM's MQSeries to deliver messages to a queue, which is consumed by other legacy applications as well as message beans.

Bibliography

[Bec00]	K. Beck, Extreme Programming: die revolutionäre Methode für Softwareentwicklung in kleinen Teams, Addison-Wesley, Munich, 2000
[Den00]	S. Denninger / I. Peters, Enterprise JavaBeans, Addison-Wesley, Munich, 2000
[Fei00]	J. Feiler, Application Servers, Powering the Web-Based Enterprise, Academic Press, San Diego, 2000
[FlFa99]	D. Flanagan / J. Farley / W. Crawford / K. Magnusson, Java Enterprise in a Nutshell, O'Reilly & Associates, Inc., Sebastopol (USA), 1999
[GrTh00]	V. Gruhn / A. Thiel, Komponentenmodelle Addison-Wesley, Munich, 2000
[Lot00]	Lotus Development Corporation, Inside Notes, The Architecture of Notes and the Domino Server, available at http://www-10.lotus.com/ldd/notesua.nsf/find/inside-notes (last visited 2004-12-11), 2000
[MaSt01]	V. Matena / B. Stearns, Applying Enterprise JavaBeans, Component-Based Development for the J2EE Platform Sun Microsystems, Inc., San Diego, 2001
[Mon99]	R. Monson-Haefel, Enterprise JavaBeans, O'Reilly & Associates, Inc., Sebastopol (USA), 1999
[Mue01]	F. Müller, Object-Oriented Analysis and Design with Enterprise JavaBeans, Institute of Software Technology and Interactive Systems, Vienna University of Technology, Vienna, 2001
[Som95]	I. Sommerville, Software Engineering, Fifth Edition,

Addison-Wesley, Harlow (England), 1995

[Sun02] Sun Microsystems, Inc., Enterprise JavaBeans Specification, Final Release, Version 2.0, available at http://java.sun.com/products/ejb/docs.html (last visited 2004-11-23), 2002

[Sun04] Sun Microsystems, Inc., Enterprise JavaBeans Specification, First Draft, Version 3.0, available at http://java.sun.com/products/ejb/docs.html (last visited 2004-11-23), 2004

[Sun99] Sun Microsystems, Inc., Enterprise JavaBeans Specification, Final Release, Version 1.1, available at http://java.sun.com/products/ejb/docs.html (last visited 2004-11-23), 1999

[Sun99C] Sun Microsystems, Inc., Enterprise JavaBeans to CORBA Mapping, Version 1.1, available at http://java.sun.com/products/ejb/docs.html (last visited 2004-11-27), 1999

[Tre01] P. Tremblett, Instant Enterprise JavaBeans, The McGraw-Hill Companies, Inc., New York, 2001

[Tre01] R.S. Pressman, Software Engineering : A Practitioner’s Approach, Fifth Edition, The McGraw-Hill Companies, Inc., New York, 2001

Web Links

http://www.sun.com

Homepage of Sun Microsystems, Inc.

http://java.sun.com/products/ejb

Resources, technical papers and downloads for EJB

http://www.microsoft.com

Homepage of Microsoft Corporation

http://www.microsoft.com/net

Introduction and further information to Microsoft .NET framework

http://www.mono-project.com

Open source implementation of the .NET framework

http://www.mozilla.org

Open source browser project

http://www.mozilla.org/projects/xpcom

Introduction and further information to XPCOM

http://www.fujitsu-siemens.com

Homepage of Fujitsu Siemens Computers GmbH

http://www.microdoc.de

Homepage of MicroDoc GmbH

Abbreviations

API, Application program interface

BMP, Bean Managed Persistence

CMP, Container Managed Persistence

CORBA, Common Request Object Broker Architecture

CPU, Central Processing Unit

DBMS, Database management system

EJB, Enterprise JavaBeans

HTML, Hypertext markup language

IDE, Integrated Development Environment

IDL, Interface Definition Language

JAR, Java archive

JDBC, Java Database Connectivity

JDK, Java Development Kit

JMS, Java Messaging Service

JSP, Java Server pages

JVM, Java Virtual Machine

ODBC, Open Database Connectivity

OOA, Object-oriented Analysis

OOD, Object-oriented Development

OOP, Object-oriented Programming

O/R, Object - Relational

OS, Operation System

OSI (Open system interconnection) reference model

PC, Personal Computer

RMI, Remote method invocation

RMI-IIOP, Remote method invocation over Internet Inter-ORB Protocol

SOAP, Simple object access protocol

WWW, World Wide Web

XML, Extensible markup language