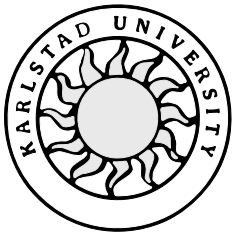


Computer Science

Henrik Boman and Carl Johan Käck

An enterprise application

- using EJB and EA Server

Bachelor's Project

2000:31

An enterprise application

- using EJB and EA Server

Henrik Boman and Carl Johan Käck

This report is submitted component in partial fulfillment of the requirements for the Bachelor's degree in Computer Science. All material in this report which is not my own work has been identified and no material is included for which a degree has previously been conferred.

Henrik Boman and Carl Johan Käck

Approved, 29 May, 2000

Advisor: Börje Arvidsson

Examiner: Stefan Lindskog

Abstract

This document contains an evaluation of the Enterprise Java Bean (EJB) specification with its features and benefits in the development of an enterprise application. It also describes Sybase Enterprise Application Server (EA Server).

The intended readers are Internet developers wanting to understand how EJB works and needing a startup on developing beans. Possible readers could also be service providers interested in future possibilities.

One of Ericsson Infotech's (EIN/I) area of responsibility is support, and for that purpose they have a database for trouble reports (TR). Ericsson has decided to rebuild the application that publishes TR from the database. Our project was to build that application using EJB and EA Server.

The main purpose of this project work was to give EIN/I a better understanding of the architecture and performance of the EJB environment. The goal was to evaluate the tools included in EA Server, the architecture of EJB and develop a working prototype.

Our conclusion is when developing enterprise applications with EJB and EA Server are a good choice.

Acknowledgements

We would like to thank our advisor Börje Arvidsson, for his professional support and help with our work. We would also like to thank our advisors from Ericsson Infotech in Karlstad Staffan Lindström and Arash Abtin. A fifth person that also has been very helpful is Lars Ohlén from WM-data eSolutions.

Contents

List of Figures

1 Introduction

This document contains a 10 weeks project to obtain a Bachelor's degree in Computer Science at Karlstad University. This project started 2000-01-17 and is made by Carl Johan Käck and Henrik Boman at Ericsson Infotech (EIN/I). We thought that EIN/I was an interesting company with great opportunities and much knowledge. We contacted them and were offered an interesting assignment, that we decided to accept.

The main reason why EIN/I decided to use Enterprise JavaBeans (EJB) is that there doesn't exist any other standards that encapsulates and separates the business logic[1] from the user interface. Another reason is that EJB is based on the Java programming language and therefor is platform independent, which was important because EIN/I is using both UNIX and Windows NT.

The choice of Enterprise Application Server (EA Server) was simpler because EIN/I was already using some programs from it as well as databases from Sybase.

Because PowerJ, discussed in chapter 4.3.2, is using Java[2] as a programming language Ericsson gets the advantage that there exist a lot of Java developers.

Our application is designed to receive an Ericsson reference number (eriref) and generate the TR as output, this is further discussed in chapter 6. The other chapters of this document is organized as follows:

In Chapter 2- Background, problems, purposes and goals- We give a description over why our project is of interests for EIN/I.

Chapter 3 – Enterprise Java Beans- explain what EJB is and what kinds of different EJB there exist. It also mentions some common used terms.

In chapter 4- Enterprise Application Server- some associated terminology within EA Server is described, and also a closer look in to some of the EA Server products is done.

Chapter 5 – Our prototype – describe in what way we have developed our prototype. We also mention our experiences, problems and how we solved them in the prototype.

In chapter 6 – Our application- we describe the development process of the application.

---

[1] Business logic describes how specific data should be manipulated.

[2] Java is an object oriented programming language created by Sun Microsystems developed in 1995. It's specially designed for Internet and other distributed environments.

Chapter 7- Experiences and recommendations- here our experiences from the development of the prototype and the application are given. We also give some recommendations upon how to develop using EA Server and EJB.

In chapter 8 – Conclusions –our opinions about EJB and EA Server is given.

2 Background, problems, purposes and goals

2.1 Background

One of EIN/I area of responsibility is support. For that purpose they have a database that contains TR. Ericsson and some customers use this database. Today a TR can only be viewed from Ericssons intranet, using the http[3] protocol. The following procedure is used; the client enters the eriref for the product of which he or she wants the specific TR. A match against the database is done and the right TR is then presented for the client.

2.2 Problems

The application that publishes the data from the database is developed using CGI[4] and Perl[5]. By using this technique it takes too much time to load a TR from the database, and it also restrict the possibilities of developing advanced applications. A third problem is that the TR can only be viewed in html[6] using http. This because the logic is not separated from the user interface. The reason for today's problems is nothing that we will discuss in this document.

[3] HTTP, Hypertext Transfer Protocol is the set of rules for exchanging files on the World Wide Web

[4] CGI, Common Gateway Interface is a standard way for a Web server to pass a Web user's request to an application program and to receive data back to forward to the user.

[5] Perl is a script programming language that is similar in syntax to the C language and that includes a number of popular UNIX facilities

[6] HTML, Hypertext Markup Language, is the set of markup symbols or codes inserted in a file intended for display on a World Wide Web browser.

2.3 Purposes

The main purpose of our project was to give EIN/I a better understanding of the architecture and performance of EJB.

Another purpose of the project was an evaluation of EA Server regarding benefits and drawbacks from the tools Jaguar CTS, PowerDynamo and PowerJ. The version of EA Server that we used was the first one to support EJB, and EIN/I was interested in knowing if there were any teeth problems with it.

It was also of interest to see if it could solve any of the problems that EIN/I wanted to solve by using EJB instead of CGI and Perl. Today's problems are: it takes too much time to load an TR, difficult to build larger applications, not able to reuse objects and not able to separate the logic from the user interface.

We knew from the beginning that it should be possible to solve these problems using EJB, because the EJB specification, see [4], says so. Therefore it was important to implement an EJB to confirm this.

2.4 Goals

A first goal of the project was an evaluation including EJB, EA Server and the development. Regarding EJB the architecture and the possibilities were of major interest. The evaluation about EA Server should include aspects like benefits and drawbacks from the tools. The development documentation would include guidelines for how to develop an EJB.

A second goal of the project was to develop a prototype that could work and solve the problems that EIN/I have with CGI and Perl.

3 Enterprise JavaBeans

3.1 Introduction

EJB technology is based on the Java programming language and defines a model for the development and deployment[7] of reusable server components.

3.2 History

During the early 90s, there was a need to shift from the twotier, client-server application model, to a more flexible multitier application model. This new model separated business logic from system services and user interface, placing it in a middletier between system services and user interface.

The growing use of the Internet and intranets for enterprise applications led to an increase of thin clients[8] and applications that wanted to communicate with the business logic.

With the twotier model the client used advanced applications stored locally on their machines. These applications often contained business logic, which communicate with the system services. Alternatively the business logic was stored at the same computer as the system services.

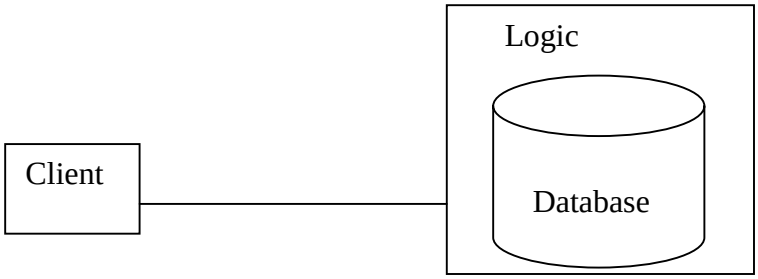
With the multitier model the business logic is stored at a separate server. The developers could now focus on programming the business logic, separated from the client application and the system services.

[7] To deploy is to spread out or arrange strategically.

[8] Thin clients are personal computers designed to be central managed and are configured with only the most essential equipment. There exist two kinds of thin clients NC developed by Oracle and Sun Microsystems, Net PC developed by Intel and Microsoft.

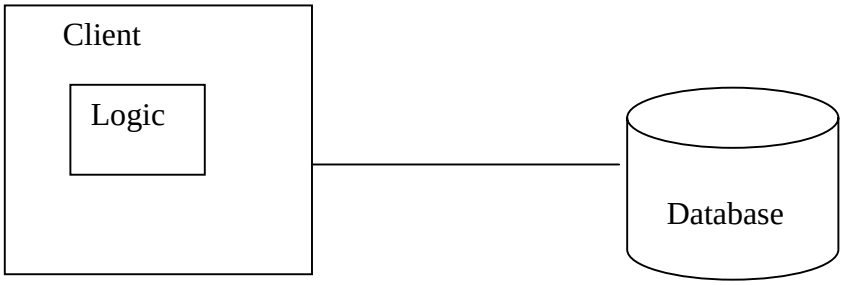
Twotier model

Alt. 1.



Twotier model

Alt. 2.



Multitier model

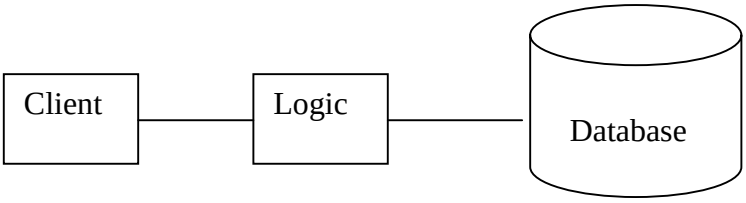


Figure 3.1: The twotier model and the multitier model.

A problem with the multitier model is that a variety of standards have been used, which has caused the following problems:

- Making it difficult for developers to build efficient applications from standardized components, due to different components with different interfaces.
- To deploy a single application on various platforms, because an application could only work on special platforms.
- Changing existing business logic.
- Vendor dependencies.

In general components that were developed in one environment could not be moved and work in another environment.

Sun Microsystems wanted to solve these problems by creating EJB.

They decided to make a standard for application interfaces so that you could gain the possibility to buy business system parts from different providers without causing any integration problems. [5]

3.3 Terminology

Within the EJB-technique there are a lot of terms and rules for how an EJB should be developed to fulfill the EJB specification.

Some of the most usual terms within EJB are *server*, *container*, *client* and *component*. Server is a program that contains data or business logic. A container consists of one or several components and acts as an interface towards the components. The term client means the user application of a system. A component is a program module that is placed in a container. The criterion for an application to be named component is that it must be developed according to the component model.

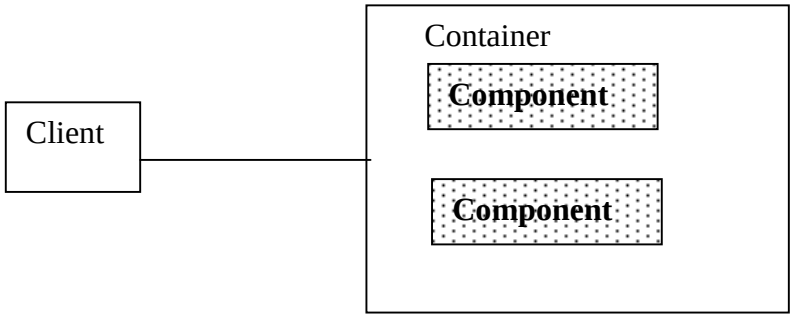


Figure 3.2: The EJB technique model.

3.3.1 Server

A server that supports EJB is designed to work as a container to provide services for its components and other containers. The server contains the business logic and often has access to the database. It is often used as a middletier, and gives the opportunity to communicate towards other databases on other computers.

3.3.2 Container

A container is a program in which the program building blocks known as components are executed. It contains one or several EJB-components and instructions how to communicate with each component. The container is responsible for creating, activating and eliminating components, depending on what kind of request the client has requested. The container is also used for handling details like a component lifecycle, transactions and security.

The container that we used in our project was Jaguar CTS, discussed in chapter 4.3.1.

3.3.3 Client

The interface of a component always looks the same for a client, independent of which container that is being used on the server side. From the client's point of view it seems to have direct contact with the component, but in reality the client communicates indirectly with the specific component via the container.

3.3.4 Component

An EJB-component is a class[9] containing a number of methods[10]. Components can be reused, which lead to the big advantage "Write once, run anywhere". This means that the developer only has to write the code once and it is executable on any platform as long as there exist a Java Virtual Machine[11] for the platform.

There exist two different kinds of components, Session Bean and Entity Bean.

3.3.4.1 Session bean

A session bean is transient and constructed on behalf of the client and in most cases it only exists during client/server session. A session bean performs operations on behalf of the client, such as accessing a database or performing calculations. But a session bean doesn't represent something in a database like the entity bean. Another difference between them is that a session bean does not recover after a system crash.

There exist two kinds of session beans, stateless and stateful.

---

[9] A class is a definition of the methods and variables in a particular object.

[10] A method is a programmed procedure that is defined as a part of a class and included in any object of that class. In many other programming languages methods are called functions.

[11] Java Virtual Machine is software that acts as an interface between the compiled Java code and the processor that actually performs the instructions.

A *stateless session bean* is quite easy to develop, and also very efficient. It requires few resources because they are neither persistent nor dedicated to one client. That means that an EJB object can use just a few instances of a stateless bean. As soon as a stateless instance has served a method invocation it can be swapped to another EJB object immediately. Due to that a stateless session bean doesn't perform any overhead of swapping for passivation or activation. It leads to faster execution, because it’s always active.

Everything a session bean method needs to know has to be passed via the method's parameters. The only exception is information obtainable from the SessionContext. This yield the following limitation, stateless session beans should not remember anything from one method invocation to the next. Because of this, the methods have to take care of the entire task in one method invocation. Once the procedure is finished nothing about the data that was manipulated or details of the request are remembered.

Stateless beans can be used for report generation, batch processing, or services like validating a credit card. Any activity that can be accomplished in one method call can use a stateless session bean. [5]

A *stateful session bean* offers an alternative that lies between entity beans and stateless session beans. It is dedicated to one client during the lifetime of the bean instance. It also acts on behalf of a client. They are not swapped among different EJB objects or kept in an instance pool like stateless bean instances. Once a stateful session bean is instantiated and assigned to an EJB object it is dedicated to that EJB object for its entire life cycle.

Stateful session beans remember the variables of the bean class between method invocations. This makes it possible for methods to be interdependent so that changes made by methods to the bean's state can affect the result of subsequent method invocations, in contrast to the stateless session bean. Although stateful session beans maintain their conversational state, they are not themselves persistent like entity beans. Stateful session beans can access a database but they don’t represent data in a database.

Stateful session beans allows you to encapsulate the business logic of a client and move it to the server, which makes the client application thinner.

The biggest difference between the stateful session bean and the other types of beans is that it doesn’t use instance pooling. The stateful session beans are dedicated to the client for their entire life, so there is no swapping or pooling of instances. Instead of pooling instances, stateful session beans are simply evicted out of memory to return resources when it is passivated. The EJB object remains connected to the client, but the bean instance is

dereferenced and garbage collected during inactive periods. The life cycle of a stateful session bean has three states: Does Not Exist, Method-Ready and Passivated. [5]

3.3.4.2 Entity bean

An Entity bean describes both the state and the behavior of real world objects. This allows developers to encapsulate the data and business rules associated with a real world object. For example an entity bean might represent a customer or a piece of equipment. This makes it possible for data associated with a concept to be manipulated consistently and safely.

There are many advantages using entity beans instead of accessing the database directly. Utilizing entity beans to objectify data provide programmers with a simpler mechanism for accessing and changing data. It is much easier to change an attribute for an object using an entity bean than executing a SQL command against a database. Using entity beans for objectifying data also makes it simpler to reuse software units. Once an entity bean has been defined its definition can be reused, in a larger consistency.

When a new bean is created a new record must be inserted into the database and a bean instance must be associated with the data. Every entity bean could be found by using its primary key.

As the bean is used and its state changes, these changes must be synchronized with the data in the database. The process of coordinating the data represented by a bean instance with the database is called *persistence*.

There are two types of entity beans, and they are distinguished by how they manage persistence.

Container-managed beans have their persistence automatically managed by the EJB container. The container knows how a bean is mapped to the database and automatically takes care of inserting, updating and deleting the data associated with entities in the database.

Beans using bean-managed persistence do all this work explicitly. The bean developer therefore must write the code to manipulate the database. The EJB container tells the bean instance when it is safe to insert, update and delete its data from the database, but it provides no other help. The bean instance does all the persistence work itself. [5]

3.3.4.3 Interfaces

There exist three kinds of interfaces; remote interface, home interface and the bean class.

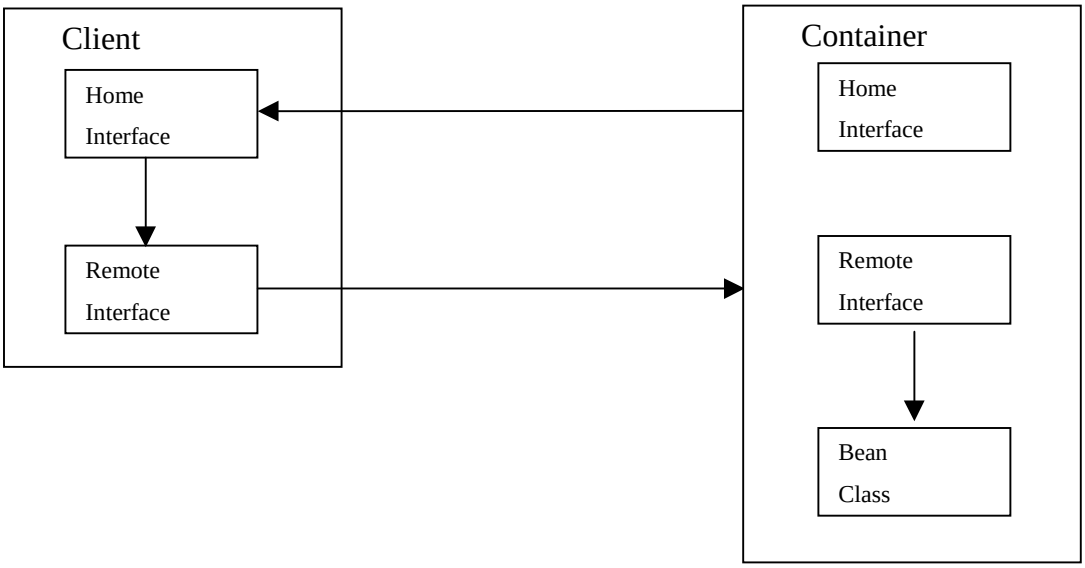


Figure 3.3: The remote interface, home interface and the bean class.

The *remote interface* for an EJB defines the bean's business methods that are accessible outside the bean.

The *home interface* defines the EJB's life cycle methods, methods for creating new EJBs, removing EJB and finding EJBs.

The *bean class* contains the bean’s business methods. Surprisingly, the bean class usually does not implement the bean's home or remote interfaces. However, the bean must have methods matching the names of the methods defined in the remote interface and it also must have methods corresponding to some of the methods defined in the home interface.

The client never interacts with the bean class directly, it always go through the methods of the bean's home and remote interface which invoke the right method. The interacting with the stub is generated automatically using the skeleton, discussed in chapter 4.1.3.
[5]

3.3.5 The component model

The component model defines the basic architecture of a component. The model specifies the structure of the component interface and the mechanisms by which the component interact with its container and with other components.

The model specifies how components should be created and implemented, so that they can co-operate to form a larger application. Thanks to the component model application developers can combine components from different vendors to construct an application.

3.3.6 Standard Interface

In order to qualify as a component, the source code must have a standard interface that enables other parts of the application to invoke its functions, access and manipulate variables within the component.

The component model defines the structure of the interface.

An application builder should therefore be able to use a component without looking at the source code.

4 Enterprise Application Server

Sybase Enterprise Application Server is a component transaction server and a dynamic web page server. It supports most industry standards, see chapter 4.2 supported standards. The products that EA Server includes are Jaguar CTS, PowerJ, PowerDynamo and some more products that won't be discussed in this document. We will discuss each product later. Using EA Server also gives you an easy development environment when most of the products are integrated with each other. [3]

There are also some common used terms that we will try to explain.

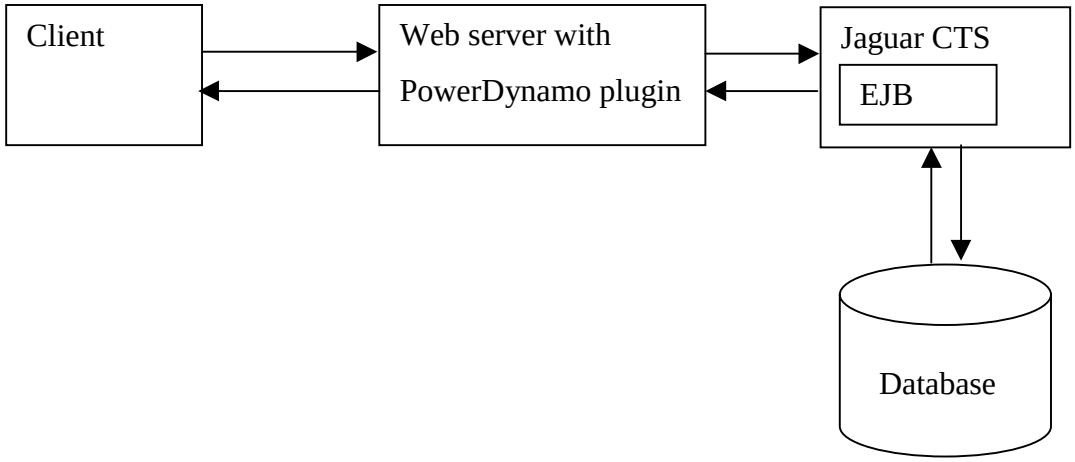


Figure 4.1: The multitier model.

4.1 Terminology

EA Server applications consist of packages, server applications and client applications. The server and client applications are made of components, which consists of a number of methods.

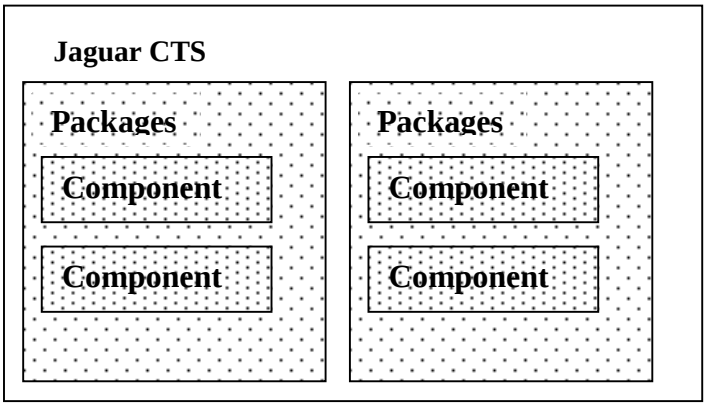


Figure 4.2: Model of EA Server consisting of packages and components.

4.1.1 Component

It is in the components you can find the logic of the application. The logic can be used for accessing a database or communicating with the client, etc. The client application informs the client about what methods exist on a server. All execution of the components is made at the server side.

EA Server supports the following types of components: [3]

- ActiveX[12]
- C[13]
- CORBA[14] using C++[15]
- Java

[12] ActiveX is a set of object-oriented programming technologies and tools from Microsoft.
[13] C is a programming language developed for the creation of UNIX.
[14] CORBA is an architecture and a specification for creating, distributing, and managing distributed program objects in a network.
[15] C++ is an object-oriented programming language.

4.1.2 Package

A package is a collection of one or several components that provides one or several services. A package represents a part or the entire application business logic, within components easily can communicate with each other. Each package acts as a unit, which makes it easier to deploy and manage the components.

4.1.3 Stub and skeleton

The stub and skeleton are responsible for making the object server (object on the middletier) and its components to look as if they are running locally on the client machine

This can only be accomplished through some kind of remote method invocation (RMI) protocol.

A *stub* is a Java or C++ class that acts like a proxy object for the component. A stub communicates with the server to initiate and activate methods.

A *skeleton* is an interface between the server runtime environment and the methods of the application that are being executed. Skeletons are compiled and linked with the components and enable the server to activate the correct method during the execution.

Every instance of the object server has its matching skeleton class.

The skeleton, placed in the middletier, sets the port and IP address and listens for request from the stub. The stub is placed at the client side and is connected via the network to the skeleton. The stub is responsible for communicating and distributes the request from the client to the object server through the skeleton.

The stub acts like the object server's surrogate and implements an interface with the same business methods as the object itself, but the stub's methods do not contain business logic. Instead the stub forwards the request to the object server and then receives the result from the real object server. The procedure is the following; the client invokes a business method on the stub, and the request is communicated over the network by streaming the name of the method invoked and passing the values as parameters to the skeleton.

When the skeleton receives the incoming stream, it parses the stream to find what method the client wants to invoke by its request, and then invokes the corresponding business method on the object server. Any value that is returned from the method on the object server is streamed back to the stub through the skeleton. The stub then returns the value to the client application as if it had processed the business logic locally.

[3]

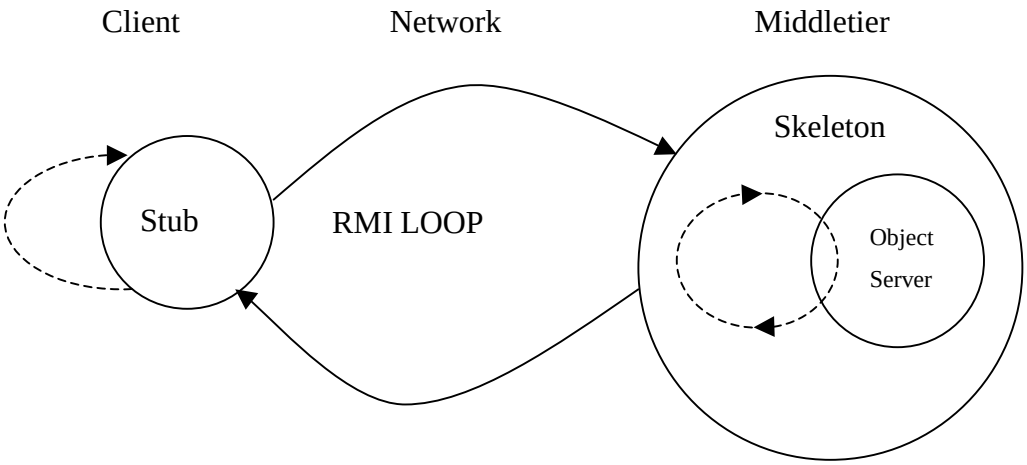


Figure 4.3: RMI loop.

4.1.4 Connection cache

The connection cache makes sure that each business object in the system can share database access.

The components could access the remote database through preallocated connections in the connection cache. A connection cache reduces the overhead of establishing connections, because a connection is established once and reused for several components, instead of that each component creates a new connection. This leads to that the server and database CPU time and memory are not consumed by opening more connections than necessary, which increases the throughput.

You can specify the load of the database from the components by setting the number of simultaneous connections in the connection cache. All connections for a cache must use the same ID, password, database server and connectivity library.

Connection caching also improves the scalability since connection caching allows the same number of clients to be served using fewer database connections and less memory.

The use of connection cache solves the problem that it takes long time to load a TR, discussed in chapter 2.2.

4.1.5 Instance pooling

Its called pooling when you are using a program for continuously checking what state a program or a device has. Often it is used to see whether a program or a device is still connected or wants to communicate.

Jaguar is using instance pooling to maintain a cache of component instances and bind them to client sessions on as-needed basis. When components support instance pooling the scalability of the application increases. Instance pooling reduces execution time and memory consumption that otherwise would be spent on allocating unnecessary component instances.

4.2 Supported standards

Because of that Jaguar CTS supports most industry standards EIN/I will support the same standards as well, which is discussed as one of the functional requirements in the system requirement specification. [3]

4.2.1 Web server

EA Server includes a web server and it also supports other web server interfaces as: [3]

- NSAPI[16]
- ISAPI[17]
- CGI

4.2.2 Distributed Protocol

To make it easy to manufacture applications for different environments EA Server supports the following distributed protocols: [3]

- Objects: IIOP[18] and IIOPS[19].
- Web: HTTP and HTTPS[20].
- Database: TDS[21] and TDSS[22]

[16] NSAPI, Netscape Server Application Program Interface produced by Netscape.
[17] ISAPI, Internet Server Application Program Interface produced by Microsoft. It's a set of Windows program calls that you can use for building a Web server application.
[18] IIOP, Internet Inter-ORB Protocol is an object-oriented protocol that makes it possible for distributed programs written in different programming languages to communicate over the Internet.
[19] IIOPS, Secure Internet Inter-ORB Protocol is the same as above but with secure extension.
[20] HTTPS, Secure Hypertext Transfer Protocol is the same as HTTP but with secure extension.
[21] TDS, Tabular Data Stream is an application protocol for Sybase databases.
[22] TDSS, Secure Tabular Data Stream is the same as above but with secure extension.

4.2.3 Data access

Today most of the enterprise applications include some interaction with external data. The supported standards of EA Server for interaction with external data are: [3]

- JDBC[23]
- ODBC[24]
- CT-Library[25]

4.2.4 Security

As we are heading for a model where data should be accessible from any computer that is connected to Internet a very important feature is security. It's also getting more important when most systems are integrated with each other to make more information available. EA Server supports the following standards: [3]

- SSL[26] 3.0
- X.509[27] v.3
- RSA[28]
- Integrated operating system security

4.2.5 Naming

Naming associates an object with a logical name. The objects can be packages, components or servers. Each object and its attached name create an object *namespace*. Applications reference an object by its namespace. The following naming standards are supported: [3]

- CORBA, CosNaming[29].
- Java, JNDI[30].
- Storage, LDAP[31].

---

[23] JDBC, Java Database Connectivity is an application program interface specification for connecting programs written in Java to a database.
[24] ODBC, Open Database Connectivity, is a standard for accessing a database.
[25] CT-Library, Component Transaction–Library, is a model for accessing a database from Sybase.
[26] SSL, Secure Sockets Layer, is a program layer created by Netscape for managing the security of message transmissions over a network.
[27] X.509 is a standard way to develop an electronic directory of people in an organization so that it can be part of a global directory available to anyone with Internet access.
[28] RSA, Rivest-Shamir-Adleman, is an Internet encryption and authentication system.
[29] CosNaming, is CORBA's naming system.
[30] JNDI, Java Naming Directory Interface, is the naming and directory service for the Java platform.
[31] LDAP, Lightweight Directory Access Protocol, is a software protocol for enabling anyone to locate organisations, individuals, and other resources such as files and devices on the Internet or on a corporate intranet.

4.3 EA Server products

EA Server's main feature is a component transaction server, Jaguar CTS. Here is the business logic for the enterprise application stored. It is also Jaguar CTS that communicate directly to external data sources.

The development environment for the business logic is PowerJ using Java as the programming language. In PowerJ you are using components and objects to develop programs. PowerJ provides an interface for creating a graphical user interface (GUI) where you create the components using drag and drop technique and set its properties. PowerJ also includes tools for deployment and debugging components.

Another product that EA Server includes is PowerDynamo that is a tool for publishing data. It supports dynamic-, static-html and xml[32]. It has it's own script language specially designed for server side scripting, DynaScript.

4.3.1 Jaguar CTS

Jaguar is a component transaction server, aimed for supplying business logic to client programs in multitier architecture. Jaguar is intended to work as a middleware[33]. Using Jaguar is very simple and most of the configuration can be done when developing the components in PowerJ, thanks to the good integration.

4.3.1.1 The multitier model with Jaguar

Jaguar CTS uses a multitier distributed computing architecture with four distinct elements, se figure 4.1. The elements works together to provide the user with data. The four elements are:

- Client-side applet or application
- Web server
- Middletier components, business logic
- The target database

When the client-side has downloaded the applet a session is started at the server and its components are instantiated to serve the client. The client application (applet) has the same interface as the server application (servlet), but it doesn't include the logic. When the user calls one of the applets methods the applet sends a request to the middletier (servlet). It

[32] xml, Extensible Markup Language, is a flexible way to present information on the World Wide Web, intranets, and elsewhere.
[33] Middleware is a general term for any program that mediates between two separate existing programs.

executes the specific method that includes the logic, which access the database. The result is returned to the applet, that presents it for the user.

4.3.1.2 Jaguar Manager

Jaguar Manager is a tool that Jaguar CTS provides to administrate and manage a Jaguar server. It serves efficient management of client sessions, security, threads and thirdtier database connection.

With Jaguar Manager you can add new components to a running server and also configure the server in runtime. Jaguar Manager also provides the opportunity to add a new server to the cluster, discussed in chapter 4.3.1.4. Jaguar Manager gives the opportunity to edit a component's property, deploy components and to use a runtime monitor utility for handling server events and statistics, which should help to predict and prevent problems.

4.3.1.3 Naming services

A naming service yields the opportunity to bind a logical name to an object, such as a package and component. Naming helps applications to easily locate and implement an object on a network. The combination of attached name and its referenced object is the name context. The collection of name context information is called namespace. Each Jaguar server can be its own name server, or you can configure Jaguar to use another server with an external naming service as the name server.

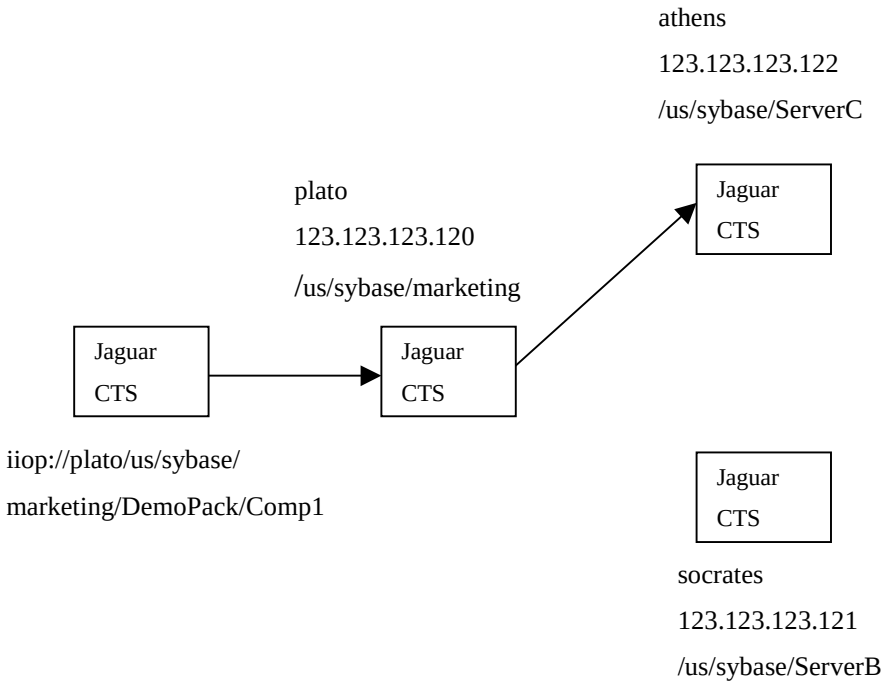


Figure 4.4: Naming Services

4.3.1.4 Cluster

A cluster is a group of servers that share replicated information. A cluster's primary purpose is to provide load balancing and high availability.

Each cluster includes a primary server, participating servers and name servers. The primary server contains the master copy of the configuration, which it distributes to all servers in the cluster. All servers in a cluster share a logical server name, which corresponds to a Jaguar server, defined in the primary server's repository.

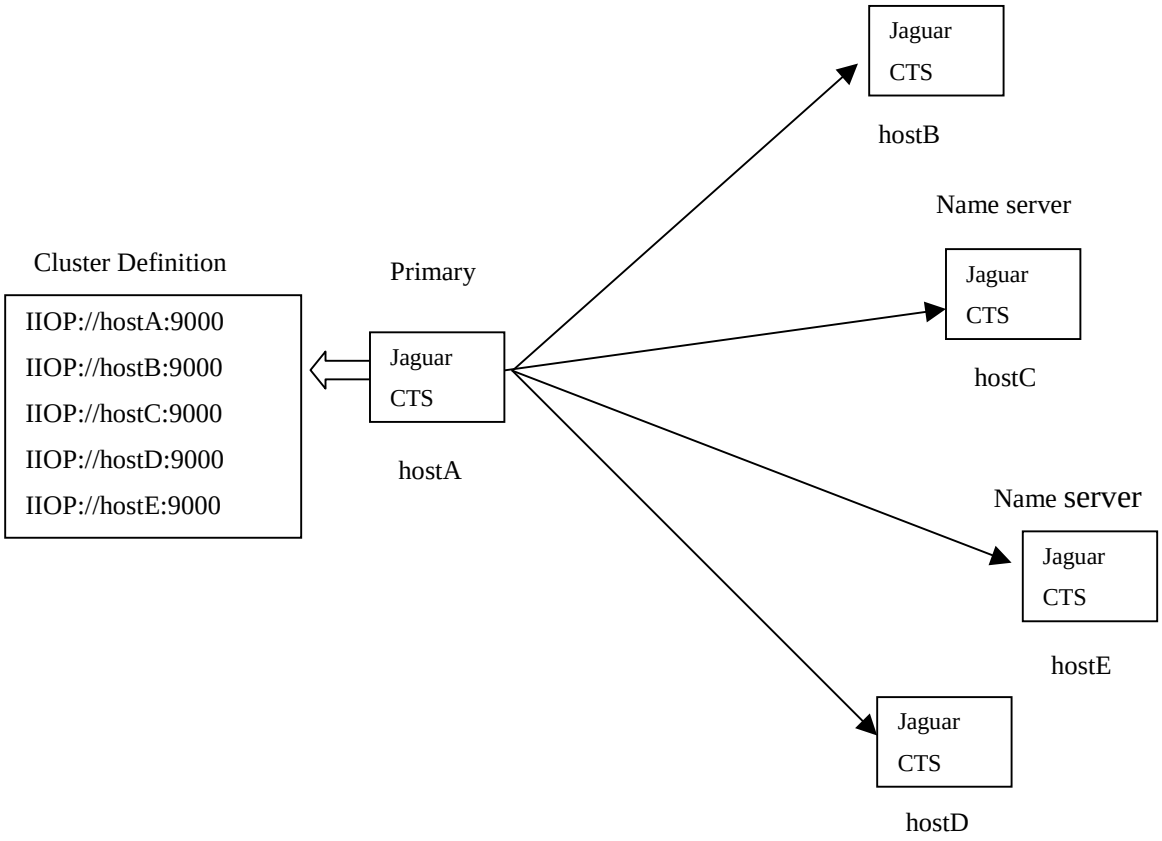


Figure 4.5: Cluster

If high availability is of major interest each server name should be shared by at least two physical servers in the cluster. If one server is unavailable the other server can respond to the request. All hosts of a cluster must have the same platform type, you can't mix NT and UNIX in a cluster and each server can only be member of one cluster.

Synchronization enables distribution of information from the primary server in a cluster to participating servers in the cluster. To synchronize a cluster, you must be connected to the primary server of the cluster.

If using clusters, synchronization ensures that logical servers in a cluster share the same application files and configuration. It is also possible to synchronize non-clustered Jaguar servers.

Synchronization provides a quick and easy way of distributing configuration information between the servers, it's also a useful alternative to importing and exporting packages. You can distribute new components from a test or development server to the other servers, in the cluster.

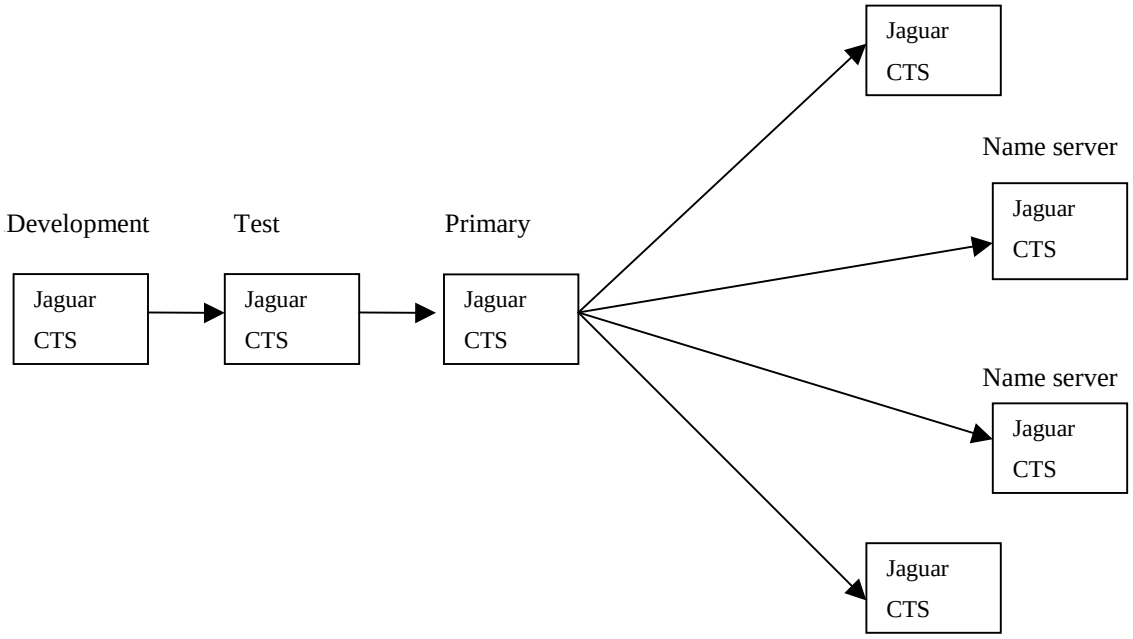


Figure 4.6: Synchronization

[3]

Load balancing is used for optimizing the performance of a Jaguar cluster by adjusting the connections across the servers. Three factors determine the load balance; *distribution policy*, *metrics* and *interoperable object reference.*

When configure the load balance you select the *distribution policy* that best matches your environment and situation. There are four load distribution policies random, round robin, weighted and adaptive. Adaptive distribution policy is the only policy that is dynamic.

If you select the dynamic load policy the *metrics* is used to determine the load on your servers and gives each server weightning, which is then used to distribute connections for optimizing the performance of the cluster.

The *interoperable object reference* contains a profile that the client uses to find a component. The profile contains the server and port number that the client should use to access the component. The distribution policy determines in what order the profiles are distributed to the clients. [3]

4.3.2 PowerJ

PowerJ is a graphical application development tool that is used for building Java applications for distributed, web and client/server applications. With PowerJ you develop applications using components, objects and routines that handle events for objects.

Components are simply classes that fulfill the JavaBean specification. They can either be visible for the user or hidden inside a program. An object is an actual instance of a component.

For example a textbox is represented by the TextBox component. The actual textbox that is displayed is an object. All kinds of objects respond to a number of events.

4.3.2.1 Developing with PowerJ

The first step when developing a component using PowerJ is to create a user interface. The design of the user interface is made by creating objects like buttons and textboxes using drag and drop technique. Next step is to set the properties and appearance for each of the objects. To handle the possible events of an object you can either write Java code by yourself or you can use a drag and drop code generator. By using the code generator it minimizes the typing and assures the correctness of the syntax.

When using PowerJ you are also given the opportunity to type all of the code yourself or just specific parts.

PowerJ also has it's own component library, which is a collection of Java classes.

4.3.3 PowerDynamo

PowerDynamo is a dynamic page server. A page developed with PowerDynamo can be compared to JSP[34] or ASP[35]. The difference between a dynamic and static page server is the way of changing the content of a document. In a static page server environment you have to change the actual file to change the content. With a dynamic page server this isn't necessary,

[34] JSP, Java Server Pages is a technology for controlling the contents or appearance of web pages through the use of servlets. JSP is developed by Sun Microsystems.

[35] ASP, Active Server Pages is an html page that includes one or more script that are processed on a Microsoft Web server before the page is sent to the user. Microsoft has developed ASP.

the page can for example receive it's content from a database and each time the document is loaded from the server it will load a new set of data from the database.

PowerDynamo is a tool for creating and managing a dynamic web environment.

It has also its own script language DynaScript, and a set of tags Dynamo tags.

4.3.3.1 The website design and administration tool

The website and the administration tool, *Manage PowerDynamo* is a plug-in to the Sybase Central for managing a website and its data. Manage PowerDynamo provides tools for adding Scripts and a Wizard to create a new template. To create and edit code there is a syntax high lightning editor available. It is also possible to view the generated html output from your script.

4.3.3.2 The web server / web server extension.

PowerDynamo includes a web server, Personal Web Server. It supports DynaScripts and Dynamo tags.

4.3.3.3 Templates

Templates are html/xml pages that can include embedded instructions, dynamo tags, for generating dynamic code. Examples of instructions are SQL queries and formatting of the result from a query. It is also possible to include a DynaScript into a template. The templates are stored as files or in a database.

4.3.3.4 DynaScript

DynaScript is an object-based server side script language. It uses syntax that is similar to JavaScript and follows the ECMAScript[36] standard. [3]

Using DynaScript you either write the code in your html document using a Dynamo tag or you write the code in a script file and call for it from the html document.

DynaScript includes control statements; if, while, do-while and for. Most of the standard data types for variables are supported. PowerDynamo also supports the use of functions and has several built in functions for working with databases, SQL and html-documents.

[36]ECMAScript is a standard script language, developed by Netscape and Microsoft and comes from Netscape's JavaScript.

The Hello world example in html using DynaScript.

```
<HTML>
<TITLE> Document using DynaScript </TITLE>
<BODY>
<!--SCRIPT
document.Write( "Hello world!" );
-->
</BODY>
</HTML>
```

4.3.3.5 Dynamo tags

Dynamo tags are an extension of the normal html tags for creating html documents using PowerDynamo. Dynamo tags are like DynaScript executed at the server in contrary to normal html tags, which are executed at the client. Dynamo tags can be implemented in any html editor, because the html editor will only see the Dynamo tags as comments. To make sure that the tags are implemented correct you can test your application in PowerDynamo or PowerJ.

Examples at Dynamo tags are SQL, FORMATTING, the error checking tags, INCLUDE and SQL_INSERT. You use the SQL tag to include SQL quires to your page.

The FORMATTING tag is used for displaying rows in the form of a table or a list.

The error checking tags are tags enabling to check whether an instruction was completed successfully or not, for example deleting a post from a database.

The INCLUDE template can be useful if several documents should contain similar SQL quires. Instead of inserting the queries to each document a template containing the common queries can be included in each document.

The SQL_INSERT is to update a database from a web page.

You can either place the code for the tags in a separate file and access it in the html document, or you can place the code into the html document direct.

Small example of an html-document using Dynamo tags.

```
<HTML>
<TITLE> SQL-tag in a html document </TITLE>
<BODY>
```

```
<!--SQL
SELECT customer.id, customer.lname, customer.fname, customer.phone
FROM DBA.customer customer
 -->
</BODY>
</HTML>
```

[3]

4.3.3.6 A Comparison between DynaScript and Dynamo tags

The main difference between DynaScript and Dynamo tags is that with DynaScript you can include logic. It's only possible to receive some logic from Dynamo tags by using the SQL tag to make a query towards the database. DynaScript is closer to a simpler programming language than Dynamo tags. But in the same manner as with scripts versus programming languages Dynamo tags gives higher readability than DynaScript.

5 The prototype

The first prototype was made early because it was of interest to see what problems that could occur during the development of an application.

This also gave us a better understanding in what EIN/I wanted us to develop.

In this section it's given a short description about our experiences as well as problems that occurred and in what way they were solved.

5.1 An overview

The idea was to display a trouble report from an html-page for a certain eriref.

The procedure is started when the user from an html-page enters the eriref in a textbox-field that he or she is interested in and then pushes the submit button. The submit button invokes the dynamo-script which fetches the eriref. The script creates the EJB and executes the queries with the entered eriref as input, towards the database. The TR is returned back to the script, which displays the TR on the screen.

5.2 The EJB

The EJB was developed using a guide for creating a distributed application. The first step was to create a transaction object towards a locally installed Jaguar server. But instead we chose to have a non-local Jaguar server because in our final application it would be non-local. The second step was to create a connection cache, which was set towards a non-local database. This database is a copy of the actual database that EIN/I today uses for TR.

The guide then suggested making one query using drag and drop technique and letting PowerJ generate the code for the object. We created eight queries because in the final application it would be about 30 queries. The SQL statements we used are the ones that Ericsson uses today on their existing system.

For each query we created a unique method in the remote interface to make them executable and reachable for the dynamo script. In the final implementation there will only be one method that executes all the queries. The reason for this is to make the application faster by reducing the number of connections and disconnections between the web server and the

Jaguar server. Using just one method also gives a simpler interface towards the EJB. The negative aspect with having one method is that the object executed at the server is larger.

Together with the eight queries we also created a method for sending the eriref parameter from the dynamo script to the bean. The eriref is used to complete the SQL statements so that they are specified to search for the right trouble report.

When creating the queries we had to specify the SQL statement, but as that the eriref is variable we couldn't complete them. After receiving the correct eriref we overwrote the SQL statements, with the correct eriref. Because of using the guide to create the query objects we couldn't change the code for them afterwards. In the final application we will create the query objects ourselves when we have received the eriref. This will result in a better programming manner without any confusing code.

To be consequent we decided to make the transaction object ourselves as well.

5.3 The PowerDynamo script

The script starts with creating a connection towards the Jaguar server.

To be able to create the EJB we first had to create a home interface object.

After that we execute the method which sends the eriref to the EJB and sets the queries.

Then we execute each query using its retrieve method defined in the EJBs remote interface. After receiving the result the script displays it on the screen.

The script ends with killing the EJB object, so that the used memory is returned back to the system.

5.4 The html-page

The html-page will not be created in our final application, due to that it already exists in today's system. The reason why we have created an html-page was to simulate the eriref input from a user. We also needed something to invoke our dynamo script with an eriref as input.

6 The application

6.1 Background

This chapter gives a description of our final application. The application that is used today by Ericsson for the purpose to get a TR from a database has turned out to be too slow. The reason for this is when using CGI you have to set up a new connection for each new request. We therefore got the assignment to create a better and faster application using EJB and EA Server. Another feature that our application should contain was the ability to be accessed from different clients, using other protocols than http.

6.2 Introduction

The main task of the application is to post a TR towards the user. This is achieved by invoking a dynamo script on a server on behalf of the client which sends an eriref. The script invokes the bean that connects to the database and executes the queries. The bean returns the result from the queries in xml to the script. The script creates an html page from the received xml and presents it for the user. The reason why the bean returns the result in xml is that we want to separate the data from the user interface so the EJB can be reused for different kinds of clients. For example if we made a xml to wml[37] parser the site could be viewed from a cellular phone using wap[38] as well.

[37] wml, Wireless Markup Language, is a language that allows the text portions of Web pages to be presented on cellular phones and personal digital assistants via wireless access.

[38] wap , Wireless Application Protocol, is a specification for a set of communication protocols to standardize the way that wireless devices can be used for Internet access.

6.3 The implementation of the EJB

When we had developed the prototype we knew quite well how to create the final EJB. We started by creating a new EJB and named the Java package to se.ericsson.ks.TRBrowse, this name might be a bit confusing but it follows the naming rules specified by the EJB specification.

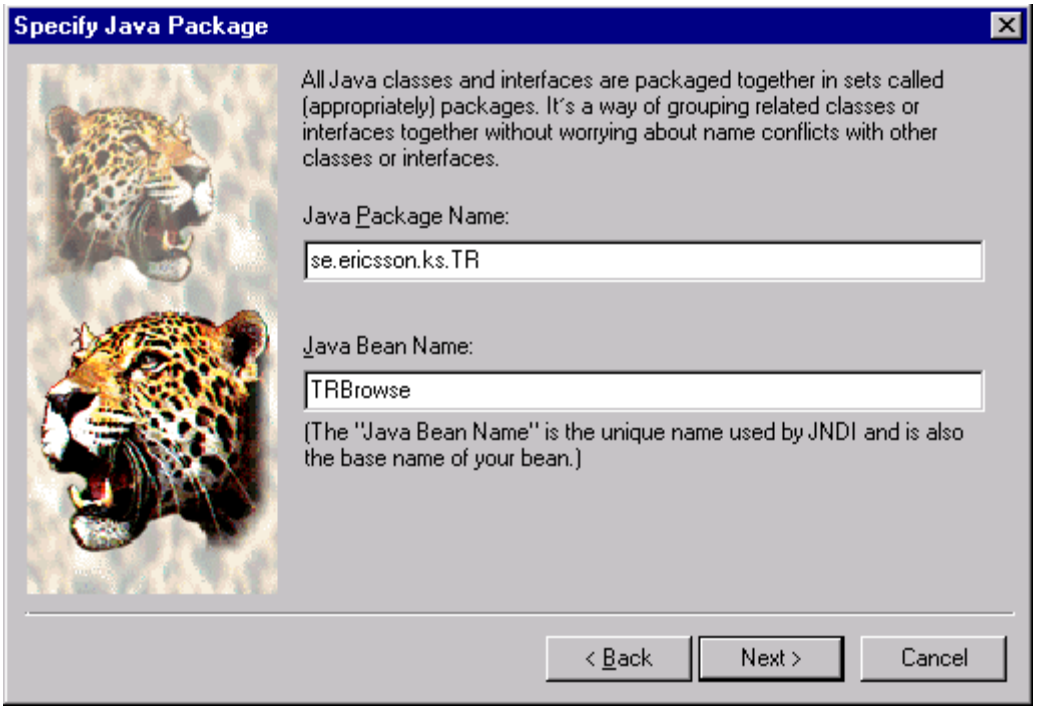


Figure 6.1: Screen view from PowerJ.

Next step was to choose whether it should be a session bean or an entity bean. We chose session bean because we wanted it to be transient and constructed on behalf of the client. We didn't chose an entity bean because our bean shouldn’t represent data from the database, it was supposed to get data from the database.

After that we had to specify whether the bean should be stateful or stateless. The EJB shouldn’t be persistent because of the overhead for creating and destroying EJB. Furthermore the EJB doesn’t have to remember anything between the sessions. Therefore we wanted the EJB to be stateless. We now had a bean with interfaces and necessary methods such as constructor and destructor.

After making the above decisions the first step was to fill in some necessary information to the precreated methods. Next step was to insert the methods we wanted in the home interface

and the remote interface so that we had a working skeleton. When a method is created in the remote interface it is also created in the bean class, were the code for the application is stored.

What we needed in the remote interface was a method that the script could use to receive the xml string that contains the data for the TR. We also used the same method for sending the eriref to the bean.

When the bean has received the eriref, it creates a transaction object and fetches the settings for it from the connection cache at the jaguar server and sets up a connection towards the database.

When the method that was invoked by the script has established a connection towards the database it executes the method for the first query, each query has it own method. The method for the first query starts with setting the SQL-statement. It then invokes a method that executes the first query and returns the result set to it. The method that executes the SQL-statement is used for all query methods. When this is done the first query method takes the sub result from the first query and sets the xml tags for each sub result. The xml string is then returned to the method that was invoked by the script. This procedural is repeated for all queries. When the last query is executed the method that was invoked by the script disconnects from the database and returns all the xml strings that it has brought together to one long string to the script.

There exist other methods than those described above but they might not be very interesting for the project. In figure 6.2 you can see the skeleton of our application.

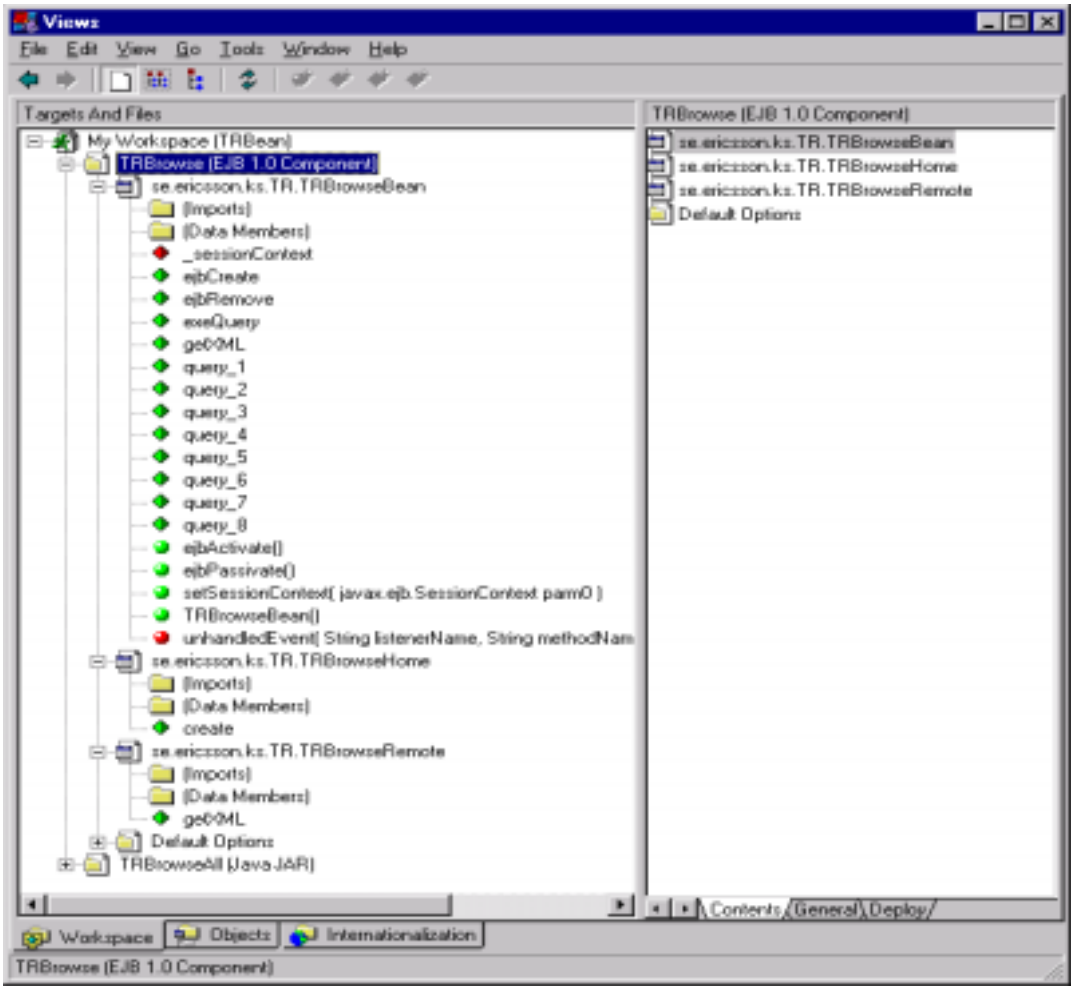


Figure 6.2: The workspace in PowerJ, displaying the skeleton of our EJB.

6.4 The PowerDynamo script

We are using PowerDynamo to create an exact copy of today's presentation of the TR towards the user.

The following procedure is done; from an html-page the user enters an eriref and then pushes a submit button. The submit button sends the entered eriref, which is fetched by the html-page developed in PowerDynamo.

The html-page has a dynamo-script embedded. The script starts by setting up a connection towards an instance of the EJB, on the Jaguar server. This means that we now have access to the home interface of our bean. After that we create a home interface object and execute the create method in the bean home interface. When this is done the remote interface of the bean

is accessible. This makes it possible to invoke methods in the bean, but only those methods that are included in the remote interface.

By invoking a method in the remote interface with the eriref we receive the result (TR) as a xml string. After parsing out the values in the xml string we place each of the values at the right tag in the html-page, and present it to the user.

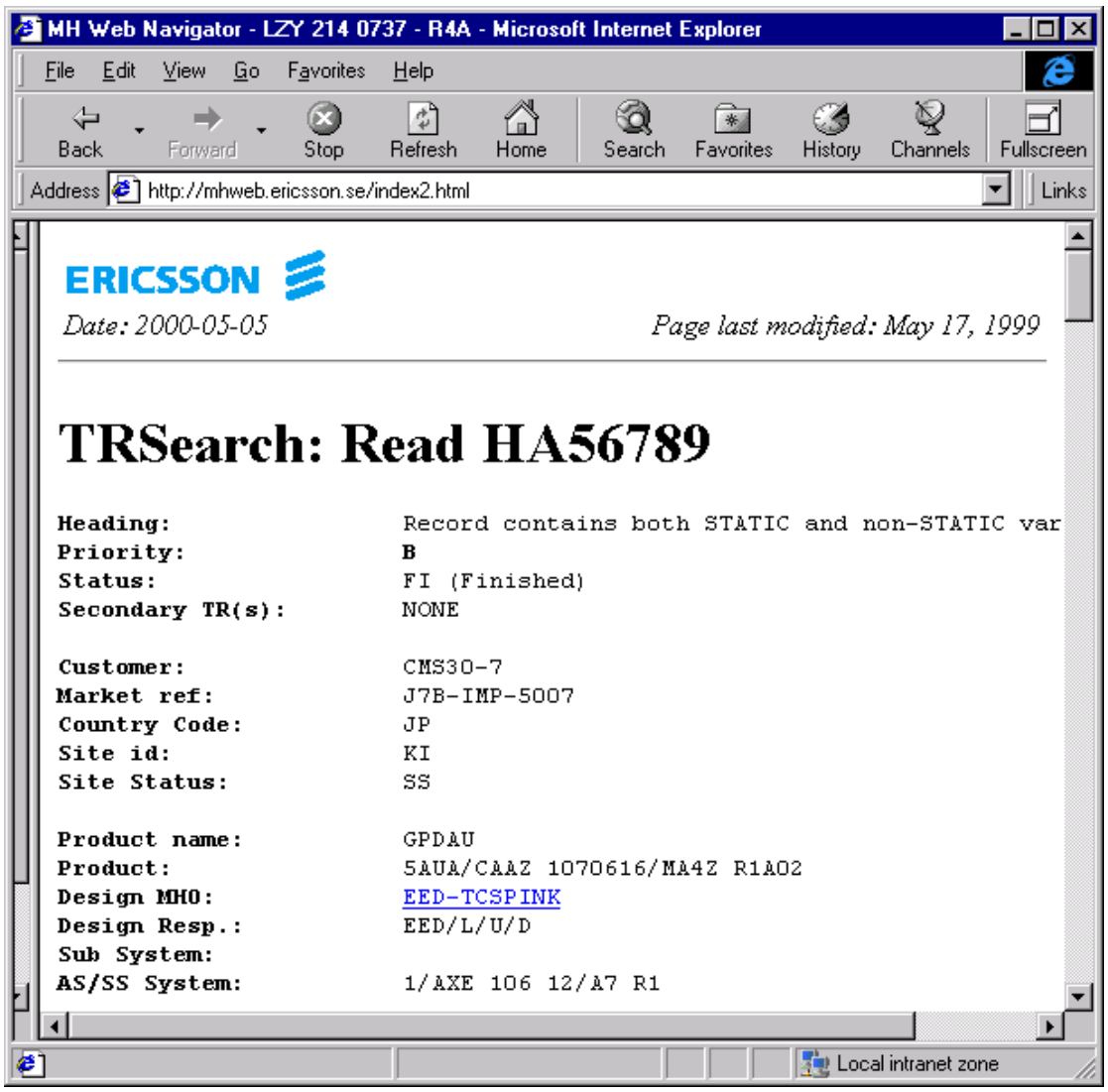


Figure 6.3: A trouble report.

The last action made is to remove the bean object. This is necessary because otherwise it would still allocate memory.

6.5 Jaguar CTS

We are using a non-local Jaguar server. Jaguar contains a connection cache that makes sure that each business object in the system can share database access. It is a preallocated connection to a remote database server that is used by the components. We have chosen to use a connection cache because it reduces the overhead in time of establishing connections. A connection is established once and reused for several components, instead of that each component creates a new connection. In our application we only use one Jaguar server and therefore no cluster handling is needed. Furthermore the server works as its own name server.

7 Other products

There exist other products that are similar to EJB; here we will mention some alternatives.

COM (Component Object Model) is Microsoft's framework for developing and supporting program component objects. It is aimed at providing similar capabilities to those defined in CORBA, a framework for the interoperation of distributed objects in a network. COM is an open software architecture for the cross-platform development of client/server applications based on object-oriented technology. COM is platform independent like the EJB technique. [6]

IBM SanFrancisco is a Java-based collection of components that allows developers to assemble server-side business applications from existing parts, rather than build from scratch. SanFrancisco is platform independent. [6]

It is also possible to create you own component model using servlets. This lead to difficulties in reusing components in other system, because of that no other system supports your components interface. It also takes much time in aspect to create everything from scratch.

8 Experiences and recommendations

During the development of the prototype and our application we have experienced some advantages and disadvantages that are described together with recommendations.

8.1 Jaguar CTS

Jaguar is a component transaction server that is aimed to work as a middleware. By using Jaguar the user is able to view TR from different systems using different protocols, and still get the same information from the database. This solves the problem that today a TR can only be viewed via html using http.

A good feature with Jaguar is that you slightly notice it when developing an application, because of the good integration with PowerJ. Most of the configuration of the component is made directly from PowerJ.

One problem with Jaguar is when managing the Jaguar server using the Jaguar manager tool you must have a local installation of Jaguar, because else you won’t be able to use the Jaguar manager. Alternatively you can administrate a Jaguar server that is running at a UNIX machine using a telnet utility, but that requires that you are familiar with UNIX.

The debug utilities provided by PowerJ were so bad that most of our debugging consisted of printing out lines telling which part of the application that was executing and the values of variables. These lines were then listed in a log file on the Jaguar server together with information listed by Jaguar during the execution of the program. This may be a problem when several developers are working towards the same Jaguar server, because they must keep in mind what prints are aimed for them.

In the beginning we used a local Jaguar server, because we thought that it would be easier that way. But in the end we started to use a non-local Jaguar server because we didn't notice any difference in the development process.

To work towards a non-local Jaguar server is what we would like to recommend because it will give more CPU-cycles and memory to be used for your computer in the development process. Another aspect is when using a non-local server you don't have to administrate the server yourself, you can have someone more familiar with Jaguar to manage it for you.

It is possible to use the Jaguar Manager for developing EJB but its not very good compared to PowerJ.

8.2 PowerJ

PowerJ is a graphical application development environment that is used for building Java programs for distributed, web and client/server applications. By using Java as programming language it is simpler and more convenient to make advanced applications, which is a problem when using CGI and Perl. Java also gives the benefit of platform independence, making it possible for Ericsson to work with both Windows NT and UNIX.

When developing an EJB developers can either choose to code the entire EJB themselves or by following a guide that is included in the software package. For the beginner we suggest you to follow the guide the first time you develop an EJB using the drag and drop technique. This gives you a good knowledge of how to use PowerJ. Another reason why a person should use the guide the first time is to see what possibilities PowerJ offers. We think that the guides are quite well performed, but we have noticed some problems concerning options in dialogs. The guide present options that are not equal to those actually displayed on the screen. Anyhow in most times you can guess what to choose. It should also be mentioned that in some cases there is lack of information.

After some time when you have developed one or several applications using the guides you could try to change the code yourself. Next step is to create the entire application on your own.

PowerJ has some restrictions, for example the first properties you set to the EJB can't be changed afterwards. This means that you should know from the beginning how these properties should be set. We have not found any list that shows what properties have been set and their values. Another less good aspect we have experienced is that some part of the code is hidden from the developer and can not be changed within PowerJ, instead you have to find the associated Java file to change these settings.

When developing we suggest using an already existing connection cache, instead of creating a new connection cache for each EJB.

Unfortunately when you deploy your components it takes a lot of time and the debugging possibilities are limited. We tried the built-in debugging possibilities but they were useless. The debug log gives no good information and it often refers to other things except to the real fault. Instead you should use the log-file that Jaguar generate to debug your code.

One important thing that is not mentioned in the guides is that if you change the skeleton you must copy the stub files from the PowerJ directory to the PowerDynamo directory, however you can make a batch-file which handles this.

To avoid copying the stub files so often vi recommend to create the methods that you think you might need in your EJB as early as possible. You can always change the logic afterwards, which don’t require that you update the stub files.

8.3 PowerDynamo

You could use the PowerDynamo Personal Web Server to test your web sites locally.

Personal Web Server is recommended to use when you develop and test your application. Therefor you don’t need two different UNIX machines one for Jaguar and another for PowerDynamo. Another aspect is that you only have to copy the stub files to the PowerDynamo directory instead of uploading them to a different machine.

This Web Server in not recommended when you decide to take your application in use. Mostly because it does not include so many advanced features, you have to do a lot of programming yourself. Another reason why we don’t recommend it is because it sometimes crashes and you must reboot the web server.

DynaScript is one of the great advantages with PowerDynamo because it is only executed at the server so no client support is needed. The client side logic for enterprise applications should be small and efficient to use the bandwidth in the best manner, which server side script like DynaScript ensures. Another benefit with executing advanced scripts at the server side is that it is much faster than executing the script on the browser machine. The only thing you must have installed on the client side is the web browser. On the contrary if the script were executed on the client side, it would require additional software to support the components (Java applets, ActiveX, or plug-ins). By using server side script it is easier to maintain information through a series of client accesses. It is also possible to include error checking to see if a command was successful. But you can't direct html to different frames in the client window.

There are several advantages of using a script language for an html-page instead of using a programming language. The use of a programming language is very good for large applications, but for a smaller application it can cause unnecessary overhead in data and instructions. It also gives a more readable code than the use of a programming language with many special built in features.

The editor included in PowerDynamo has a good way of presenting the syntax in different colors.

9 Conclusions

We believe that EJB and EA Server can solve the EIN/I problems, discussed in chapter 2.2. The requirement verification matrix in the system requirements specification [1] is achieved accept the requirement 2, 3 and 4. Requirement 2 couldn't been verified because we cant perform the test, but we believe that it should work. Why requirement 3 and 4 are not verified is due to that we don't know the formatting rules for a TR.

Just as the Java platform has changed the way we think about software development, we believe that EJB will change the way we think about business logic development. This because EJB solves most of the problems with the multitier model, discussed in chapter 3.2. It also combines server side components with distributed object technologies to greatly simplify the task of application development. EJB automatically takes into account many of the requirements of business systems: security, resource pooling, persistence, concurrency and transactional integrity.

We believe that continuous developing of the EA Server according to the EJB standard will make it to one of the future leading products in the area of developing enterprise systems.

References

[1] System Requirement Specification.

[2] Richard Monson-Haefel. Enterprise Java Beans. O'Reilly , first edition 1999.

[3] EA Server litterateurs who are included in the EA server software package.

[4] Peter van der Linden. Just Java. SunSoft Press A Prentice Hall Title, sec. edition 1997.

[5] http://java.sun.com/products/ejb

[6] http://www.whatis.com

A The PowerDynamo script

```
<HTML>
<HEAD>
<TITLE>TRSearch: Read $eriref</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF"><HR><H1>TRSearch: Read $eriref</H1>
<FORM NAME="read_tr">
<PRE>


<!--SCRIPT xml_to_html.ssc created on Wed, 03 May 2000 15:49:35
    import "system/utils/javaqry.ssc";

    var eriref = document.value.eriref;

    var EJBObjHome = java.GetHomeInterface("TRBrowse","iiop://einks663.ks.ericsson.se:9000","jagadmin");
    if(EJBObjHome==null)
    {
        document.writeln("<PRE>");
        document.writeln("Failed to getHomeInterface ");
        document.writeln(site.GetErrorInfo());
        document.writeln("</PRE>");
        exit(1);
    }

    var EJBObj = EJBObjHome.create();
    if( EJBObj == null)
    {
        document.writeln("<PRE>");
        document.writeln("Failed to create EJBObj:: ");
```

```
        document.writeln(site.GetErrorInfo());
        document.writeln("</PRE>");
        exit(1);
    }

    var xmlstr = EJBObj.getXML(eriref);

    function getXMLvalue(tag)
          {
        var start = xmlstr.indexOf("<"+tag+">")+2+tag.length;
                    var stop  = xmlstr.indexOf("</"+tag+">");
                    var value = xmlstr.substring(start,stop);
                    return value;
          }

          document.writeln ("<B>Heading:              </B>" + getXMLvalue("Heading"));
          document.writeln ("<B>Priority:             </B>" + getXMLvalue("Priority"));
          document.writeln ("<B>Status:               </B>" + getXMLvalue("Status"));
          document.writeln ("<B>Secondary TR(s):      </B>" + getXMLvalue("Secondary_TR"));
          document.writeln ("<B>                      </B>");
          document.writeln ("<B>Customer:                                 </B>" + getXMLvalue("Customer"));
          document.writeln ("<B>Market ref:           </B>" + getXMLvalue("Market_ref"));
          document.writeln ("<B>Country Code:         </B>" + getXMLvalue("Country_Code"));
          document.writeln ("<B>Site id:              </B>" + getXMLvalue("Site_id"));
          document.writeln ("<B>Site Status:          </B>" + getXMLvalue("Site_Status"));
          document.writeln ("<B>                      </B>");
          document.writeln ("<B>Product name:         </B>" + getXMLvalue("Product_name"));
          document.writeln ("<B>Product:              </B>" + getXMLvalue("Product"));
    document.writeln ("<B>Design MHO:           </B>" + "<A HREF=\"/cgi-
bin/TRsimple.pl?job_show_info1&=tag=" +getXMLvalue("Design_MHO")+"&type=mho\">" +
getXMLvalue("Design_MHO")+"</A>");
          document.writeln ("<B>Design Resp.:         </B>" + getXMLvalue("Design_Resp."));
          document.writeln ("<B>Sub System:           </B>" + getXMLvalue("Sub_System"));
          document.writeln ("<B>AS/SS System:         </B>" + getXMLvalue("AS/SS_System"));
          document.writeln ("<B>                      </B>");
```

```
          document.writeln ("<B>Current MHO:          </B> <A HREF=\"/cgi-
bin/TRsimple.pl?job_show_info=1&tag=" + getXML("Current_MHO") + "&type=mho\"></A>");
    document.writeln ("<B>Arrived to MHO:  </B>        ");
          document.writeln ("<B>Current User:         </B>  <A
HREF=\"http://people.ericsson.se/?x500userid=" + getXML("Current_User") +"\"></A>");
          document.writeln ("<B>                      </B>");
          document.writeln ("<B>Register Date:        </B>" + getXMLvalue("Register_Date"));
          document.writeln ("<B>Registered at MHO:    </B> <A HREF=\"/cgi-
bin/TRsimple.pl?job_show_info=1&tag=" + getXMLvalue("Registered_at_MHO") + "&type=mho\">" +
getXMLvalue("Registered_at_MHO") +"</A>");
          document.writeln ("<B>Registered by:        </B> <A
HREF=\"http://people.ericsson.se/?x500userid=" + getXMLvalue("Registered_by") + "\">" +
getXMLvalue("Registered_by") + "</A>");
          document.writeln ("<B>Prepared by:          </B>" + getXMLvalue("Prepared_by"));
          document.writeln ("<B>Prepares phone:       </B>" + getXMLvalue("Prepares_phone"));
          document.writeln ("<B>                      </B>");
          document.writeln ("<B>Additional info:      </B>" + getXMLvalue("Additional_info"));
          document.writeln ("<B>Comment:              </B>" + getXMLvalue("Comment"));
          document.writeln ("<B>                      </B>");

          document.writeln ("<B>APZ:                  </B>" + getXMLvalue("APZ"));
          document.writeln ("<B>APT/APR:              </B>" + getXMLvalue("APT/APR"));
          document.writeln ("<B>                      </B>");

          document.writeln ("<B>ISP (Design):         </B>" + getXMLvalue("isp"));
          document.writeln ("<B>Technical Answer Date:</B>" + getXMLvalue("Technical_Answer_Date"));
          document.writeln ("<B>Answer code:          </B>" + getXMLvalue("Answer_code"));

          document.writeln ("<B>Fault code:           </B>" + getXMLvalue("Fault_code"));

    EJBObj.remove();

-->

</body>
</html>
```

B The bean class

```
/*
    se-ericsson-ks-TR-TRBrowseBean.java
*/

package se.ericsson.ks.TR;


// custom imports for TRBrowseBean

import com.sybase.jaguar.jcm.*;
import java.sql.*;


public class TRBrowseBean extends java.lang.Object implements javax.ejb.SessionBean
{
    protected boolean create() throws java.lang.Exception
    {

        return true;
    }

    public TRBrowseBean()
    {
    }
    private void unhandledEvent( String listenerName, String methodName, java.lang.Object event )
    {
    }
    public void setSessionContext( javax.ejb.SessionContext parm0 ) throws java.rmi.RemoteException
    {
        this._sessionContext = parm0;
```

```
}
public void ejbActivate() throws java.rmi.RemoteException
{
}
public void ejbPassivate() throws java.rmi.RemoteException
{
      }
private javax.ejb.SessionContext _sessionContext;
/**
 * ejbCreate Method
 */

public void ejbCreate() throws java.rmi.RemoteException, javax.ejb.CreateException
{
    try
                {
        create();
                }
    catch( java.lang.Exception __e)
                {
        System.err.println("::Failed to create::" +__e.toString() + " " + __e.getMessage() );
    }
}
/**
 * ejbRemove Method
 */

public void ejbRemove() throws java.rmi.RemoteException
{
}
/**
 * exeQuery Method
 */

public java.sql.ResultSet exeQuery(String query,Statement stmt) throws java.rmi.RemoteException
{
```

```
      java.sql.ResultSet rs;

                try
                {
                          rs = stmt.executeQuery(query);
                }

                catch(Exception e1)
                {
                          System.err.println(e1.getMessage());
                          return null;
                }
                return rs;
}
/**
 * getXML Method
 */

public String getXML(String eriref) throws java.rmi.RemoteException
{
    String xmlString = null;
                JCMCache cache=null;
                Connection conn=null;
                Statement stmt=null;

                try
                {
                          cache = JCM.getCacheByName("solstam1");
                }
                catch(Exception e1)
                {
                          System.err.println(e1.getMessage());
                          return "failed to recive solstam1";
                }

                try
```

```
{
          conn = cache.getConnection(JCMCache.JCM_NOWAIT);
}
catch(java.sql.SQLException e1)         {
          System.err.println(e1.toString());
          return "failed to get connection";
}
catch(com.sybase.jaguar.util.JException e1)       {
}

try
{
          stmt = conn.createStatement();
}

catch(Exception e1){
          System.err.println(e1.getMessage());
          return "failed to create statement";
}

xmlString = query_1(eriref, stmt);
xmlString = xmlString + query_2(eriref, stmt);
xmlString = xmlString + query_3(eriref, stmt);
xmlString = xmlString + query_4(eriref, stmt);
xmlString = xmlString + query_5(eriref, stmt);
xmlString = xmlString + query_6(eriref, stmt);
xmlString = xmlString + query_7(eriref, stmt);
xmlString = xmlString + query_8(eriref, stmt);

try
{
          stmt.close();
          stmt = null;
          cache.releaseConnection(conn);
}
catch(Exception e2)
```

```
                    {
                              System.err.println(e2.getMessage());
                    }

                    return xmlString;
          }
          /**
           * Query_1 Method
           */

          public String query_1(String eriref,Statement stmt) throws java.rmi.RemoteException
          {
                String query ="select isnull(p1.prdes,p2.prdes), isnull(p3.desresmho,''),
isnull(p1.desresoffice,p2.desresoffice),
t.heading,t.addinfo,t.marketref,t.comment,t.priority,t.currmho,t.curruser,t.regmho,t.reguser,t.prepby,t.pr
epphone,t.siteid,t.customer,t.sitestat,t.country,t.subsystem,t.sysprnoi,t.sysprrevi,t.objecttype,t.rejecte
d,t.troubleref,t.trtype,t.lastanswrev,t.status,convert(char(12),t.mhodate,102),t.object,t.objectrev,z.trou
bleid,z.lastanswrev,convert(char(12),t.regdate,102),convert(char(12),t.techdate,102) from PRODUCT p1,
PRODUCT p2, PRODISSUE p3,TROUBLE z, TR t where (p1.prnoi=*t.object) and( p2.prnoi =* t.prnoi) and
(p3.prnoi =* t.prnoi) and (p3.prrevi =* t.prrevi ) and (z.troubleid=t.troubleref) and (t.objecttype='P')
and (t.eriref='" +eriref +"') union select 'DOCUMENT TR ','DOCUMENT TR ','DOCUMENT TR '
,t.heading,t.addinfo,t.marketref,t.comment,t.priority,t.currmho,t.curruser,t.regmho,t.reguser,t.prepby,t.p
repphone,t.siteid,t.customer,t.sitestat,t.country,t.subsystem,t.sysprnoi,t.sysprrevi,t.objecttype,t.reject
ed,t.troubleref,t.trtype,t.lastanswrev,t.status,convert(char(12),t.mhodate,102),t.object,t.objectrev,z.tro
ubleid,z.lastanswrev,convert(char(12),t.regdate,102),convert(char(12),t.techdate,102) from TROUBLE z, TR t
where (t.objecttype='D') and (z.troubleid=t.troubleref)and (t.eriref='" +eriref +"')";
                    java.sql.ResultSet rs = exeQuery(query, stmt);
                    if(rs ==null)
                              return null;

                    int i=1;
                    String xmlString= null;

                    try
                    {
                              if (rs.next())
```

```
{
                xmlString = xmlString +"<Product_name>"+ rs.getString(i++)
+"</Product_name>";
                xmlString = xmlString +"<Design_MHO>"+ rs.getString(i++)
+"</Design_MHO>";
                xmlString = xmlString +"<Design_Resp.>"+ rs.getString(i++)
+"</Design_Resp.>";
                xmlString = xmlString +"<Heading>"+ rs.getString(i++)
+"</Heading>";
                xmlString = xmlString +"<>"+ rs.getString(i++) +"</>";
                xmlString = xmlString +"<Market_ref>"+ rs.getString(i++)
+"</Market_ref>";
                xmlString = xmlString +"<>"+ rs.getString(i++) +"</>";
                xmlString = xmlString +"<Priority>"+ rs.getString(i++)
+"</Priority>";
                xmlString = xmlString +"<>"+ rs.getString(i++) +"</>";
                xmlString = xmlString +"<>"+ rs.getString(i++) +"</>";
                xmlString = xmlString +"<Registered_at_MHO>"+ rs.getString(i++)
+"</Registered_at_MHO>";
                xmlString = xmlString +"<Registered_by>"+ rs.getString(i++)
+"</Registered_by>";
                xmlString = xmlString +"<Prepared_by>"+ rs.getString(i++)
+"</Prepared_by>";
                xmlString = xmlString +"<Prepares_phone>"+ rs.getString(i++)
+"</Prepares_phone>";
                xmlString = xmlString +"<Site_id>"+ rs.getString(i++)
+"</Site_id>";
                xmlString = xmlString +"<Customer>"+ rs.getString(i++)
+"</Customer>";
                xmlString = xmlString +"<Site_Status>"+ rs.getString(i++)
+"</Site_Status>";
                xmlString = xmlString +"<Country_Code>"+ rs.getString(i++)
+"</Country_Code>";
                xmlString = xmlString +"<>"+ rs.getString(i++) +"</>";
                xmlString = xmlString +"<AS/SS_System>"+ rs.getString(i++);
                xmlString = xmlString + rs.getString(i++) +"</AS/SS_System>";
```

```
                                        xmlString = xmlString +"<Junk>"+ rs.getString(i++) +"</Junk>";
                                        xmlString = xmlString +"<Junk>"+ rs.getString(i++) +"</Junk>";
                                        xmlString = xmlString +"<Junk>"+ rs.getString(i++) +"</Junk>";
                                        xmlString = xmlString +"<Junk>"+ rs.getString(i++) +"</Junk>";
                                        xmlString = xmlString +"<Junk>"+ rs.getString(i++) +"</Junk>";
                                        xmlString = xmlString +"<Status>"+ rs.getString(i++) +"</Status>";
                                        xmlString = xmlString +"<Junk>"+ rs.getString(i++) +"</Junk>";
                                        xmlString = xmlString +"<Product>"+ rs.getString(i++);
                                        xmlString = xmlString + rs.getString(i++) +"</Product>";
                                        xmlString = xmlString +"<Junk>"+ rs.getString(i++) +"</Junk>";
                                        xmlString = xmlString +"<Junk>"+ rs.getString(i++) +"</Junk>";
                                        xmlString = xmlString +"<Register_Date>"+ rs.getString(i++)
+"</Register_Date>";
                                        xmlString = xmlString +"<Junk>"+ rs.getString(i++) +"</Junk>";
                                }
                        }
                        catch(java.sql.SQLException e)
                        {
                                System.err.println(e.getMessage());
                                return "Failed in SQL execution";
                        }
                        catch(java.lang.Exception err)
                        {
                                System.err.println(err.getMessage());
                                return "Failed to create String";
                        }
                        return xmlString;
                }
        /**
         * query_2 Method
         */

        public String query_2(String eriref, Statement stmt) throws java.rmi.RemoteException
        {
                String query ="select troubleref from TR where (eriref = '" +eriref +"')";
                        java.sql.ResultSet rs = exeQuery(query, stmt);
```

```
                    if(rs ==null)
                    {
                              return null;
                    }

                    int i=1;
                    String xmlString= null;

                    try
                    {
                              if (rs.next())
                              {
                                        xmlString = xmlString +"<Junk>"+ rs.getString(i++) +"</Junk>";
                              }
                    }
                    catch(java.sql.SQLException e)
                    {
                              System.err.println(e.getMessage());
                              return "Failed in SQL execution";
                    }
                    catch(java.lang.Exception err)
                    {
                              System.err.println(err.getMessage());
                              return "Failed to create String";

                    }
                    return xmlString;
    }
    /**
     * query_3 Method
     */

    public String query_3(String eriref, Statement stmt) throws java.rmi.RemoteException
    {
          String query ="select eriref from TR where (troubleref = '" +eriref +"') and (eriref != '"
+eriref +"')";
```

```
                    java.sql.ResultSet rs = exeQuery(query, stmt);
                    if(rs ==null)
                    {
                              return null;
                    }

                    int i=1;
                    String xmlString=" ";

                    try
                    {
                              if (rs.next())
                              {
                                        while(rs.getString(i) != null)
                                        {
                                                  System.err.println("::query_3 kor while slingan::");
                                                  System.err.println(rs.getString(i));
                                                  xmlString = xmlString +"<Product_name>"+
rs.getString(i++) +"</Product_name>";
                                                  System.err.println(rs.getString(i));
                                                  xmlString = xmlString +"<Design_MHO>"+ rs.getString(i++)
+"</Design_MHO>";
                                                  System.err.println(rs.getString(i));
                                                  xmlString = xmlString +"<Design_Resp.>"+
rs.getString(i++) +"</Design_Resp.>";
                                        }
                              }
                              else
                              {
                                        System.err.println("No rows found!");
                              }
                    }
                    catch(java.sql.SQLException e)
                    {
                              System.err.println(e.getMessage());
                              return "Failed in SQL execution";
```

```
                }
                catch(java.lang.Exception err)
                {
                        System.err.println(err.getMessage());
                        return "Failed to create String";
                }
                return xmlString;
    }
    /**
     * query_4 Method
     */

    public String query_4(String eriref, Statement stmt) throws java.rmi.RemoteException
    {
                String query = "select prnoapz, prrevapz, prnoapt, prrevapt, eriref from TR_P_AXE
where eriref='"     +eriref +"'";
          java.sql.ResultSet rs = exeQuery(query, stmt);
                if(rs ==null)
                {
                        return null;
                }
                int i=1;
                String xmlString=" ";
                try
                {
                        if (rs.next())
                        {
                                xmlString = xmlString +"<>"+ rs.getString(i++) +"</>";
                                xmlString = xmlString +"<>"+ rs.getString(i++) +"</>";
                                xmlString = xmlString +"<>"+ rs.getString(i++) +"</>";
                                xmlString = xmlString +"<>"+ rs.getString(i++) +"</>";
                                xmlString = xmlString +"<Junk>"+ rs.getString(i++) +"<Junk/>";
                        }
                }
                catch(java.sql.SQLException e)
                {
```

```
                        System.err.println(e.getMessage());
                        return "Failed in SQL execution";
                }
                catch(java.lang.Exception err)
                {
                        System.err.println(err.getMessage());
                        return "Failed to create String";
                }
                System.err.println("::query_4 slutar::");
                return xmlString;
    }
    /**
     * query_5 Method
     */

    public String query_5(String eriref, Statement stmt) throws java.rmi.RemoteException
    {
          String query = "select a.measurecode,c.answer,a.ispmark, a.troubleid,a.troublearev from TRANSWER
a,TRANSWERCODE c where (a.eriref='" +eriref +"') and (a.trarev=' AV ') and (a.measurecode = c.answercode
)";
          java.sql.ResultSet rs = exeQuery(query, stmt);
                if(rs ==null)
                {
                        return null;
                }
                int i=1;
                String xmlString=" ";
                try
                {
                        if (rs.next())
                        {
                                xmlString = xmlString +"<Answer_code>"+ rs.getString(i++);
                                xmlString = xmlString + rs.getString(i++) +"</Answer_code>";
                        }
                }
                catch(java.sql.SQLException e)
```

```
                    {
                              System.err.println(e.getMessage());
                              return "Failed in SQL execution";
                    }
                    catch(java.lang.Exception err)
                    {
                              System.err.println(err.getMessage());
                              return "Failed to create String";
                    }
                    return xmlString;
          }
     /**
      * query_6 Method
      */

     public String query_6(String eriref, Statement stmt) throws java.rmi.RemoteException
     {
          String query = "select w.faultcode,f.fault from TROUBLEANSWER w,TROUBLEACODE f where
(w.troubleid='"     +eriref +"') and (w.troublearev=' AV ') and (w.faultcode=f.faultcode)";
          java.sql.ResultSet rs = exeQuery(query, stmt);
                    if(rs ==null)
                    {
                              return null;
                    }
                    int i=1;
                    String xmlString=" ";
                    try
                    {
                              if (rs.next())
                              {
                                        xmlString = xmlString +"<Fault_code>"+ rs.getString(i++);
                                        xmlString = xmlString + rs.getString(i++) +"</Fault_code>";
                              }
                    }
                    catch(java.sql.SQLException e)
                    {
```

```
                        System.err.println(e.getMessage());
                        return "Failed in SQL execution";
                }
                catch(java.lang.Exception err)
                {
                        System.err.println(err.getMessage());
                        return "Failed to create String";
                }
                return xmlString;
    }
    /**
     * query_7 Method
     */
          public String query_7(String eriref, Statement stmt) throws java.rmi.RemoteException
    {
          String query = "select isnull(convert(char(10), b.custtargdate,102),''),
isnull(convert(char(10), b.destargdate,102),''),isnull(b.primtechuser,''),
isnull(b.primtechinfo,''),isnull(b.sectechuser,''), isnull(b.sectechinfo,''), isnull(b.trinfo,''),
isnull(b.ispmark,''), isnull(b.hotmark,'N') from TR_EXTENDED b where (b.eriref='" +eriref +"') ";
          java.sql.ResultSet rs = exeQuery(query, stmt);
                if(rs ==null)
                {
                        return null;
                }
                int i=1;
                String xmlString=" ";
                try
                {
                        if (rs.next())
                        {
                                System.err.println("::query_7 kor if slingan::");
                                while(rs.getString(i) != null)
                                {
                                        System.err.println(rs.getString(i));
                                        xmlString = xmlString +"<Answer_code>"+
rs.getString(i++);
```

```
                                                System.err.println(rs.getString(i));
                                                xmlString = xmlString + rs.getString(i++)
+"</Answer_code>";
                                        }
                                }
                                else
                                {
                                        System.err.println("No rows found!");
                                }
                        }
                        catch(java.sql.SQLException e)
                        {
                                System.err.println(e.getMessage());
                                return "Failed in SQL execution";
                        }
                        catch(java.lang.Exception err)
                        {
                                System.err.println(err.getMessage());
                                return "Failed to create String";
                        }
                        System.err.println("::query_7 slutar::");
                        return xmlString;
        }
    /**
     * query_8 Method
     */

    public String query_8(String eriref, Statement stmt) throws java.rmi.RemoteException
    {
                        String query = "select corr.corrid from CORRDATA corr, TROUBLEAAC aac where
(aac.troubleid='" +eriref +"') and (aac.troublearev=' AV ') and (aac.acid=corr.corrid) ";
                java.sql.ResultSet rs = exeQuery(query, stmt);
                        if(rs ==null)
                        {
                                return null;
                        }
```

```
                int i=1;
                String xmlString=" ";
                try
                {
                        if (rs.next())
                        {
                                while(rs.getString(i) != null)
                                {
                                        System.err.println(rs.getString(i));
                                        xmlString = xmlString +"<Answer_code>"+
rs.getString(i++);

                                        System.err.println(rs.getString(i));
                                        xmlString = xmlString + rs.getString(i++)
+"</Answer_code>";
                                }
                        }
                        else
                        {
                                System.err.println("No rows found!");
                        }
                }
                catch(java.sql.SQLException e)
                {
                        System.err.println(e.getMessage());
                        return "Failed in SQL execution";
                }
                catch(java.lang.Exception err)
                {
                        System.err.println(err.getMessage());
                        return "Failed to create String";
                }
                System.err.println("::query_8 slutar::");
                return xmlString;
        }
 }
```

C The remote interface

```
/*
    se-ericsson-ks-TR-TRBrowseRemote.java
*/

package se.ericsson.ks.TR;


public interface TRBrowseRemote extends javax.ejb.EJBObject
{
    /**
     * getXML Method
     */

    public String getXML(String eriref) throws java.rmi.RemoteException;

}
```

D The home interface

```
/*
    se-ericsson-ks-TR-TRBrowseHome.java
*/

package se.ericsson.ks.TR;


public interface TRBrowseHome extends javax.ejb.EJBHome
{
    /**
     * create Method
     */

    public TRBrowseRemote create() throws java.rmi.RemoteException, javax.ejb.CreateException;

}
```