Development And Evaluation Of A Scalable, Fault Tolerant Telecommunications System Using EJB And Related Technologies

Oisin Kim

Department Of Computer Science

Trinity College Dublin

A dissertation submitted to the University Of Dublin,
in partial fulfilment of the requirements for the degree of
Master of Science in Computer Science

September 16, 2001

Declaration

I declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed: ____________________

Date: September 16, 2001

Permission to lend and/or copy

I agree that Trinity College Library may lend or copy this dissertation upon request.

Signed:	__________________
Date:	September 16, 2001

Acknowledgments

Dedicated To The Memory Of My Nana, Agnes Kelly.

I would like to thank my supervisor, Dr. Andrew Nisbet for guidance and assistance in the completion of this dissertation, especially those outside of office hours that I stole from him.
Thanks go to Declan Vize, Brian Kelly, Eoin Cavanagh, Pierre-yves Baloche and all the staff at Logica Mobile Networks for their help and advice.
Thanks go out to all the members of the MSc NDS class of 2001, thanks to you all for your advice and teamwork over the year.
Thanks go to members of the Trinity College Department, especially;
All the MSc NDS lecturers sharing their knowledge with me.
Simon Dobson, for proof reading, advice and instruction.
Vinny Cahill, for his advice and help over the year.
Stefan Weber, for his help in configuring the Linux Cluster.
Tom Kearney, for his hardware help and networking advice.
Peter Barron, for his help with Java.
Ray Cunningham for his help with Java.
Colin Harris, for his help with EJBs.
Rachel Noctor, for all her help finding old thesis and reports.
Rosemary Welsh, for all her help getting licenses for software.
Thanks go to members of the Irish Linux Users Group, this community had answers for me outside of office hours, late at night and weekends on many linux related questions.
Thanks go to members of the Orion Interest Mailing list, especially one Juan Pablo Lorandi. Gracias.
To my friends for their advice over the year, yes I really am finished college!
Thanks go to my little brothers Jong and Howard for their advice and proof

reading of this dissertation, any errors or ommissions that remain are totaly my own fault.
Last but certainly not least, thanks to my mum, for her unending love, support and guidance, her faith in me over the last twenty six years is the reason I am where I am today. I could never repay you, but I hope you know that I realise the sacrifices you have made to get me here, thanks.

Abstract

The objective of the research project is to investigate how Sun's Enterprise Java Bean technology (EJB) and Java 2 Enterprise Edition (J2EE) could be used to produce a prototype Short Message Service Centre (SMSC) which will have the following attributes:

- No single point of failure.
- Load sharing between machines in the cluster.
- High availability.
- Scalablity.

The production of the prototype demonstates the benefits and cost savings, in terms of re-use of existing infrastructure, of using the EJB architecture as a starting point for development of complex enterprise systems. We achieved this by producing a prototype with qualities as listed above. A basic performance evaluation of the prototype SMSC was then carried out and we document possible bottlenecks in the system along with some possible solutions.

This project was funded and supported by Logica Mobile Networks.

Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction

The SMS is the most popular of all mobile data type services. Every month in Europe alone, billions of short messages are sent and received. In fact, in the UK over one billion messages were sent in the month of August 2001 according to the Mobile Data Association [Ass01].

In todays modern telecommunications systems 'five nines' reliability is the expected norm. Five Nines is terminology used to described systems which require a high degree of Reliability, Availability and Serviceability (RAS). Five nines RAS means that 'servers (and the data they serve to the enterprise) are up and running 99.999 percent of the time. Computed out, that means those servers are down for a total of about five minutes a year' [New00].

The use of hardware and software clusters of systems has grown in recent years. The worlds most popular search engine at the present time is Google, it provides its service using a cluster of over 10,000 Linux machines [Web01].

The primary method followed to provide systems with high levels of RAS, is to replicate the service and / or provide redundant backups to servers running in the system. This project uses replication of services / servers without having any redundant services / servers. This is preferential over redundant systems as it mean we can theoretically use machines closer to their physical limits. We describe a cluster of machines which act together to provide the SMSC with high RAS levels.

The project is an industry sponsored MSc Dissertation which investigates the possible use of the Java 2 Enterprise Edition (J2EE)[Micd] architecture to produce a system which will offer the basic services of a Short Message Service Center (SMSC). The sponsors Logica Mobile Networks Limited, have made available documentation, hardware and technical support through a series of meetings held at Logica's offices at the Customhouse Plaza in Dublin.

The goals of the project were to address real issues in computing today, to investigate the use of powerful and relatively inexpensive hardware platforms coupled with low or no cost software solutions. The Dissertation produced a prototype SMSC written in Enterprise JavaBeans (EJB) [Micc], JavaServer Pages (JSP)[Micj] and Java Remote Method Invocation (RMI)[Mich]. The system was then critically analysed and time was taken to identify possible bottle necks in the system and partial solutions were offered to these problems.

The system was deployed on a cluster of four workstations running the GNU / Linux operating system. Sample random input to the system was created on two Departmental machines on the local network which submitted messages to the SMSC system as fast as the system could handle them.

1.2 Qualities Of High Degree RAS Systems

The architecture of the system was created with four main goals as were mentioned in the abstract of this Dissertation; they are the standard requirements for High degree RAS Systems. Here we describe how our system should map to these qualities.

1.2.1 No Single Point Of Failure.

Transaction processing must survive failures of individual cluster machines, failure of LAN cards or Ethernet network failure. The system must be able to handle faults within any component in the system. This was achieved using the replication of system functionality and state within areas of the cluster called *islands*. An *island* is a subsystem of a cluster which shares functionality, state and data. We can afford to loose, through fault, all members of an island apart from one, and the system will still be operational. For the purposes of the prototype, it had been specified that the system should survive one failure at least, therefore an island size of two nodes per island was fixed. Further resilience to failure could be achieved by adding more nodes to each island.

1.2.2 Load Sharing.

Transaction processing is spread between available cluster members. Load sharing adjusts dynamically as platforms join or drop out of the cluster.

The separation of the Input Router or load sharing component from the SMSC messaging processing functionality enabled the removal of the Input Router component without disturbing the SMSC cluster itself. In addition it removed the cost of creating sample random SMS messages for testing purposes and also allowed the easy modification of routing algorithm chosen for the prototype. All that is required to modify the load sharing algorithm is to modify one method per island. Future versions could easily change this to one method for a network topology that wasn't fixed. Round Robin sharing was implemented for the prototype.

1.2.3 High Availability / Failover.

It is possible to add / remove platforms without stopping transaction processing. Software upgrades can be achieved without bringing the system out of service. The commercial system, as produced by Logica Mobile Networks, called Telepath, is world renowned for its reliability, satisfying the 'five nines' requirement. The prototype has been designed to allow for the hot addition or subtraction of both Input Routers and Nodes from an initially fixed network topology. This would allow an administrator of the system to choose machines to reboot or upgrade as necessary. The prototype has been designed to allow for loss of at best 2 nodes out of the initial 4, if each Island looses one node. For an island size of N, we can afford to loose at best N-1 nodes at most and still have a functioning system.

1.2.4 Scalability.

It is possible to increase performance by adding additional platforms. This can be done without bringing the system out of service. Theoretically infinite scalability has been achieved in the system by following a concept called a ‘shared nothing’ architecture. By this, we mean that sections of the system can operate independently of one another as they never share any state or data. In the prototype this is achieved by identifying islands in the network topology as mentioned earlier. We can scale the system by simply adding more islands. Moreover we have shown, for small networks, the performance of the system, as measured by the number of messages a second it can process grows almost linearly with the addition of more islands.

The dynamic addition of new nodes has been made trivial, simply by starting the nodes and then adding the nodes in pairs by notifying the external interface to the system, called the Input Router, of the locations of the new nodes. As mentioned earlier, the initial network topology was fixed so they were hard coded into the Input Router.

Through the use of the Orion Console Administrator, which comes as a standard but beta product with the release of Orion Server 1.5.2 [Sof], it is possible to remotely administer all network nodes through a simple to use graphical user interface. In addition, a JavaServer Pages (JSP)[Micj] interface to the system was created which, could be used to perform administration tasks on the fly. In future versions this could also easily allow for the updating of deployed classes through the use of custom class loaders and other run time administration. Both the administration tools have simple login facilities. The JSP solution has the added benefit of allowing the administrator to use SSL [ea01]to ensure a secure connection. As mentioned earlier the Orion Console Application is a beta product and had some issues, including one particular important one, all username and passwords are kept in a plaintext configuration file.

The prototype produced in this dissertation had many qualities that were desirable, including:

- An highly scalable architecture.
- Fully compliant A.C.I.D. Transactions [HR83], for critical methods.
- Remote administration, through the use of not only the Orion System Console, which allows remote administration of the entire cluster through a graphical user interface, but also via a JSP interface which give the added benefit of having a secure connection to a machine by the use of Netscape's Secure Sockets layer (SSL) technology [ea01].
- Rapid development time, due to the re-use of existing Application Programmer interfaces (APIs) and also by leveraging the existing functionality of the EJB / J2EE architecture.
- Low cost. (In fact future versions could run on a completely no cost platform by use of open source software, such as JBoss [JBo01]).
- Acceptable performance, when considering performance relative to the total cost of ownership of a cluster in terms of hardware and software and also to the mean demand on such as server.

1.3 The Short Messaging Service (SMS) and SMS Centers (SMSC).

SMS, as defined within the GSM digital mobile phone standard [Ins01] has several unique features, not all of which were implemented by the prototype:

- A single short message can be up to 160 characters of text in length. Those 160 characters can comprise of words or numbers or an alphanumeric combination.

- Non-text based short messages for simple pictures and graphical information are possible.

- The Short Message Service is generally a 'store and forward' service, in other words, short messages are not sent directly from sender to recipient, but always via an SMS center.

- Each mobile telephone network that supports SMS has one or more messaging centers to handle and manage the short messages. The prototype will be a very basic implementation of such a center. Logica's current production Short Message Service Center (SMSC) is called Telepath, the soon to be released next generation is called Picasso 1000. At the time of writing Logica Mobile Networks Telepath SMSC is the clear world market leader [Gro01].

- The Short Message Service can feature confirmation of message delivery. The sender of the short message can receive a return message back notifying them whether the short message has been delivered or not.

- There are multiple ways of sending multiple short messages available. Stringing several short messages together (so called SMS concatenation) and getting more than 160 characters of information within a single short message (so called SMS compression) have been defined and incorporated into the GSM SMS standards.

To send or receive a SMS a user needs a destination to send a short message to, or receive a message from. This is usually another mobile phone but may be a fax machine, PC or Internet address. The SMS Center is the server side software that accepts for delivery messages from Message Service Centers (MSC) which have already accepted messages from External Short Message Service Entities (ESME). The SMSC does all the necessary processing and delivers the SMS to the destination. Figure 1.1 shows the architecture of a basic SMSC system.

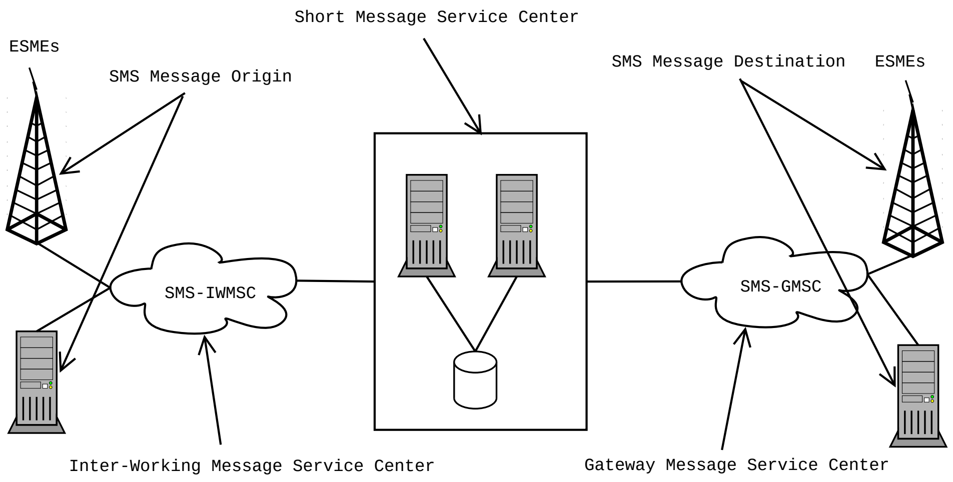

Short Message Service Center
ESMEs
SMS Message Origin
SMS Message Destination
ESMEs
SMS-IWMSC
SMS-GMSC
Inter-Working Message Service Center
Gateway Message Service Center

Figure 1.1: System Architecture Of A Basic SMSC.

1.4 Goals

To recap, the goals of the Dissertation are:

- To create a clean modular design.
- To produce a prototype SMSC using EJBs and related technologies on a low or no cost architectural platform.
- The evaluation of the resultant system.
- The identification of bottlenecks in the system.

- The proposal of some possible solutions that would solve or reduce their effects.

1.5 Thesis Outline

Chapter 1 *Introduction* attempts to introduce the reader to some of the technology in the SMS area, the motivation for the project, what the goals of the project were and what was achieved by it.

Chapter 2 *State of the Art* reviews the state of the art in the fields of cluster based systems and enterprise component based architectures. This includes an in depth review of EJB technology as this is the primary component of the system architecture.

Chapter 3 *Requirements, Specification and Design* gives a detailed Requirements Specification and gives an overview of how components of the system architecture interact with each other in normal system operation. It also signifies subsections or components of the system that have been omitted due to time restrictions.

Chapter 4 *Implementation* describes the implementation of the system. It details how particular design features were implemented in the system. Small simple code examples are described to reinforce points.

Chapter 5 *Evaluation* attempts to provide a analysis of the prototype and the performance of the prototype using benchmarking data collected by running tests on the system.

Chapter 6 *Conclusion* reviews the the goals and how they were met, it then suggests which areas of the system warrant future work and research.

1.6 Summary

A basic introduction into Short Messaging Centers and related technology has been provided in this chapter. In addition an attempt has been made to introduce the reader to some of the terminology used in describing the prototype system along with a description of the goals of the project. The following chapter will attempt to outline some of the technologies available today and define the state of the art and gives a review of the literature available.

Chapter 2

Enterprise JavaBeans

2.1 Introduction

In this chapter, we examine the state of the art in distributed object technology and component based systems. We start with a review of what exactly the requirements of todays Enterprise systems are, we then follow with a short appraisal of the three main distributed object technologies available to developers. We then focus our attention on the EJB architecture specifics.

2.2 Enterprise Computing

Enterprise Computing can be defined as systems programming that spans multiple machines where different machines or group of machines can focus on different facets of an enterprise. The goals of Enterprise computing are many, and should become apparent as this chapter continues.

2.2.1 Requirements

Roman [Rom99] mentions the following requirements of enterprise systems:

- Remote method invocations. We need logic that connects a client and server together via a network connection. This includes dispatching method requests, brokering of parameters, and more.

- Load-balancing. Clients must be fairly routed to servers. If a server is overloaded, a different server should be chosen.

- Transparent fail-over. If a server crashes, or if the network crashes, can clients be re-routed to other servers without interruption of service? If so, how fast does fail-over happen? Seconds? Minutes? What is acceptable for your business problem?

- Back-end integration. Code needs to be written to persist business data into databases, as well as integrate with legacy systems that may already exist.

- Transactions. What if 2 clients access the same row of the database simultaneously? Or what if the database crashes? Transactions protect you from these issues.

- Clustering. What if the server contains state when it crashes? Is that state replicated across all servers, so that can clients use a different server?

- Dynamic redeployment. How do you perform software upgrades while the site is running? Do you need to take a machine down, or can you keep it running?

- Clean shutdown. If you need to shut a server down, can you do it in a smooth, clean manner so that you don't interrupt service to clients who are currently using the server?

- Logging and auditing. If something goes wrong, is there a log that we can consult to determine the cause of the problem? A log would help us to debug the problem so it doesn't happen again.
- Systems Management. In the event of a catastrophic failure, who is monitoring our system? We would like monitoring software that paged a system administrator in case of these catastrophes.
- Threading. Now that we have many clients connecting to a server, that server is going to need the capability of processing multiple client requests simultaneously. This means the server must be multi-threaded.
- Message-oriented middleware. Certain types of requests should be message-based where the clients and servers are very loosely coupled. We need infrastructure to accommodate messaging.

2.3 Distributed Object Technology

Today, Distributed Object Technology (DOT) has taken three main, but largely different, paths. DOT enables different parts of a system to be located in different locations, where they make more sense. In other words it allows business logic to be reached from remote locations [Hae00]. Examples of DOT are Java Remote Method Invocation (RMI)[Mich], Microsoft's Distributed Common Object Model (DCOM)[Mica] (now becoming the .NET platform[Micb]) and the Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA)[Gro]. Distributed objects have evolved from a legacy form of three-tier architecture, which is used in Transaction Processing (TP) monitor systems such as IBM's CICS [IBM]or BEA's TUXEDO[BEA]. These systems attempt to separate the presentation, business logic and database tiers. They provide a simple wrapper for logging transactions.

2.4 Component Based Systems

Booch [Boo98] defines a component as ‘a physical and replaceable part of a system that conforms to and provides the realisation of a set of interfaces’. The interesting point made is use of the word replaceable, if we consider this, this means we can build our systems using Components and replace the parts of the system that don’t operate as we would have wished, with a more correct solution, even if this means going to a third party to purchase a pre-packaged solution or getting in a sub-contractor or specialist in this area to create the necessary code. [Hop00] states ‘A software component is a physical packaging of executable software with a well-defined and published interface’. Hopkins states that this definition is ‘formed from several other published definitions’. Component Based Systems are partial solutions to problems. They allow the developer to leverage the existing architectural platform or employ components created by third parties to reduce the development time and/or increase the quality of a system. [Lar00]

2.4.1 The Goal Of Component Based Design

‘Component-based software engineering also has to do with raising the level of abstraction for software development. The progression from machine language to assembly language to higher level languages has drastically increased productivity because the number of lines of code produced per unit time is roughly independent of the level of the language. Thus a day of work writing code in a higher level language produces much more functionality than a day

of work writing code in low level language.' [ST96]
Simply put the goal of component based design is to enable developers to be more productive. Some of the after effects are that by re-using tested components, quality is improved, possibly in terms of performance or robustness.

2.5 Enterprise JavaBeans And The Java 2 Enterprise Edition Overview

Sun's Enterprise JavaBean (EJB) distributed component programming model together with the Java 2 Enterprise Edition (J2EE) Platform has recently been touted as a 'silver bullet' for enterprise level distributed applications by Sun and it's devotees. The expectation exists in some papers and articles [RO01] that the evolutionary and open way in which the technology has been developed has enabled Sun to define a more practical architecture which will stand up to the rigors of industrial use. Conversely, its detractors state that there can be no silver bullet for distributed applications and that many components in the J2EE architecture are simply 'bloatware'. Here, we delve into the mechanics of EJBs.

2.6 Enterprise JavaBeans

EJBs components come in three architecturally different types: entity, session, and message-driven (MDB). The project used version 1.1 beans and this excluded the message-driven type beans from selection. [1] Session and

[1] At the time of writing it was considered that MDB were too unstable to be considered for a production system. However, some of the qualities of MDBs are impressive and there is no doubt that they could play a significant part in future enterprise systems.

entity beans are basically Java RMI based server-side components that are accessed using standard DOT protocols. The message-driven bean, is an asynchronous server-side component that is used to respond to Java Message Service (JMS)[HC01] messages. [Hae00] states 'A good rule of thumb is that entity beans model business concepts that can be expressed as nouns. For example, an entity bean might represent a customer, a piece of equipment, an item in inventory, or even a place. In other words, entity beans model real-world objects; these objects are usually persistent records in some kind of database.'

Session beans should be considered an extension of the client application and are responsible for dictating the flow of control of a task or process. Another analogy could be that made is that the relationship between entity and session beans is that which is between a script and the actors in a play. Where the actors are the entity beans and the script the session beans. Entity beans provide methods for doing things directly to the data in an object but doesn't say anything about the context under which those actions are taken. Some tasks require that we use a entity bean and also a lot of things that have nothing to do with the bean. In this way we are using many different beans to achieve a single goal, this is a prime example of where we would use session beans to direct the flow of a task. Session beans usually represent a particular action or activity. They deal with the relationships between different enterprise beans.

The task that a single session bean achieves is transient, you start doing something, it does some processing, it returns its result and then it's finished. In this way session beans are not unlike an Application Programmer

Interface (API). These beans do not represent persistent things which need to exist after we are finished working with the bean. If we do need to modify a persistent item, we do so by manipulating entity beans.

2.6.1 Entity Beans

Entity beans model data and operations on that data. They attempt to provide a clean, reusable and consistent interface to data stored in a database. Operations on an entity beans are sometimes referred to as business rules since these methods define the business logic behind how a business works. Take for example, a cinema, a simple entity bean could be used to model a ticket, and a simple method of a ticket could be a method which returns a boolean called `purchase`. Here we are taking the actual logic of what an action is doing and implementing it directly in code in a clear, consistent and most importantly easy to understand manner. We should also note that entity beans can model the relationships beans have with one another.

Entity beans can also be shared by multiple clients. In this way multiple clients can interact with the bean, and the operations and data relating to a particular bean can be used concurrently by those multiple clients. The EJB specification ensures that entity beans will keep in sync with updates from clients. [Hae00] states that 'Entity beans are designed to service multiple clients, providing fast, reliable access to data and behavior while protecting the integrity of data changes. Because entity beans are shared, we can rest assured that everyone is using the same entity and seeing the same data as it changes. In other words, we don't have duplicate entities with different representations of the same data.'

2.6.2 Session Beans

It has been stated previously that entity beans are useful for objectifying concepts that are nouns or things but they have difficulty describing work flow or tasks using beans. In other words entity beans don't give much information on the context under which actions on a bean might be used. The EJB architecture and specification encourages us to keep business logic away from the client, instead we'd like have a clean multi-tiered system. As tasks span more than one bean especially, it would be incorrect for us to define methods in our entity bean that interact with other data that the bean doesn't represent. We create session beans when we wish to model some particular task or process that can if we wish span multiple entity beans interacting with them and controlling the work flow in a consistent and correct manner.

Session beans act as a script for a client which will carry out a set out function or set of functions which work at manipulating the flow of execution. Session beans are not persistent like an entity bean, although they can carry out activities which will have persistent effects on the system. The activities of a session bean are carried out in a transient manner and when we think of it make sense, session beans are modelling tasks not data. They use entity beans to modify data but they themselves do not have a store of data in a database or file. Session beans model the workflow of a system.

Stateful/Stateless

There are two different types of session beans; stateful and stateless. Stateful session beans maintain conversational state with each client that uses them. Conversational state is not persistent or written to disk. It is cached in the memory of the EJB server while clients are using it. This is part of the bean-container contract which will be discussed later. Conversational state allows a client and bean pair to maintain a rapport, and be aware of the others state. As methods are invoked on a bean, its quite possible that the state of the bean will change, and that this will affect the workflow as the results of method invocations may depend on the current state of a bean. Perfect examples of these dependencies are typical set methods for beans. After data members have been set, the bean will return different values depending on what exactly has been set. These set methods represent the bean at a certain state at a particular time. The data transferred by this rapport, called conversational state, is only valid while a client is actively communicating with a bean. Once the client shuts down or releases it's reference to a session bean, the rapport is ended and any state shared by the pair is now lost. In this way concurrent access to beans is prohibited, as a bean belongs to it's client for the duration of it's communication.

Stateless session beans do not maintain any conversational state or rapport with a client. Every invocation of a method is done as a single entity and ends when the method is finished or when a result is passed back to the client. In this way stateless session beans act as an Application Programmer Interface (API). Since the stateless session bean doesn't need to maintain any conversational state it is thus more efficient and has a lower overhead than other types of beans, they provide the highest performance in terms of throughput and resource consumption compared to entity and stateful session beans because only a few stateless session bean instances are needed to serve hundreds, possibly thousands of clients.[Hae00]

2.6.3 Container

A container as described by [MH99] ‘is a system that functions as the ‘container’ for enterprise beans. Multiple beans can be deployed in the same container. The container is responsible for making the home interfaces of its deployed enterprise beans available to the client through Java Naming and Directory Interface (JNDI) API extension. Thus the client can look up the home interface for a specific bean using JNDI API’[Micf].

2.6.4 Remote Interface

The remote interface of an enterprise bean defines the methods that relate to business activities that we wish to make available to clients. The methods a bean defines in the remote interface are similar to the methods defined in a Java RMI interface class. This is because they are essentially the same thing. This is shown when we examine first the class from which the remote interface of a EJB must extend, `javax.ejb.Remote`. This in turn extends `java.rmi.Remote`. This interface is used by entity, session, message-driven, stateful and stateless beans alike.

2.6.5 Home interface

The home interface of an enterprise bean extends `javax.ejb.EJBHome`, which in turn extends `java.rmi.Remote`. [MH99] describes the home interface in the following way:
A container implements the home interface of the enterprise bean installed in the container. The object that implements a bean’s home interface is called a session or entity EJBHome Object as previously mentioned. The container makes the bean’s home interfaces available to the client through the Java

Naming And Directory Interface (JNDI) API.
The functionality of a remote home interface allows a client to do the following:

- Create a new bean.
- Remove a bean.
- Obtain a handle for the home interface.
- Get the `javax.ejb.EJBMetaData` interface for a bean.
- Entity beans may also be located through their home interface.

2.6.6 Bean class

The bean class name is usually appended with 'EJB' or 'Bean'. It actually contains the implementation for the EJB's business methods. [Hae00] states that we should note that 'the bean class for session and entity beans usually does not implement any of the bean's component interfaces directly.' What is required is that it has methods that match the signatures of those defined in the remote and local interfaces and some of the methods defined in the both the remote and local home interfaces. Entity beans implement `javax.ejb.EntityBean` and session beans implement `javax.ejb.SessionBean`. Both the `EntityBean` and `SessionBean` classes extend `javax.ejb.EnterpriseBean`.

This might sound confusing, but it becomes much clearer when we actually develop beans, we are simply enforced to ensure that if we declare methods in our remote interface, we need to ensure that the same methods exist in the actual implementation class. For example, if we have a method in our home interface that declares it possible to create a EJB with one parameter, say an int, then we should have a corresponding method called `ejbCreate` that takes one parameter that is an int. We can go and simplify things a step further and argue that an entity bean is just a representation of data in a datastore with a collection of finder and business methods and a session bean is simply a representation of actions or tasks on that data in the datastore.

2.6.7 Primary key

Every entity bean has a unique identity within the EJB server. This identity is ascertained by its primary key. Two entity beans are equal if their primary keys are equal. The EJB Specification [MH99] allows a primary key to be any class that is a legal Value Type in RMI-IIOP[Mick]. The primary key class may be specific to an entity bean class (i.e. each entity bean class may define a different class for its primary key, but it is possible that multiple entity beans use the same primary key class). A client can reference an entity bean if it knows its primary key using the method `findByPrimaryKey()` which returns a reference to an entity object or `null`.

2.6.8 Persistence

There are two methods of entity persistence a bean developer can choose from, they are Container managed and Bean managed persistence. Persistence is can be described as the data access protocol for transferring data or state belonging to entity beans and the datasource. This section describes them both briefly.

Container Managed Persistence

Container managed persistence is the model chosen for the prototype, it was chosen because it appeared significantly faster than the bean managed model described next. In this model, the developer does not have to worry about writing Standard Query Language (SQL) code and executing it. The developer simply described the data members of an entity bean wishes to be made persistent and then allows the container to look after the construction of tables, and ultimately the reading and writing of entity bean state from / to the datasource.

Bean Managed Persistence

Bean managed persistence forces the developer to write the necessary SQL code for table creation, reading and writing from the database. The datasource is accessed directly by Java DataBase Connectivity (JDBC)[Mice] / SQL calls which the developer hard codes into the bean at development time.

2.6.9 Instance Pooling

Instance pooling is a technique used to improve the performance of EJB servers. It is a relatively common practice, many vendors supply products which pool database connections as the creation of a database connection is an expensive operation. By having a pool of connections created and available to all to use, the developer can conserve expensive resources. Some EJB servers employ a technique called instance pooling which reduces the number of EJB objects running on a server. The technique is identical to database connection pooling, by reducing the number of actual instances of EJB objects instantiated on a server, the server uses less resources. Figure 2.1 shows how an EJB server pools database connections and EJB objects.

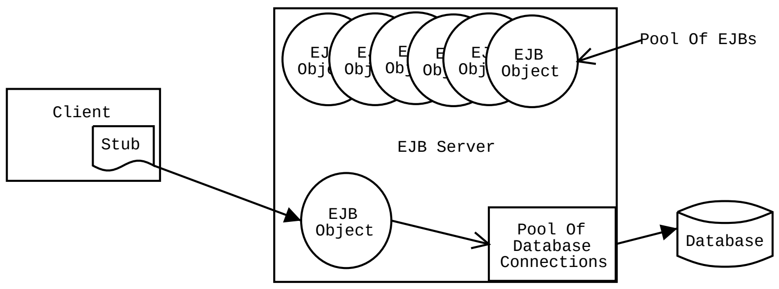


Figure 2.1: EJB Server Pooling.

2.6.10 Concurrency

Session Beans

Session beans do not support concurrent access by clients, the reasoning for this is that they represent actions or tasks, only one client can invoke the task at a time. In addition, stateful beans only interact with one client who 'owns' them, concurrent access doesn't make sense here. Since stateless beans don't share state, they don't need to be concurrent.

Entity Beans

As described earlier entity bean is just a representation of data in a datastore with a collection of finder and business methods. This data must be available to concurrent users of the system. Multiple clients must be able to create, read or update entity beans concurrently. To enforce the integrity of the data an entity bean represents the container needs to ensure all clients have versions of beans that are in sync. Simultaneous access to beans may corrupt the data each client holds, to ensure that this doesn't happen the EJB specification [MH99] by default prohibits concurrent access to the same particular entity bean instance, while multiple clients can have references to a particular bean instance, only one can access it at a time. This is shown in figure 2.2. This is not unlike making the methods of a regular Java class `synchronized`, although the EJB specification explicitly prohibits this. Also the EJB specification prohibits the use of threads inside beans, this is to ensure that the container has full control over the execution of processes in or to a bean.

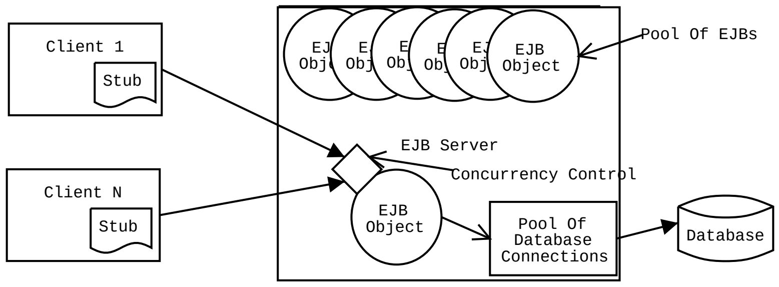


Figure 2.2: Concurrency Control.

Reentrance

Reentrance, is the concept of a thread of control attempting to reenter a bean instance in which it has already been in. The EJB specification [MH99] recommends that beans are nonreentrant by default. The reasoning behind this is that beans cannot loop back in to themselves. By forbidding this, we ensure that it is much more difficult to cause infinite loops in our code.

EJB 2.0 [MH01] specifies that most local references between beans will use the local interface, but it may still be necessary or desirable to use the remote interface, the reasoning behind this is that remote interface enforce location transparency, one of the main goals of distribute systems. For systems such as the distributed SMSC this dissertation produced, we can use only remote interfaces and then locate beans on other servers and not have to change much code at all, in fact all we have to change is where our client bean gets its initial context from, and no other code changes are necessary. This makes for very scalable systems, as we have almost total location transparency.

If we consider two beans, and lets give them names, one is Alice, the other Bob. Alice invokes a method on Bob, that method then attempts to make a call back to Alice before returning its results (if any), then this bean is reentrant. This is shown in figure 2.3. We describe in the XML deployment descriptor whether a bean is reentrant or not, this is covered in section 4.7.1.

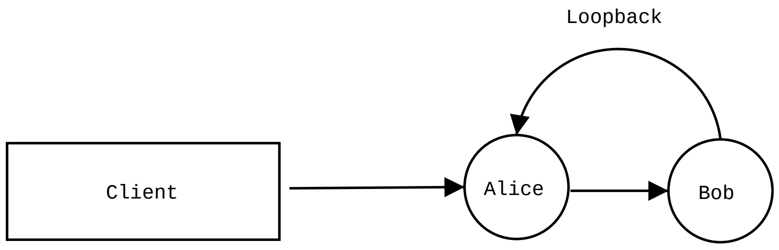


Figure 2.3: Alice And Bob's interaction.

2.6.11 Naming

All DOT services have to use a well defined and published naming service. Java Remote Method Invocation - Java Remote Method Protocol (RMI-JRMP) [Micl] uses JNDI as mentioned earlier. The primary function of a naming service is to allow clients to locate a service or resource. To this end naming services such as JNDI provide two primary operations that must be executed in order, they are:

- A Binding Service.
- A Lookup Service.

All EJBs deployed on an application server are available through what is called a JNDI context. Using this text based reference identifier a client can obtain a reference to a particular bean by first binding to the server and then doing a lookup on that server for a particular String which represents the bean. The EJB 1.1 Specification [MH99] recommends that all beans be preceded with the String `'java:comp/env/'` and then the local JNDI name. The reasoning for this is that it will enforce some sort of standard way to access beans in a JNDI tree. It also recommends that all beans be prepended by `'ejb/'`, again this is to bring some standardisation to the hierarchical structure of the JNDI tree.

The fact that the JNDI is a directory enabled naming service allows us to do this, we can classify and create a hierarchical structure to represent the contents of a system, i.e. the beans, datasources or environment variables. In addition, meta-information about services or objects can be added to their references in the JNDI tree. The metadata provides attributes that describe objects or resources and can if a client wishes, be used to search the directory structure for particular objects or services with qualities that match the meta-data. This meta-data can represent location, size or any quality a developer wishes.

JNDI also has the quality of being both dynamic and virtual, we can create links in the directory tree to external locations and they will become transparent entries in the tree. To improve the functionality of Application Servers, [MH01] specifies that they must all at the very least support the CORBA naming service in addition to any other naming service they wish to support. This now means that all beans are accessible as CORBA services in the same way regular CORBA services are available. Enterprise JavaBeans 1.1 requires the use of the `PortableRemoteObject.narrow()` method to cast remote references obtained from JNDI into the required Home-Remote interface type. Again this is to make EJBs CORBA compliant services, although its use in industry is at the present time questionable.

2.6.12 Locating Beans with JNDI

This section will attempt to describe how beans are located and accessed using the JNDI API. It has some small code examples to re-enforce the points made. Initially a client, be it application client[2], JSP, servlet or another EJB must obtain an initial context, this is done using the following code:

```
Context ctx = getInitialContext();
Object ref = ctx.lookup("comp:ejb/env/ejb/JNDIName");
EJBHome home =
  (EJBHome)PortableRemoteObject.narrow(ref,EJBHome.class);
```

- First an initial context is obtained, this is our binding service, it allows us to browse or query the JNDI tree of the system we have gotten a binding to.

[2]An Application Client is simply a type of client which uses EJBs

- Next we do a lookup for a particular bean, in this case it is a bean with a JNDI name `ejb/JNDIName`.

- We then use the `PortableRemoteObject.narrow()` method to cast the reference the JNDI lookup has given us to a class type we are expecting. This will throw a `ClassCastException` if the reference is of the incorrect type.

- Once a client has a reference to the remote home interface of an EJB it can create, locate or destroy as required.

2.6.13 Security

Application servers support as many as three different kinds of security: authentication, access control, and secure communication. Only access control is specifically addressed by EJBs so we'll focus our attentions on that.

Authentication

Authentication confirms the identity of a user and allows certain users to access certain particular sections of the system, an example of authentication would be a login screen used by a web portal.

Access control

Access control applies administrator defined explicit security policies that regulate what specific users can do in a system. Good access control systems permit correctly entitled users to access. Policies represent particular business goals and particular objectives which must then be understood by the server before it can determine the right of access to a resource. An example of this is the ownership of files, some users may have read privileges while others may have write, usually the owners of the file. The EJB deployment descriptor allows us to identify users and groups and permit access to methods to particular users or groups. This is particularly powerful as it allows the administrative functionality to be bundled with the regular functionality of beans. We can specify that only users of group type administrator can access the administrative methods, thus securing the system from malicious access by non-permitted users.

Secure communication

A critical section of security to be considered is secure communication. At present there is no specification for the secure transfer of instructions across a network using EJBs. Companies have produced products which fill this gap but have yet to be really tested by crackers. One solution could be to use Netscape's SSL technology to ensure secure communication between servers or communication across the network, but this is potentially expensive if used for all communication between servers.

2.6.14 The Sun Blueprints, Patterns For Success?

[Mic00]The Sun Blueprints aid the development of systems which conform to 'best practice' guidelines. These Blueprints have grown into a whole new collection of Design Patterns specially written for the Enterprise Architecture. For the most part they attempt to translate business logic into network services. Others focus on particular areas of the architecture, where particularly significant gains can be made. Some of the obvious benefits are:

- Capturing business logic more generically.

- Easing the transgression from application to networked service.
- Use of experience from other developers, a community of sorts.

2.7 Summary

This chapter contained an introduction into the technologies used by in the course of the dissertation. It attempted to outline some of the technologies available today and define the state of the art in component based design and computing. The following chapter describes the Requirements, Specification and Design of the prototype SMSC.

Chapter 3

Requirements, Specification And Design

3.1 Requirements

As mentioned earlier, the project sponsor, Logica Mobile Networks Ltd, is seeking a prototype system to act as a distributed Short Messaging Service Center (SMSC) for a network of Hewlett Packard Kayak workstations running the GNU / Linux operating system. Figure 3.1 shows the main components of the prototype system.

3.1.1 Functional Requirements

The system must allow clients called External Short Messaging Entities (ESME) to submit and retrieve Short Messages to / from the system. In addition there should be a centralised point of access to update configuration files for the system. It is important that data is held accessibly across the system, so the system should attempt to survive transient faults within its network of workstations; equally, it should attempt to minimise the network

traffic involved in carrying out its duties.
These short messages usually contain the following information before they reach the system:[Ins01]

- Destination Address
- Originating Address
- SMSC Address
- Status Report Request
- Service Center TimeStamp
- Validity Period
- Data Coding Scheme (not included in prototype)
- Protocol Identifier (not included in prototype)
- Reply Path
- Message reference (not included in prototype)
- Message Length
- Reject Duplicates (not included in prototype)
- User Data Header Information (not included in prototype)
- SMS Commands (not included in prototype)
- Message Type Indicator (Submit, deliver, Command, Status report)
- Message Content

Other data will be appended to the short message as it passes through the system.

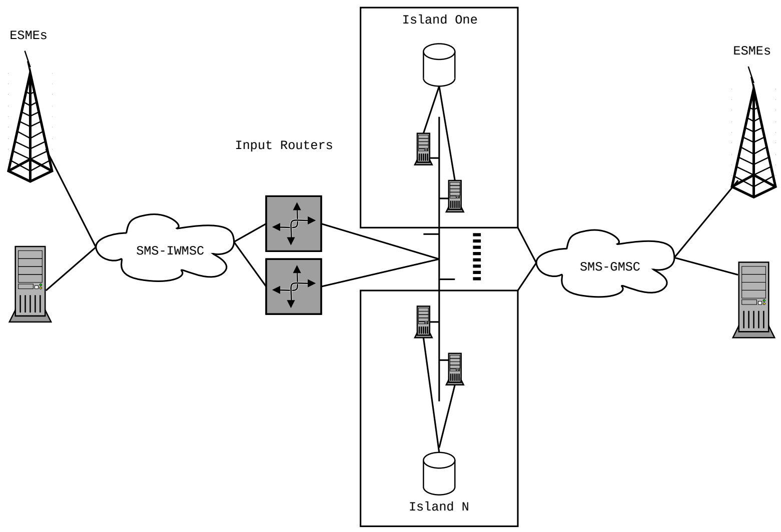


Figure 3.1: Architectural Overview Of The Prototype System.

The system will ensure the correct transfer of short messages to and from the ESME. These messages must be processed in the correct temporal order with respect to source / destination pairs and priority listing. The SMSC will control concurrent access to each network node ensuring message order and message content consistency. Simple hashing of SMS submission time is the methodology favoured for selection of message delivery order in addition to the destination address.

The SMSC must be tolerant of failure of network workstation nodes. Since the prototype will only support ‘store and forward’ type service, catastrophic system failure will be tolerated, as this is a non-volatile memory based system. As this is a prototype, there are no security requirements, no constraints on the network, no restriction on topology apart from domain size, which has

been defined by the number of machines supplied by the sponsor, Logica Mobile Networks, the system size is four Messaging nodes. The system must export a well-defined interface to MSC clients.
The minimum specification of the prototype as defined by meetings at Logica's offices at Customhouse Plaza Dublin, were:

- Simplest possible J2EE Cluster
- Handle Single Node Loss.
- Simple I/O protocol, client to drive.
- Some concept of routing.

An evaluation to examine performance and bottlenecks was also mentioned. Emphasis was also placed on a modular design.

3.1.2 Non-functional requirements

The system will run on a small but scalable set of workstations. The client sides of the system (i.e. the ESMEs and MSCs) should be considered external rather than internal to the system, thus simple programs will generate sample and test traffic. The SMSC will be completely automated after start up and BASH scripts will be written to start the service on reboot. A completely GNU / Linux PC-based network of workstations is assumed as our implementation platform, although this is not a limitation of the system. The software platform, apart from the operating system is available for any system with a Java Runtime Environment. This is of course referring to a Java 2 Enterprise Edition (J2EE) compliant Application Server (initially Orion Application Server [Sof]). No constraints have been placed on the choice of implementation paradigm used for the system but it is thought that this will closely mimic the current production (Telepath SMSC) and pre-production (Picasso 1000 SMSC) systems produced by the project sponsor Logica Mobile Networks.

3.1.3 Contractual constraints

The sponsor has requested a prototype of the required system. This was interpreted as a functional system demonstrating basic operation of a SMSC and its administration as laid out above. Security considerations are explicitly excluded from the project. A sample test harness for the demonstration has been provided by the author. It is assumed that the machines on which the project is being developed will be the test suite.

3.2 Specification

The major uses of the system, with their dependencies, are shown in the use case and sequence type diagrams below. They include:

- ESME submits a message.
- ESME receives a message.
- System start-up.
- Node loss.
- Node addition.
- Message order control.

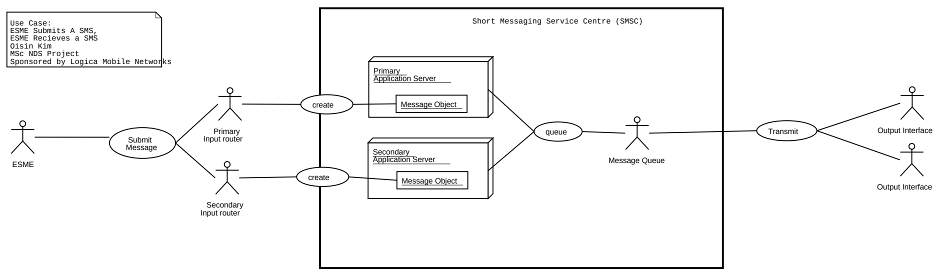


Figure 3.2: ESME Submit And Receives A SMS Message

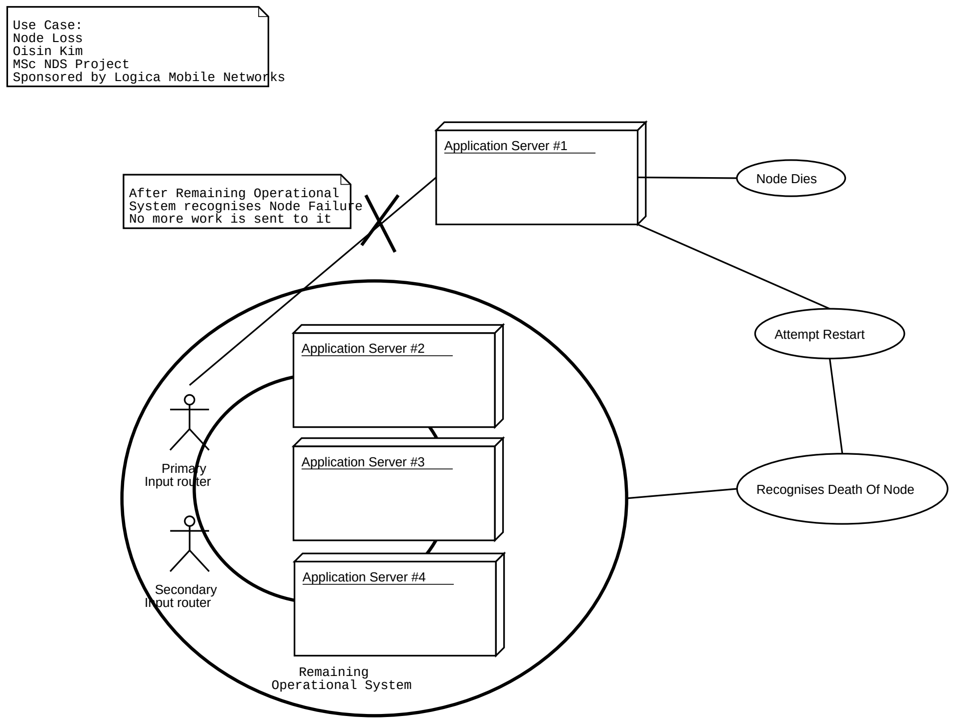


Figure 3.3: Single Node Loss

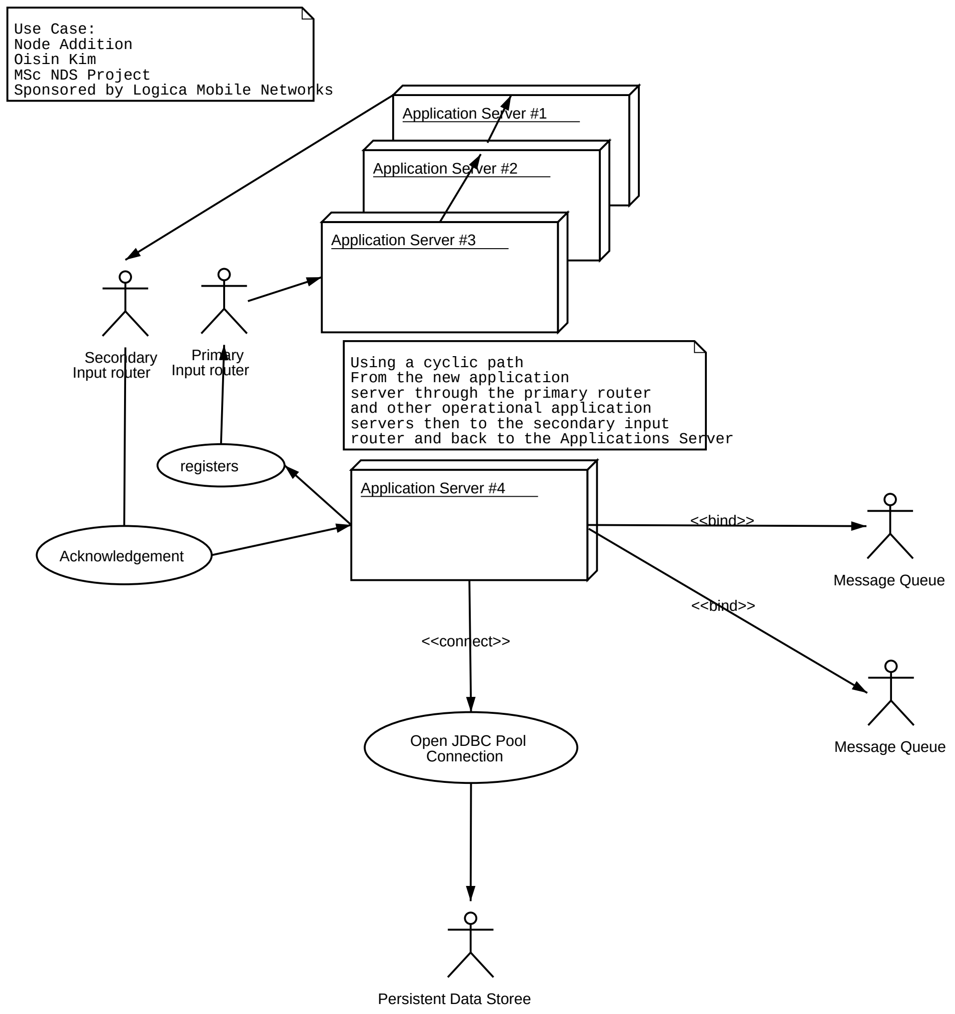


Figure 3.4: Single Node Addition

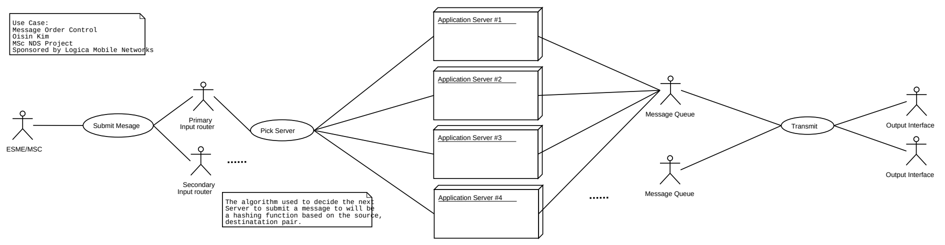


Figure 3.5: Message Order Control

3.2.1 The Project

The project centers on the development of a prototype Short Message Service Center (SMSC). The prototype will be implemented using the Java 2 Enterprise Edition (J2EE) architecture and run on a network of 4 HP GNU / Linux boxes. This section describes the basic functionality of a SMSC and what the prototype system will offer in terms of: Administrative Architecture. Architecture. Messaging Features. Clustering. Recovery. Database / Persistence.

3.2.2 Administrative Architecture

Our proposal is to develop the Administration system using the Model View Controller (MVC) [ea95] Architecture whereby multiple clients and SMSC nodes interact. Some benefits of the MVC architecture to this system are:

- Ease of maintenance, this would reduce the costs of modifying the system in the future if it was required also the pluggable nature of system allows components to be modified individuality.

- The Architecture fits well with standard Object Oriented best practices, for example UML [wRP99] and allows the development of components (model, view and controller) individually, which potentially could optimise up the development time, as there are less dependencies on different sections of the architecture that could be developed in parallel.

3.2.3 Architecture

The prototype will mimic the Logica production SMSC in many ways, some of the common architectural features are:

- Use of shared nothing architecture.
- Use of a cluster-proper architecture by replicating services to provide improved throughput.
- Single point of management for the locally distributed network of nodes (a node is intended to mean a GNU / Linux box offering short message processing services).
- Heterogeneous platform support High system / service availability, achieved by an architecture resilient to single node failure initially but with resilience to multiple failure (N-1) easily obtained as mentioned previously by adding additional islands.
- Basic distributed short message processing.
- Basic billing management.
- Basic load control / balancing.

The prototype will attempt to address the standard industrial requirements for Reliability, Availability and Scalability (RAS). Another important issue is the emphasis on a clean modular design, ESMEs will operate independently of each other. Each client has access to the full range of message operations, i.e. Submission and reception.

Each node will contain a replica of a generic node capable of processing messages. The properties and configuration of the system will be distributed throughout the system on each node individually to remove any single point of failure. When an ESME binds to the service it can then submit or receive messages. The input routers will use round robin load balancing to ensure close to optimum load balancing between nodes as the sample test data is random. The message nodes in a particular island will replicate the message objects in their storage.

No constraints were placed on the implementation methodology of our system. We have therefore initially specified an initial static network topology where SMSC nodes are limited in number to four. For the prototype we will not provide a customized database or database management system, instead we will use the Open-Source Database PostgreSQL [Pos] which incidentally has been chosen for RedHat's [Red] newly announced Redhat DB. The ESME will exhibit a clean modular design that should be easily extended and plugged into.

Client-Server communication will be initially achieved by Remote Procedure Call (RPC).

Fault tolerance is implemented using the standby approach. This means that every message data object is replicated on another messaging node. This is similar in ways to the redundant backup approach used by some HA systems, but in the prototype all nodes are functional and operate equally if available. A number of possible approaches to fault tolerance are discussed later.

3.2.4 Messaging Features

We are primarily concerned with two aspects of the manipulation of SMS messages, order and storage.

- Message Ordering
  Initially the prototype will guarantee order of messages with the same [priority, destination] pair. This will stop concurrent access to messages from different nodes.

- Message Storage

  All messages will be stored as entity beans on the application server which ensures that beans are stored with a disk based back-up (database).

3.2.5 Clustering

It is vital to note that cluster computing consists of two distinct branches:

- Compute clustering (such as Beowulf) uses multiple machines to provide greater computing power for computationally intensive tasks. This type of clustering is used as the system replicates services which increases the computing power of the cluster.

- High Availability (or HA) clustering uses various technologies to gain an extra level of reliability for a service. HA clustering is the intended solution for the prototype.

Figures 3.6 and 3.7 show sample configurations for the HA branch [Red00] [Pro01]:

It should be noted that the level where the clustering resides might change. There are three distinct solutions of which two are under investigation:

1. Clustering at the Operating System level.

2. Clustering at the application server level.

3. Clustering at the Enterprise Java Bean (EJB) level.

The solutions this project is interested in was 2 and 3 above, although clustering at the application server level is unavailable for all but the most expensive application servers. To this end the author hardcoded the clustering into the actual bean code.

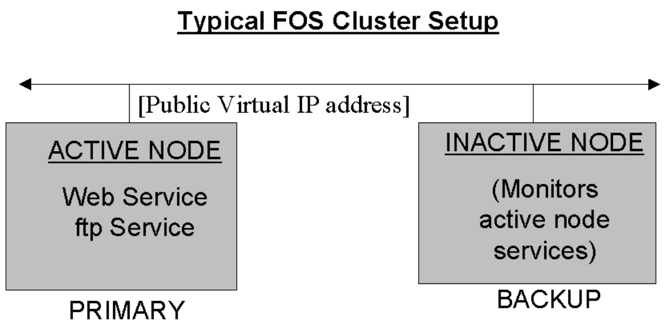


Figure 3.6: Typical Fail Over Service Architecture

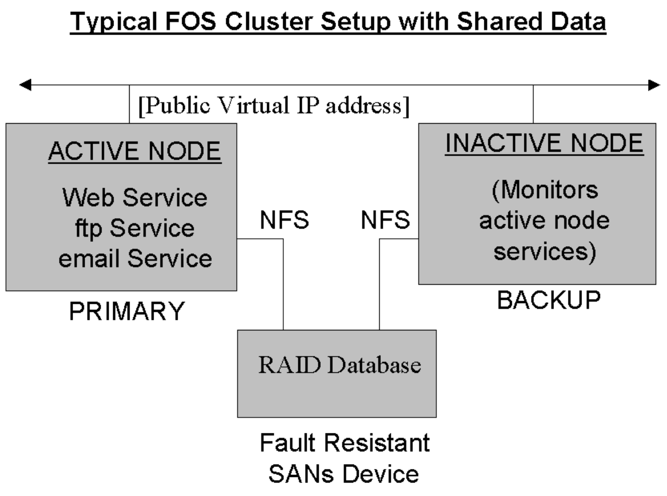


Figure 3.7: Typical Shared Database Fail Over Service Architecture

3.2.6 Recovery / Failover

One of the primary goals of High Degree RAS computing is to attempt to create software that can operate in the face of failure of cluster nodes. The detection of nodes that have died is important to ensure that those nodes may be restarted or identified as needing attention from a system administrator or technician. The current production architecture employed by the Logica SMSC is to halt all processing while nodes rejoin or leave the cluster while messages are transferred between operational nodes, this has a detrimental effect on the optimal or maximum possible output achievable by all operational nodes as they must wait for changes to filter through before processing any further messages.

3.3 Design

This section attempts to describe how the modules of the architecture are expected to interact and the reasoning behind their construction.

3.3.1 Beans

There are four beans, each with a specific purpose, we have two entity beans, the Home Location Registrar (`HLRBean`) and the Message Node (`MessageNodeBean`). There are two session beans, both stateless, they are the Queue bean (`QueueBean`) and the Transmit bean (`TransmitBean`). We will now describe the functionality of each bean.

Home Location Registrar (HLR)

Essentially the HLR is tasked with providing a service which will do the following:

- Allow the SMSC to determine if a subscriber's mobile telephone is on or off.
- Notify the SMSC when a subscriber becomes available.

Message Node

The MessageNodeBean objectifies a SMS message and the operations we need on it. Some of the things we need to do with a SMS message are:

- Submit a SMS to the system.
- Transmit a SMS when the destination address is available.
- Query a SMS to determine its up-to-date current state.
- Retry to send a SMS after backoff period has elapsed.
- Return the contents of a SMS to a client.

Queue

SMSs cannot be simply transmitted when a subscriber is available. First all pending messages for a particular destination must be added to a queue and sorted so that they are delivered to a subscriber in correct temporal order. The queue bean has the task of transmitting the SMS messages for a particular destination address in the correct order, considering time and priority.

Transmit

The TransmitBean is used to simulate the actual transmission of a SMS to a subscriber. It simply writes to file a message which indicates that a message had been delivered to a particular subscriber using a particular network.

3.3.2 Input Router

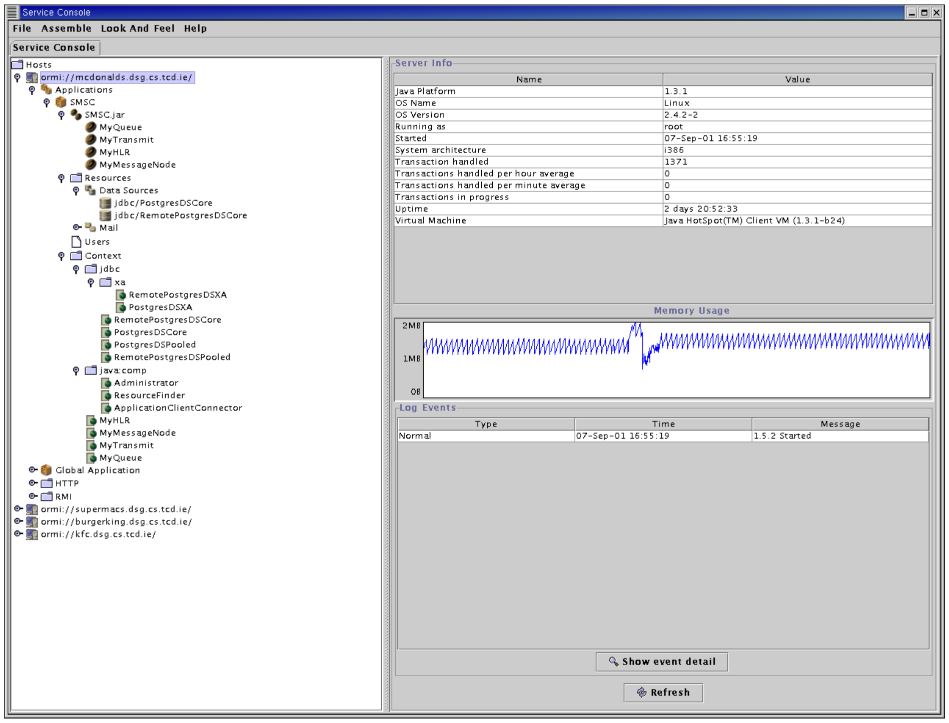
The Input Router is the external interface of the SMSC. It receives SMS messages from clients and distributes them to a node in the cluster for processing. It should act independent of the SMSC cluster. For the prototype the Input Router will simply send messages to the nodes in a round robin fashion. It is also possible to modify the topology of the cluster, at run time, to add or remove nodes from the original static topology.

3.3.3 Network

The network has been designed so that there is a public network, used by the Input Router to submit messages and communicate with the cluster, and also a private network, used by the nodes to communicate with each other. This is achieved by simply having two network cards and stating in the `/etc/hosts` file that all traffic for a particular set of IP addresses should use a particular network card. This is achieved by have each network card on machine (there are two) represent a different network.

3.3.4 Administration JSP

As mentioned in the Introduction, there are two methods for administering the cluster, one is the Orion Console Administrator, which comes as a beta tool for the Orion Server. Figure 3.8 shows the Orion Console in operation. Another option for administering the cluster remotely is to use a JSP based system. Here after a login, a user can call methods depending on their security clearance.



```
Hosts
  ormi://mcdonalds.dsg.cs.tcd.ie/
    Applications
      SMSC
        SMSC.jar
          MyQueue
          MyTransmit
          MyHLR
          MyMessageNode
        Resources
          Data Sources
            jdbc/PostgresDSCore
            jdbc/RemotePostgresDSCore
          Mail
          Users
        Context
          jdbc
            xa
              RemotePostgresDSXA
              PostgresDSXA
            RemotePostgresDSCore
            PostgresDSCore
            PostgresDSPooled
            RemotePostgresDSPooled
          java:comp
            Administrator
            ResourceFinder
            ApplicationClientConnector
          MyHLR
          MyMessageNode
          MyTransmit
          MyQueue
    Global Application
    HTTP
    RMI
  ormi://supermacs.dsg.cs.tcd.ie/
  ormi://burgerking.dsg.cs.tcd.ie/
  ormi://kfc.dsg.cs.tcd.ie/
```

Name	Value
Java Platform	1.3.1
OS Name	Linux
OS Version	2.4.2-2
Running as	root
Started	07-Sep-01 16:55:19
System architecture	i386
Transaction handled	1371
Transactions handled per hour average	0
Transactions handled per minute average	0
Transactions in progress	0
Uptime	2 days 20:52:33
Virtual Machine	Java HotSpot(TM) Client VM (1.3.1-b24)

Type	Time	Message
Normal	07-Sep-01 16:55:19	1.5.2 Started

Figure 3.8: Screenshot of the Orion Console Administrator.

3.4 Summary

In this chapter we have described the Requirements, Specification and Design of our prototype system. We have showed the different components of the systems and explained what is expected of each component. The next chapter will describe the implementation of the system and argue why some implementation options were followed.

Chapter 4

Implementation

This chapter attempts to describe the implementation issues involved in the prototype. We start by describing in detail the platform used to deploy the system on. We continue by describing the beans and components of the system and how they were packaged. We finish by giving a description of how the beans are deployed on the platform.

4.1 Implementation Platform

The database chosen for the system was PostgreSQL 7.0.3, the application server was Orion Application Server 1.5.2., the Java Development Kit (JDK) and Java Runtime Environment (JRE) was Sun's JDK 1.3.1.

4.2 Beans

This section describes the construction of the beans used in the prototype.

4.2.1 HLRBean

As described earlier in section 3.3.1, the main functionality of the HLR is to let the SMSC know if a subscriber is online / available and notify the SMSC when a subscriber comes back online / available after a period of inactivity. Although the notification was not included in the prototype the method was defined so that future versions of the EJB could use exactly the same interface as the one defined by the prototype, this is an example for future proofing a EJB. We include the functionality that we expect the EJB to have and we can then as needed, implement the methods.
The choice for the HLRBean was simple, we had a group of data that we wished to access in an Object Oriented (OO) way. Entity beans as already described in section 2.6.1 satisfy this requirement. Concurrent access to the bean is handled by the container as described in section 2.6.10. The HLR-Bean remote interface exposes the business methods that clients to the Home Location Registrar will use to interact with the HLR EJB. This is the first class you generally code when writing an entity EJB.

```
import javax.ejb.EJBObject;
import java.rmi.RemoteException;

public interface HLR extends javax.ejb.EJBObject {
    public int getSubscriberNumber()
      throws RemoteException;
    public void setSubscriberNumber(int number)
      throws RemoteException;
    public boolean getAvailable()
```

```
      throws RemoteException;
    public void setAvailable(boolean bool)
      throws RemoteException;
    public void notifySMSC()
      throws RemoteException;
}
```

The methods defined are all obvious perhaps except the `notifySMSC()` method. This method as mentioned above wasn't implemented, by would allow for the Home Location Registrar to notify a SMSC that a particular subscriber was online after a period of inactivity. The other methods allow a client to either set or get attributes from the database. For the purposes of the prototype the database was filled with 100,000 user with subscriber / phone numbers in the range 1,000,000 to 1,100,000. All even numbers were set to be online, all odd, offline. A client (usually a Messaging Node in the cluster) could get a reference to the HLR and and ascertain if a particular user was available by examining the return value of the method `getAvailable()`. Its perhaps worth pointing out that no SQL code was required to be written by the developer for the database access as it is one of the responsibilities of the container for CMP type entity EJBs.

4.2.2 MessageNodeBean

The MessageNodeBean describes the main functionality of the SMSC. The MessageNode remote interface defines the "business methods" that clients to the MessageNode (i.e. The input router) will use to interact with the MessageNode EJB. The main functionality of the MessageNode is to:

- Accept for submission a SMS from the Input Router.
- Transmit a SMS when the destination subscriber is available.
- Attempt Queuing of the SMS when the destination ESME is available.
- Query a message to check for it's delivery status.
- Retry to Queue a Message when directed to by it client.

We show here the code for the remote interface, MessageNode.

```
import javax.ejb.EJBObject;
import java.rmi.RemoteException;
import javax.ejb.RemoveException;

public interface MessageNode
  extends javax.ejb.EJBObject {

    public boolean SubmitMessage()
      throws RemoteException;
    public boolean TransmitMessage()
      throws RemoteException;
    public String QueryMessage()
      throws RemoteException;
    public boolean RetryMessage()
      throws RemoteException;
    public Message ReturnMessage()
      throws RemoteException;
}
```

SubmitMessage

This is the method called after the Message has been created on the Application Server. It first checks to see if the `DestinationAddress` user is online by obtaining a reference to the `HLRBean`. If the subscriber is available, it will queue the message, and any others for that subscriber for delivery by calling the `TransmitMessage` method below.

TransmitMessage

Here we know the user is online so we simply add the message and any others for the destination to the queue. We do this by creating a QueueBean with a list of Messages due for a particular subscriber.

QueryMessage

The `QueryMessage` method simply return the contents of the Status, more functionality could be added as desired.

RetryMessage

The `RetryMessage` method obtains a reference to the `HLRBean` to see if the `DestinationAddress` user is online. If they are it it will queue the message, and any others for that subscriber for delivery by calling the `TransmitMessage` method.

ReturnMessage

The `ReturnMessage` method will return a `Message` object based on the contents of the `MessageNodeBean`'s data members.

4.2.3 QueueBean

The `QueueBean` is responsible for the message order control. It determines the order of transmission of messages from the SMSC. We now examine the code for the Queue remote interface.

```
import java.rmi.RemoteException;
import java.util.ArrayList;

public interface Queue
  extends javax.ejb.EJBObject {

    public boolean Deliver(ArrayList Messages)
      throws RemoteException;
}
```

The single method defined `Deliver(ArrayList Messages)` will attempt to deliver all the messages contained in `Messages`.This is a perfect example of how a session EJB acts as an API.

4.2.4 TransmitBean

The `TransmitBean` is a simulation for the signaling software/hardware that would interface with a SMSC. It allows for the transmission of messages via one of two networks, which are Eircom or Esat.

```
import java.rmi.RemoteException;
import java.util.ArrayList;

public interface Transmit
```

```
  extends javax.ejb.EJBObject {

    public boolean ToEsat (Message MO) throws RemoteException;
    public boolean ToEircom (Message MO) throws RemoteException;
}
```

The methods `ToEsat (Message MO)` and `ToEircom (Message MO)` define the two signaling options we have, for example, `ToEsat (Message MO)` could have been a local network for which we could have used our own signaling hardware / software and `ToEircom (Message MO)` could have been a remote network to which we would have needed to contact the destination subscriber's SMSC.

4.3 InputRouter

The Input Router deals with the routing of messages into the SMSC cluster. As previously described it uses a round robin scheduling algorithm to determine the next node to receive a SMS message. It was implemented as a RMI service. Here is the remote interface:

```
import java.rmi.Remote;
import java.rmi.RemoteException;
import MessageDataObject;

public interface InputRouter
  extends java.rmi.Remote{

    public boolean getReferencesToServers()
      throws java.rmi.RemoteException;
    public boolean configureServers(String ServerNames)
      throws java.rmi.RemoteException;
    public void DistributeMessage(MessageDataObject MDO)
      throws java.rmi.RemoteException;
    public void removeServer(String ServerName)
      throws java.rmi.RemoteException;
    public void addServer(String Servername)
      throws java.rmi.RemoteException;
}
```

`getReferencesToServers()` directs the RMI server to obtain references to nodes in the cluster.
`configureServers(String ServerNames)` configures the names of the servers to which messages must be routed to.
`DistributeMessage(MessageDataObject MDO)` actually will submit the message on one of the messaging nodes.
`removeServer(String ServerName)` will remove a node from the list of active servers so that no more SMS messages will be routed to it.
`addServer(String Servername)` will return a server to the list of servers to be chosen from that messages will be submitted on.

4.4 Client

The client simply submits SMS messages to the SMSC system as fast as it can create them and the system can submit them. It was considered external to the system so is not investigated here.

4.5 MessageDataObject

The `MessageDataObject` objectifies the SMS message just after the External Short Message Entity (ESME) has submitted it. The data it holds is:

```
int DestinationAddress;
int OriginatingAddress;
String SMSCAddress;
String StatusReportingRequest;
long ServiceCentreTimeStamp;
String ReplyPath;
int Length;
String MessageTypeIndicator;
String Content;
```

4.6 Message

The `Message` Object represents the SMS message as it is just before, during and after processing by the SMSC. It extends `MessageDataObject` and therefore inherits all its protected and public data members (as listed above) and methods. The data it holds is:

```
int Priority;
boolean BillingDetails;
String RoutingInfo;
boolean IsPrimary;
long HashValue;
boolean WasTransmitted;
```

```
int Cost;

String NodeName;

int ClusterIsland;

int MessageID;
```

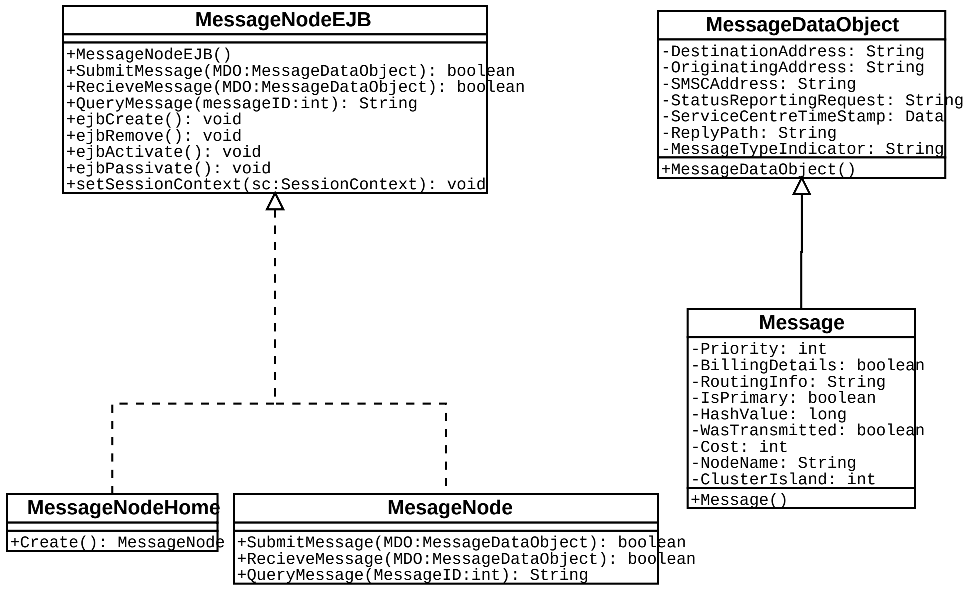

MessageNodeEJB
+MessageNodeEJB()
+SubmitMessage(MDO:MessageDataObject): boolean
+RecieveMessage(MDO:MessageDataObject): boolean
+QueryMessage(messageID:int): String
+ejbCreate(): void
+ejbRemove(): void
+ejbActivate(): void
+ejbPassivate(): void
+setSessionContext(sc:SessionContext): void
MessageDataObject
-DestinationAddress: String
-OriginatingAddress: String
-SMSCAddress: String
-StatusReportingRequest: String
-ServiceCentreTimeStamp: Data
-ReplyPath: String
-MessageTypeIndicator: String
+MessageDataObject()
Message
-Priority: int
-BillingDetails: boolean
-RoutingInfo: String
-IsPrimary: boolean
-HashValue: long
-WasTransmitted: boolean
-Cost: int
-NodeName: String
-ClusterIsland: int
+Message()
MessageNodeHome
+Create(): MessageNode
MesageNode
+SubmitMessage(MDO:MessageDataObject): boolean
+RecieveMessage(MDO:MessageDataObject): boolean
+QueryMessage(MessageID:int): String

Figure 4.1: Basic Messaging Class Diagrams

4.7 Packaging The Beans

The beans are packaged in a file called SMSC.jar. It is created using the Java Archive (jar) [Mici] command. The command to create the jar file is :

```
jar cvf SMSC.jar *.class /META-INF/ejb-jar.xml
```

The parameters passed to jar are:

`c` - To create a jar file.

`v` - For verbose to show what it's doing.

`f` - To specify a file.

`SMSC.jar` - The desired jar file name.

`*.class` - Tell jar to add all files in the current directory with .class as the file extension.

`/META-INF/ejb-jar.xml` - Tells jar to add the directory META-INF and a file in there called ejb-jar.xml.

The ejb-jar file can be used to deploy the application on any J2EE compliant application server. The only problem is that the same datasource must be available on the new application server. Datasources are shown in the deployment descriptor next.

The Jar File's Contents

To display the contents of a jar file we use the jar command again, passing in three parameters, they are:

`t` - List the table of contents.

`f` - To Specify a file input.

`SMSC.jar` - To Specify a file as input.

The result of calling the command are shown below together with an explanation of what the command is doing:

```
[oisin@mcdonalds ejb]: jar tf SMSC.jar
META-INF/
META-INF/MANIFEST.MF
Client1.class
```

```
HLRBean.class
HLR.class
HLRHome.class
Message.class
MessageDataObject.class
MessageNodeBean.class
MessageNode.class
MessageNodeHome.class
QueueBean.class
Queue.class
QueueHome.class
TransmitBean.class
Transmit.class
TransmitHome.class
META-INF/ejb-jar.xml
```

4.7.1 Deployment Descriptor

The deployment descriptor is a well formed XML document which describes the application described in the ejb-jar file. The deployment descriptor defines how beans in an application are assembled and can describe local JNDI references that beans can use to get references to and make invocations on.

The EJB Specification [MH99] describes two kinds of information in the deployment descriptor,

1. EJB Structural Information.
   This refers to the structure of a EJB and also any external dependencies the EJB might have.

2. Application Assembly Information.

   This refers to how the EJBs contained in a ejb-jar file are composed to form larger 'application deployment units'.

Here we examine some of the sections of the ejb-jar.xml file which is the deployment descriptor for the SMSC.jar file / application. We will show the basic tags used for deploying a session bean, entity bean and the assembly descriptor.

Session Bean Deployment Descriptor

```
<session>
    <description>The Simulated Transmission Bean</description>
    <ejb-name>MyTransmit</ejb-name>
    <home>TransmitHome</home>
    <remote>Transmit</remote>
    <ejb-class>TransmitBean</ejb-class>
    <session-type>Stateless</session-type>
    <transaction-type>Container</transaction-type>
</session>
```

In this fragment of the deployment descriptor we can see the following items clearly specified:

- A human readable `description` of the bean.
- The `ejb-name` of the bean which usually becomes the JNDI name.
- The `home,remote,bean` classes.
- Whether the bean is `Stateless` or `stateful`.

- The transaction-type, Container or Bean.

Entity Bean Deployment Descriptor

```
<entity>
    <description>The Home Location registrar Bean<description>
    <ejb-name>MyHLR</ejb-name>
    <home>HLRHome</home>
    <remote>HLR</remote>
    <ejb-class>HLRBean</ejb-class>
    <primkey-class>java.lang.Integer</primkey-class>
    <reentrant>False</reentrant>
    <persistence-type>Container</persistence-type>
    <cmp-field><field-name>SubscriberNumber</field-name></cmp-field>
    <cmp-field><field-name>Available</field-name></cmp-field>
    <primkey-field>SubscriberNumber</primkey-field>
  </entity>
```

In this fragment of the deployment descriptor we can see the following items clearly specified:

- A human readable description of the bean.
- The ejb-name of the bean which usually becomes the JNDI name.
- The home,remote,bean classes.
- What Java type the primary key class is.
- If the bean is reentrant or not.

- The `persistence`, `Container` or `Bean`.
- The `cmp-fields`, all those data members we wish to be made persistent.
- The `primkey-field`, to be used as the primary key of the entity.

Assembly Descriptor

```
<assembly-descriptor>
  <container-transaction>
    <method>
      <ejb-name>MyHLR</ejb-name>
      <method-name>*</method-name>
    </method>
    <trans-attribute>NotSupported</trans-attribute>
  </container-transaction>
  .....
</assembly-descriptor>
```

In this fragment of the deployment descriptor we can see the following items clearly specified:

- A list of all the EJBs described in the deployment descriptor.
- The `ejb-name` of the bean and a list of the methods for that bean. Represented by `*` here.
- The `trans-attribute` for that method.

4.8 Deploying The Beans

To deploy the beans all that is required is the following steps:

1. Create a user called postgres by typing:

   ```
   createuser postgres
   ```

2. Create the Database sms by typing :

   ```
   su postgres
   createdb sms
   psql sms
   ```

   `\d` will show you tables (after orionserver creates them)

3. Edit the `/var/lib/pgsql/pg_hba.conf` to allow access from any server in the island on each node.

4. Add the PostgreSQL JDBC driver to the Application Server's classpath.

5. Copy the jar file to the location e.g. `/usr/orion/SMSC/ejb/SMSC.jar`

6. Modify the `/usr/orion/config/data-sources.xml` file to recognise the Postgresql database.

7. Modify the `/usr/orion/config/principals.xml` to add users / groups.

8. Modify `/usr/orion/config/server.xml` to point to the application e.g. `application name="SMSC" path="../SMSC/ejb"/`

9. Deploy the bean by starting up orion server. (ensure there are no errors) use

   ```
   java -server -jar orion.jar -verbosity 20
   ```

10. Modify `/usr/orion/config/schemas/postgresql.xml` to have `long = int8` not `long = integer` [1].

This list assumes that the user has already tailored each MessageNodeBean by modifying the RemoteServer static data member in the class. This requires recompilation of the classes.

[1] For some strange reason, Iron Flare software (the makers of Orion Server) have determined that a long value should be written to the PostgreSQL database as an int value. This of course, would cause trouble for those actually using long values. Thankfully a simple solution was found. By modifying the database schema as described above the problem is eradicated.

Chapter 5

Evaluation

5.1 Functionality

It would be difficult to estimate exactly how much of the production system was replicated in the prototype. The prototype SMSC does offer however most of the processing features relating to queuing and order of delivery. The transmission of messages was simulated using a simple method call `Transmit.ToEsat()` or `Transmit.ToEircom()`, in a production system this method call would be responsible for interfacing with the signaling system or routing the SMS message across a network to the Destination's SMSC. The system also offers full ACID compliant transactions for the submission of SMS messages. In this way, if a SMS messages is accepted we know the following:

- It has been accepted by one of the Application Servers in it's `createEJB (Message amessage)` method.

- It has been replicated, for fault tolerance, on another machine.

This ensures that a message is stored persistently and will be delivered if one of the Application Servers is running when the person is online.

5.2 Fault Tolerance

The system used multiple (two) disk based storage of messages waiting to be delivered. This ensured the system could survive failure of nodes. As mentioned previously, the production system stops processing messages for a short period of time to re sync nodes in the cluster that were out of action. This could be implemented using checkpoints, where we note that all island nodes are in sync, up to a certain point in time. When a node wishes to rejoin the island, all that is required is that the out of sync node obtains all data that it missed, beginning from it's last checkpoint. As each node has a reference to a remote datasource on its paired node, this would be a trivial act. Simply by retrieving all messages from a certain time onwards and inserting those messages into the database of the out of sync node, we can ensure that all nodes are in sync again. This is possible since we can set a tag in one of the Application Servers files, in Orion's case `orion-ejb-jar.xml` to notify that the database is accessible by processes other that itself. The tag's contents is `exclusive-write-access="false"`.

The prototype system did not meet the standards of High Degree RAS systems. Failure of Messaging Nodes was not tolerated, this was due to exceptions being thrown by the application server which the author did not have time to track down and fix. The short development time as described below was the key factor. However clients which simulated failure of both Input Routers and Messaging node were created and demonstrated the direction a system with more development time would have gone in.

5.3 Development Time / Lines Of Code

The total time for the project is overshadowed by the fact that over two months was spent reading a mass documentation. This included proprietary documentation from Logica and other web based documents. The design consisted of over one months revision of UML diagrams and architectural prototypes. Since the author was a novice EJB programmer three weeks was spent learning the EJB architecture and speculating how the architecture could be used to its fullest. In all under nine weeks was spent coding the prototype, this was a testament to the readily available functionality that did not have to be rewritten. However it was felt the core functionality, for message delivery and ordering was implemented.
In terms of Lines Of Code written, the production systems as verified by Logica amount roughly to:
Telepath : 1,000,000 Lines Of Code
Picasso : 600,000 Lines Of Code
The prototype did not implement the full functionality of a SMSC center, but the total lines of code written, including test harnesses was under 10,000. The re-use of the container / EJB Server functionality meant we would use the transactional and other complex facilities that would normally be commissioned especially for the Logica production system. The reuse of generic code on each machine also attributed to the small code size.

5.4 Performance Evaluation

The system was benchmarked by running one or two clients which submitted messages to the server as fast as the server could handle them. For the purposes of the benchmarking, the clients and the Input Router component were situated on machines outside the cluster. The results were consistent in that the number of messages handled by the cluster remained within a 2% variation of the mean. The very small difference in the performance could be attributed to the following items:

1. Network traffic, as the system is handling more messages per unit time.
2. As the Number of Messages processed by the cluster per minute grows, so does the load on the client to produce messages.
3. Again as the number of messages grow, the load on the Input Router grows accordingly as it has more nodes to perform it's routing on. In the case of the prototype data below 50% more. This involves a search on 50% more data.[1]

5.5 The Data

The data shown in Table 5.5 illustrates the performance of the system under a variety of conditions. For the test we have varied:

1. Number Of Input Routers : 1 or 2.
2. Number Of Message Nodes : 2 or 4.

[1]This is calculated since the first test examined the system using 2 messaging nodes, the second test 4 nodes.

3. Time (Seconds) : 600 or 6000 seconds.

It should be noted that varying the number of Message Nodes between 2 and 4 is equivalent to varying the number of islands in the system between 1 and 2.

Number Of Input Routers	Number Of Message Nodes	Time (Seconds)	Number Of Messages Processed (Total)	Messages Processed Per Minute Per Island
1	**2**	**600**	**6459**	**646**
2	**2**	**600**	**6466**	**647**
1	**2**	**6000**	**64598**	**646**
2	**2**	**6000**	**64672**	**647**
1	**4**	**600**	**12884**	**644**
2	**4**	**600**	**12936**	**647**
1	**4**	**6000**	**129213**	**645**
2	**4**	**6000**	**129410**	**646**

5.6 Performance Requirements

To get an idea of the required performance of a SMSC, we done the following analysis on the prototype; we have worked out the number of cluster nodes our system would require to handle the number of chargeable messages sent in the whole of the UK for the month of July 2001.

The number, according to the MDA [Ass01], is 900,000,000.
To obtain the number of messages per day we get:
900,000,000 / 31 = 29032258.06 (number of messages per day)
29032258.06 / 24 = 1209677.419 (number of messages per hour)
1209677.419 / 60 = 20161.29032 (number of messages per minute)
20161.29032 /646 = 31.20942774 (number of islands required to handle load)
This equates to a load that would require 32 islands or 64 servers, a small cluster when compared to Google's 10000+ [Web01].

However, this analysis does not take into account the uneven spread of data, since obviously the vast majority of the SMS messages sent would be in the hours of 8am until 12pm, when most people are awake. If we re-calculate the totals, taking the worst case, that all messages are sent between the hours 8am to 12pm as listed above, we get:
900,000,000 / 31 = 29032258.06 (number of messages per day)
29032258.06 / 18 = 1612903.226 (number of messages per hour)
1612903.226 / 60 = 26881.72043 (number of messages per minute)
26881.72043 /646 = 41.61257033 (number of islands required to handle load)

This equates to a load that would require 41 islands or 82 servers, again a relatively small cluster. Another benefit of the architecture is the ease of expansion, by simply adding more islands the system can handle more SMS messages per minute, 646 to be exact. It should be noted however that this analysis considers the Input Router to be external to the cluster and therefore not included in the number of servers required.

To consider this in real terms, the income generated by these messages would be in the region of:
900,000,000 X .10(STG £)= STG £90,000,000.00
We can assume this figure since the data provided by the Mobile Data Association is for 'chargeable SMS messages' [Ass01] and the average price for a SMS message is ten pence.

5.7 Bottlenecks

We have identified the following list of bottlenecks in the system, these have been examined as they are the standard list of bottlenecks in distributed systems as defined by [Ren96], we have also added Legacy Bandwidth in our appraisal, they are:

- Use Of Transactions.
- Database / Disk I/O.
- CPU Load.
- Network Bandwidth.
- Legacy Integration.

5.7.1 Use Of Transactions

Transactions are expensive and bring in the possibility of deadlock into systems. Therefore we should act carefully when selecting which methods are transactional. Using the ejb-jar.xml file we can select methods to be one of

Container or Bean managed and also we can select from one of the following trans-attributes:

1. NotSupported
2. Supports
3. Required
4. RequiresNew
5. Mandatory
6. Never

This gives us very fine grained control over what methods are or are not transactional and what type of transaction they support if any.

Solution

- Minimise the use of transactions.
- Use enforcement of use of transactions at method level as ejb-jar.xml file permits.

5.7.2 Database / Disk I/O

Database / Disk I/O seemed to be the bottleneck in the system, as mentioned below the CPU load never got above 55% in testing and memory use never above 60%, this would suggest that the machine was waiting for disk I/O to terminate. Also adding more Clients to the system did not make a significant difference to CPU load or throughput. Again this suggests that one section of the system was working at it's limit. We believe that this limit was reached by the Disk I/O. In addition, the output of the GNU top application in Linux showed that the 'postmaster' [2] process was taking up most of the CPU load. Further investigation showed that a simple Java application that used JDBC to write to the database could max out disk I/O using just 35% of the CPU load. Also, not all the functionality of the database is used, it may be less expensive to use a custom type to data store which would not have the high overhead of a Relational DataBase Management System (RDBMS) like PostgreSQL.

Solution

- Use SCSI 3 Hard drives to improve Disk I/O rates.
- Investigate use of custom or Object Oriented Database to improve performance.
- Investigate use of memory based database e.g. Hypersonic [Inc01].

5.7.3 CPU Load

The CPU load as shown by the GNU top application never showed loads values above 55% when including user + system + nice usage at full load. This would suggest that CPU load was not an issue with the system.

Solution

- Use smallest object possible for data values, e.g. use a char array not String.

[2]The Postmaster process is the process for the PostgreSQL database running on the server

- Test performance over a number of Application Servers to determine which is most suited.
- Investigate use of Message Driven Beans or JMS instead of expensive entity beans.
- Replicate the beans in memory not on disk.

5.7.4 Network Bandwidth

Although this was not an issue for the prototype, we describe an architecture to solve the future problem that could arise if bandwidth limits were reached. The architecture is based on the idea that we can create private networks between selected nodes in the cluster. Figure 5.1 shows clearly the two networks in operation. We have a private network that all data between nodes in the cluster communicate with. This was easily implemented by modifying the `/etc/hosts` file and hardcoding the destination to be used by nodes in the cluster when they are communicating with one another. The public network is used for access to the cluster from anywhere not inside the cluster. If necessary, and the island size was large enough, we could create private-island networks for each island, so that each island would have its own network. Each private network would require it's own network hub.

Solution

- Use private and public networks.
- If for larger island sizes, network bandwidth is still an issue, use private island networks to reduce data read by all islands.

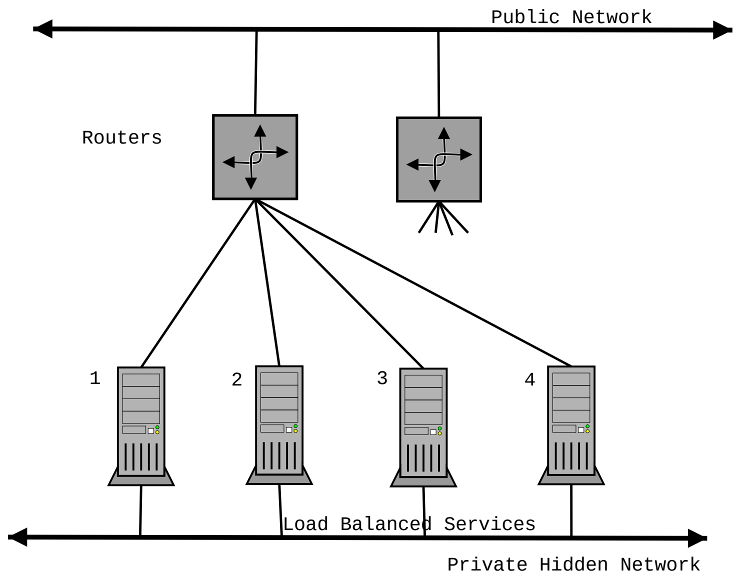


Figure 5.1: Linux Virtual Cluster Architecture

5.7.5 Legacy Integration

Legacy Integration did not affect the prototype as the system simulated the integration with the signaling hardware that provides SMSC systems with their communication link to the signaling stack. This would be an issue with a production SMSC but was not with the prototype.

Solution

- Research scheduling and queuing of messages to improve throughput.
- Investigate use of JNI [Micg] to interface with legacy software.

Chapter 6

Conclusions

We believe we have produced a system which addresses the majority of the requirements of RAS systems as set out in the preceding chapters, we will now examine the requirements one by one:

- Fault Tolerance

  By replicating the service over 2 generic nodes per island, we have ensured that the system can survive failure of one node in each island. This equates to N - 1 nodes failure per island. To increase the level of fault tolerance we simply add another node to the island. The equation to evaluate the level of fault tolerance of the systems is given by:
  (Number Of Island Members) - 1 = maximum number of permitted failures, Or simply N - 1.
  It should be noted that adding more nodes to each island would have a detrimental effect on performance, as the network bandwidth would be more saturated. With the advent of Gigabit Ethernet and the use of private networks as defined in Chapter 5 this would not be so much of an issue.

- High Availability

  The use of a cluster of machines means we can choose to bring nodes in or our of service and upgrade or update software and then allow them to rejoin the system. For the prototype this would involve bringing one node from each island out of service and then upgrading its software. Next we would remove the other node in each island and allow the upgraded node to rejoin. If the new node is ready before the changeover is made, this would be a 'Zero Latency' failover as all that is required is that the Input Router is told to make a change to its routing table.

- Load Sharing

  The Input Router of the prototype enforces round robin load sharing. By using a single method to return the next node for each island we have created a system that requires only one method per island to be changed to modify the algorithm used by the Input Router for selecting the next node for submitting a SMS message to. This was one of the goals as set out in section 1.4, we believe the system we have created is modular and easily extendible.

- Scalability

  As Chapter 5 shows the system is scalable for modestly sized clusters. The success of this is primarily due to the fact that each island in the cluster operates as part of a 'shared nothing' architecture. In this way, the throughput of the system, neglecting Input Router Load and Network load, will grow almost linearly.

6.1 Benefits

The use of a GNU/Linux platform and low or no cost software, has enabled us to produce a system with a particularly attractive low cost of ownership. The reuse of the standard features of EJBs, namely Concurrency, Transactions, Persistence and in the future clustering, has enabled a system to be produced in a rapid time. In fact the entire coding of the system was accomplished in a period of 9 weeks. The remainder of the time was spent understanding the requirements of the system, and designing a system that would be acceptable in terms of functionality and extensibility.
The choice of a very modular design has two main benefits, one was testing was simple, each component of the system was tested separately, using client classes. The other, is the fact that we can plug in new components to replace old ones as and when required. In fact, we can update EJB classes on the fly as they are being used on the Application Server.

The primary finding of the dissertation was that EJB has indeed a place in telecommunications software, although further research would be needed to investigate this further, but if we list exactly where EJB excels, we see:

- Transaction Management
- Resource Management
- Concurrency Management
- Persistence
- Security
- Scalability
- Interoperability

This list shows the components of a distributed system developers can reuse in their own code for 'free' by using the EJB architecture. This simply makes the EJB architecture a more productive development environment, as was seen by the development of the prototype.

6.2 Limitations Of The System

There were two area in which the system did not handle itself as a high degree RAS system should, these areas were:

- Static Network Topology

  The Input Router requires the hardcoding of the network topology in one method to function. This was due to time restrictions, but efforts have been made to ensure the easy extension of the system to handle dynamic network topologies. Most notably, the Input Router will allow the dynamic addition and subtraction of nodes in the cluster, the messages routing itself though will fail since it uses the method which has the hardcoded references.

- Node Failure

  The final prototype did not handle catastrophic failure of the nodes cleanly. This was due to exceptions thrown from the application server. If we had used an Open Source Application Server, e.g. JBoss, we possibly could have addressed this issue which impacted on the ability of the system to survive failure. This would involve the re-writing of small areas of the system to take into account the exceptions thrown. The system did not support the re synchronisation of nodes which were out of action for a time period. Again, a simple solution exists that is the standard checkpointing of the database at regular intervals, and the replaying of a log of transactions or writes that have occurred in the period of time that the server wishing to rejoin was out of action for.

6.3 Future Work

This dissertation, we believe has shown that future work and research is indeed warranted in the use of the EJB / J2EE architecture with telecommunications systems. The speed of development and re-use of existing software greatly reduced the development time of the system and also improved it's robustness. The performance of the system was considered to be acceptable since we have made it trivial to add more computational power to the system, by simply adding more islands to the system. We have addressed some perceived bottlenecks and also offered some possible solutions that would improve the performance of the system.

Bibliography

[Ass01] Mobile Data Association. Monthly UK chargable SMS messages. `http://www.mda-mobiledata.org/`, 2001.

[BEA] BEA. BEA's TUXEDO (Transaction Processing Monitor) Overview. `http://www.bea.com/products/tuxedo/index.shtml`.

[Boo98] G. Booch. *The Unified Modelling Language User Guide.* Addison Wesley, 1st edition edition, 1998.

[ea95] Gamma et al. *Design Patterns, Elements Of Reusable Object-Oriented Software.* Addison-Wesley, 1st edition edition, 1995.

[ea01] George Coulouris et al. *Distributed Systemsm, Concepts And Design.* Addison-Wesley, 3rd edition edition, 2001.

[Gro] Object Management Group. Introduction to OMG Specifications - Corba. `http://www.omg.org/gettingstarted/specintro.htm`.

[Gro01] Credit Suisse First Boston Technology Group. Logica Profile. `http://www.tech.csfb.com/eurotech/profiles/logica.pdf`, 2001.

[Hae00] Richard Monson Haefel. *Enterprise JavaBeans.* O'Reilly, second edition edition, 2000.

[HC01] R.M. Haefel and D.A. Chappell. *Java Message Service.* O'Reilly, 1st edition edition, 2001.

[Hop00] Jon Hopkins. Component primer. *Communications Of the ACM*, 2000.

[HR83] T. Harder and A. Reuter. Principals of transaction-oriented database recovery. *Computing Surveys Vol. 15 No. 4.*, 1983.

[IBM] IBM. CICS Overview. `http://www-4.ibm.com/software/ts/cics/`.

[Inc01] Hypersonic Inc. Memory based database. Website. `http://sourceforge.net/projects/hsqldb/`, 2001.

[Ins01] European Telecommunications Standards Institute. Website. `http://www.etsi.org/`, 2001.

[JBo01] JBoss. Open Source Application Server. `http://www.jboss.org`, 2001.

[Lar00] Grant Larsen. Component-based enterprise frameworks. *Communications Of the ACM*, 2000.

[MH99] Vlada Matena and Mark Hapner. *Enterprise JavaBeans Specification.* Sun Microsystems, v1.1 edition, 1999.

[MH01] Vlada Matena and Mark Hapner. *Enterprise JavaBeans Specification.* Sun Microsystems, v2.0 edition, 2001.

[Mica] Microsoft. The Distributed Component Object Model (DCOM). `http://www.microsoft.com/com/tech/DCOM.asp`.

[Micb] Microsoft. .NET Overview. `http://www.microsoft.com/net/default.asp`.

[Micc] Sun Microsystems. Enterprise JavaBeans(EJB) technology. http://java.sun.com/products/ejb/.

[Micd] Sun Microsystems. The Java 2 Platform, Enterprise Edition (J2EE) technology. http://java.sun.com/products/j2ee/.

[Mice] Sun Microsystems. Java DataBase Connectivity (JDBC) technology. http://java.sun.com/products/jdbc/.

[Micf] Sun Microsystems. Java Naming and Directory Interface (JNDI) technology. http://java.sun.com/products/jndi/.

[Micg] Sun Microsystems. Java Native Interface (JNI) technology. http://java.sun.com/products/jdk/faq/jnifaq.html.

[Mich] Sun Microsystems. Java Remote Method Invocation (RMI) technology. http://java.sun.com/products/jdk/rmi/.

[Mici] Sun Microsystems. The Java Technology Tutorial - online. http://java.sun.com/docs/books/tutorial/jar/.

[Micj] Sun Microsystems. JavaServer Pages (JSP) technology. http://java.sun.com/products/jsp/.

[Mick] Sun Microsystems. Remote Method Invocation (RMI) over Internet Inter-Orb Protocol (IIOP). http://java.sun.com/products/rmi-iiop/.

[Micl] Sun Microsystems. RMI and Java Distributed Computing. http://java.sun.com/features/1997/nov/rmi.html.

[Mic00] Sun Microsystems. *Blueprints*. Sun Microsystems, 1st edition edition, 2000.

[New00] Ted Neward. *Server-Based Java Programming.* Manning, 1st edition edition, 2000.

[Pos] PostgreSQL. Product Homepage. `http://www.postgresql.net`.

[Pro01] Linux HA Project. Linux High Availability Website. `http://linux-ha.org`, 2001.

[Red] RedHat. Company Homepage. `http://www.redhat.com`.

[Red00] Redhat. Linux Virtual Cluster / Server (LVS) Introduction. `http://ha.redhat.org`, 2000.

[Ren96] Paul E. Renaud. *Introduction to Client/Server Systems.* John Wiley And Sons, 1st edition edition, 1996.

[RO01] Ed Roman and Rickard Oberg. EJB and J2EE vs. Com+ and Windows DNA, A series of whitepapers by Ed Roman and Rickard Oberg. The Technical Benefits Whitepaper. `http://www.theserverside.com`, 2001.

[Rom99] Ed Roman. *Mastering Enterprise JavaBeans and the Java 2 Platform, Enterprise Edition.* John Wiley And Sons, 2nd edition edition, 1999.

[Sof] Ironflare Software. Producer of Orion Server Aplication Server. Company website. `http://www.orionserver.com`.

[ST96] Software Engineering Institute (SEI) Carnegie Mellon University Scott Tilley. Perspectives on Legacy System Reengineering. `http://www.sei.cmu.edu/reengineering/pubs/lsysree/node155.html`, 1996.

[Web01] Google Website. The Technology Behind Google. `http://www.google.com/press/guide/reviewguide_9.html`, 2001.

[wRP99] Perdita Stevens with Rob Pooley. *Using UML: software engineering with objects and components.* Object Technology Series. Addison-Wesley Longman, 1999. Updated edition for UML1.3: first published 1998 (as Pooley and Stevens).