University of Oslo
Department of Informatics

Evaluation of the EJB 2.0 and COM+ component models using a common testcase

Cand. Scient thesis

Jan Henrik Gundelsby,
Steinar Henrik Johnsen

February 1st, 2002

ABSTRACT

A component model is a defined set of services that assist the developer with traditionally difficult tasks such as transaction handling, synchronization, and security. The two component models Enterprise Java Beans 2.0 and COM+ are the two main competitors on the current market, and they are compared both from a service and performance perspective.

To benchmark performance, implementations of a common test case were conducted in both technologies. WebLogic 6.1 was used for the Enterprise Java Beans implementation. The set of services offered by the component models are compared systematically, and mapped according to the authors' own experience and to the test implementation.

The primary goal of this thesis is to provide an unbiased comparison of the two component models. The findings of this thesis indicate that the two component models are quite analogous with respect to services. From the performance point of view, COM+ proved to be somewhat faster than Enterprise Java Beans.

TABLE OF CONTENTS

LIST OF FIGURES

Number *Page*

ACKNOWLEDGMENTS

We would like to acknowledge the encouragement and constructive advice offered by our mentors, Knut Sagli and Arne Maus, in the completion of this thesis.

We are indebted to Genera AS for their support and encouragement in our studies.

We wish to thank Magali Rouyer for invaluable support and help in reading and assessing the readability of this thesis. Big thanks also to Bruno Kieba.

A huge hug goes out to our families for always having a hot meal ready for us. Finally, a magical and totally recyclable thanks to our most loved ones, Bente and Magali, for showing us patience in the final phase of this thesis.

GLOSSARY

Application Programming Interface (API)	A set of routines, protocols, and tools for building software applications.
Bean	Sun Microsystems calls a component a "Bean" (thus continuing their coffee analogy). A Bean is simply the EJB variation on the idea of a component.
Business transaction	A business transaction is a collection of methods that model or emulate expected behavior of the system.
Component	A physical, replaceable part of a system that packages implementation and conforms to, and provides the realization of a set of interfaces.
Component	A component is a reusable program building block that can be combined with other components in the same or other computers in a distributed network to form an application.
Component model	A component model is a defined set of services that assist the developer with traditionally difficult tasks such as transaction handling, synchronization, and security.
Component Object Model (COM)	Component Object Model (COM) is Microsoft's framework for developing and supporting program component objects
Component Object Model + (COM+)	COM+ is both an object-oriented programming architecture and a set of operating system services. COM+ is an extension of Component Object Model (COM).
Container	A container is an application program or subsystem in which the program building block known as a component is run.
Distributed system	A distributed system allows objects to be distributed through a heterogeneous network, which allows every component to cooperate.
Distributed Component Object Model (DCOM)	DCOM is a set of Microsoft concepts and program interfaces in which client program objects can request services from server program objects on other computers in a network. DCOM is an extension to COM.
Dynamic Link Library (DLL)	Packages containing object implementations used by COM+.

Enterprise Java Beans (EJB)	Enterprise JavaBeans (EJB) is an architecture for setting up program components, written in the Java programming language, that run in the server parts of a computer network that uses the client/server model.
Graphical User Interface (GUI)	A GUI is a graphical, rather than purely textual, user interface to a computer.
Integrated Development Environment (IDE)	A programming environment integrated into an application.
Interface Definition Language (IDL)	IDL is a generic term for a language that allows a program or object written in one language to communicate with another program written in a language unknown to the given program.
Java Archive (JAR)	A file format used to bundle components used by EJB.
Java Message Service (JMS)	JMS provides a reliable, flexible service for the asynchronous exchange of critical business data and events throughout an enterprise.
Java Naming and Directory Interface (JNDI)	An API for naming-service-independent resource location. This provides Java applications with a unified interface to multiple naming and directory services on the enterprise.
Java Transaction Monitor (JTS)	JTS is an API to ensure data integrity across several systems and their databases using two-phased commits and rollbacks.
Java Virtual Machine (JVM)	A JVM is a platform-independent programming language that converts Java byte code into machine language and executes it.
Just In Time (JIT) activation.	When JIT activation is activated in a component, the instance is not created before a call is made to the component, and the component is terminated immediately after the call is done.
Let's Buy Some Records business transaction	This business transaction is implemented as a persistent component.
Load balancing	Load balancing is dividing the amount of a computer's work between two or more computers so that more work is accomplished in the original amount of time. As a result, all users are usually served faster
Microsoft Message Queue (MSMQ)	The Microsoft Message Queue Server (MSMQ) guarantees a simple, reliable and scalable means of asynchronous communication freeing up client applications

	to perform other tasks without waiting for a response from the other end.
Microsoft Transaction Server (MTS)	The MTS manages application and database transaction requests on behalf of a client computer user
Middleware	The term middleware is used to describe separate products that serve as the glue between two applications.
New Customer business transaction	This business transaction is implemented as a stateless component that accesses the RDBMS.
New Order business transaction	This business transaction is implemented as a stateful component that accesses the RDBMS.
Object Pooling	The application server keeps a pool of objects instantiated to enhance performance. When the instance is terminated by the client, it does not get physically terminated, but it is put back into the object pool.
Object Request Broker (ORB)	The ORB is a broker that handles the request from a distributed object, and ensures that this request is carried out.
Performance	The effectiveness of a computer system, including throughput and individual response time.
Populate Shopping Cart business transaction	This business transaction is implemented as a stateful component with no RDBMS access.
Relational DataBase Management System (RDBMS)	A RDBMS is a program that allows creating, updating, and administering a relational database.
Skeleton	The skeleton is the generic server side code that allows communication between different components.
Stateful component	A stateful component is session-oriented, meaning that it maintains state across methods calls and transactions. It is to be considered a private resource for a client.
Stateless component	A stateless component is relatively short-lived and typically provides a single-use service, independent of which client is calling the service, e.g. adding a customer to the record shop.
Stub	The stub is the generic client side code that allows communication between different CORBA components.
TPC-C	A standardized transaction processing benchmark.

Transaction	A sequence of information exchange and related work (such as database updating) that is treated as a unit for the purposes of satisfying a request and for ensuring database integrity.
Unified Modeling Language (UML)	A general-purpose notational language for specifying and visualizing complex software, especially large, object-oriented projects

Chapter 1

INTRODUCTION

Systems based on components solve many of the problems that have arisen from the vast increase in the number of distributed systems. Distributed systems appeared as a consequence of the existence of several decentralized organizations and the introduction of the Internet. With component-based development, it is possible to buy and develop components, needed by a system, but also appropriate for reuse as part of a larger application. This is both time and cost efficient. Component-based technology is a way to ease communication between different applications across distributed networks.

A component model is a defined set of services that assist the developer with traditionally difficult tasks such as transaction handling, synchronization, and security.

The object of this thesis is to compare and contrast the two major component models currently available , namely Microsoft's Component Object Model+ (COM+) and SUN's Enterprise Java Beans (EJB). The EJB implementation used in the practical work of this thesis has been conducted with BEA WebLogic 6.1.

1.1 The authors

This thesis was written by Jan Henrik Gundelsby and Steinar Henrik Johnsen. The work was equally divided in the sense that Mr. Gundelsby primarily handled the EJB technology and that Mr. Johnsen primarily handled COM+. Both authors evenly participated in the discussion chapters, and they feel that their respective contribution to this work is equivalent.

1.2 Motives

The motives underlying the writing of this thesis are numerous. First of all, it is relevant to say that the authors find the topic of application server technology both extremely interesting and exciting. Another telling argument is that the application server communities are debating this topic on a daily basis, and the industry follows the debate with great interest.

Next, it is important to indicate that no objective and neutral comparison of these two component models have yet been made. Finally, as the authors were professionally involved with these two technologies, they were naturally inclined to choose this subject of study, which findings are of practical use to them, and their motivation was spontaneously enhanced.

The primary goal of this thesis is to provide an unbiased comparison of the two component models. The secondary goal is to help the reader, upon the study of this material, to acquire a broader perspective for making the most qualified technology choice considering a given project.

1.3 Delimitation

In such a large field as component models, it is important to remain focused on the most significant aspects. Additionally, in order to keep this thesis within the scope of a Cand. Scient. degree, further delimitations had to be made. All delimitations, except the tuning of the application servers, are delimitations of the performance test implementation. The theoretical research of these subjects is included.

Owing to lack of resources, clustering, and therefore load balancing and partly scalability are excluded from the practical part of this thesis.

Since event-driven communication was not implemented in WebLogic (the EJB implementation used throughout this study) at the time of performance testing, it could not be part of the test implementation.

The security aspects of the application servers are very extensive and they are similar for both technologies. By excluding security in the implementation, the security overhead in the comparison is not an issue.

Finally, the application servers are only tuned to have the same parameters set, and not to improve performance. The tuning of application servers is a huge area, and is also beyond the scope of this thesis.

1.4 Overview

Chapter 1 first introduces the reader to component technology. Besides, it presents the motives for writing this thesis as well as some delimitation. It also provides an overview of all the chapters and appendixes presented in order to complete the thesis.

Chapter 2 provides the reader with a summary of technologies that have been, and still are in use for component-based technology. It defines concepts and background terminology, and introduces common conceptions. Finally, it describes the concept of benchmarking.

Chapter 3 describes the approach to writing the thesis. First, it describes the applied research methods, subsequently the methodology of the development. Finally, it specifies the implementation setup and the way in which the tests are conducted.

Chapter 4 presents the results found when comparing side-by-side the two technologies. It points how the technologies handle the runtime, development and external qualities. Finally, it presents the performance results.

Chapter 5 discusses the findings of chapter 4.

Chapter 6 is devoted to an evaluation of the authors' approach. It compares the results with existing work, and examines the hypothesis (common conceptions).

Chapter 7 contains the conclusion of the thesis. It presents a summary of the most significant findings.

Chapter 8 describes further research that could be conducted, induced by the material presented in this thesis.

Appendix 1 presents the software, hardware and platform used in the performance test.

Appendix 2 contains a sample log file of the performance test.

Appendix 3 provides a comparison table of COM+ and EJB's most fundamental qualities.

Chapter 2

BACKGROUND

This chapter presents a description of COM+, EJB, and the technologies from which they arose. It is observed the way component-based technology came to be, as well as its evolution throughout the years. At the end of the chapter, common conceptions about the two technologies are presented, and finally benchmarking is explained.

2.1 Component-based development

There exist countless components and possibilities available for use. In a word processor, such as Microsoft Word, several components are present: the thesaurus, graphics viewing, printing, and graph functionality, undo/redo functionality, etc. These components are independent of the word processor component and can be used freely by other isolated applications within Windows. This is how components are reused, hence saving valuable development costs and time in the development of new applications. System maintenance is simplified and becomes less time consuming thanks to the easier localization of potential problems.

Purchase or reuse allows the developer to reuse functionality in several projects, reducing the cost and increasing the time effectiveness of component-based development. The alternative to developing themselves the whole environment, as opposed to buying components, will be chosen by other developers who will reuse, at a later time, in other projects.

A component, typically, but not always, works under several operating systems (UNIX, Linux, Windows, Mac OS etc.). A component can also communicate with any other given component through a standard interface; hence a component can be programmed to communicate with other components by exchanging information and functionality.

2.2 What is a component?

> *"Oh no!! You're making graph support in both Word and PowerPoint. Implement it once, and find a way to reuse it".*
> **Bill Gates, Microsoft (The birth of OLE)**

A component is a widely used term. The authors' interest, in respect of the scope of this thesis, resides in the context of software, although the meaning of the word *component* can then be extensive as well: class libraries, encapsulated software modules, CASE models, pre-built applications, etc. Their common denominator is that they can be combined with other components so as to shape and create an application.

2.2.1 The component – a superior explanation

> **Definition [OMG]:** *"A physical, replaceable part of a system that packages implementation and conforms to and provides the realization of a set of interfaces. A component represents a physical piece of implementation of a system, including software code (source, binary or executable) or equivalents such as scripts or command files"*

A standalone component is not an application but can be combined with several other components in order to shape one. It has to encapsulate its implementation and offer a standard interface to communicate with other components. Components can share methods, independently of the component implementation programming language or the underlying operating system. A component is able, through its interface, to communicate with other components. Lastly, a component should contain everything it needs to complete the tasks it is meant to accomplish (modular).

2.2.2 The component market

At the time of writing this thesis, Microsoft is still the leader of the component market with their Windows-specific components. They provide various encapsulated software in the forms of Visual Basic Controls (VBXs), Object Linking and Embedding (OLE) and OLE Custom Controls (OCX). A component is sold from about ten up to tens of thousands of US dollars.

There are many components available on the market – developed with Microsoft's tools, Sun's tools or with the aid of the Common Object Request Broker Architecture (CORBA). These components vary in complexity, from simple buttons, through a graphical user interface (GUI), to more advanced software packages such as a database interface.

2.3 What is a component model?

The popularity of distributed systems has compelled several different component architectures and technologies to handle communication between components in a distributed system.

Different sources give different definitions of a component model. If one defines it as "a complete component-based architecture for distributed systems", some delimitation has been done.

> **Definition [IBM]:** *An architecture and a set of APIs that allow developers to define software components that can be dynamically combined together to create an application.*

The Application Programming Interfaces (API) in this definition can be explained as a set of system services that are offered by the component model.

Sun [SUN] defined their component model as a set of services:

- *Component interface and discovery.* A component can communicate with another component, discover its characteristics and the way to communicate with it. This renders the possibility for various providers to implement components communicating with each other, without directly knowing which components are cooperating.
- *Component properties.* A component should publicly offer its properties to other components.
- *Event management.* A component should be able to deliver a message to one or more components to notify that an event (e.g. the user pushed a button) has occurred, so that the component(s) receiving the message can respond to the event.
- *Persistence.* The possibility to store the component state for later use.
- *Application building support.* Components should not only be easy and flexible to introduce into a distributed network, but users should be able to easily create new components and view properties of existing ones.
- *Component packaging.* Since components often have several associated files, such as icons or other graphic files, the Sun component model includes facilities to pack files together in an easily administrated and distributed format. Sun calls the component packages Java Archives (JAR).

This defines Sun's set of services for a component model. Other component models offer other sets of services.

Another way of defining a component model is to determine what the industry uses. In order to decide what component architectures satisfy the requirements to be a component model, one could observe which component models application servers use.

An application server offers services, as interfaces, targeted for an accepted component infrastructure. Application servers offer services for the Enterprise JavaBeans (EJBs) model from Sun Microsystems [SUN], the Component Object Model (COM) [COM95] from Microsoft Corp. Another alternative is to combine the use of Java and CORBA to achieve a simple distributed component model although it has the inconvenience of lacking the complexity of EJB and COM+ [ASU 99].

A component model is defined as joint characteristics between existing services for component architectures that are currently supported by application servers, namely EJB and COM+.

2.4 What is a distributed system?

> ***Definition [BLAIR97]:*** *A distributed system consists of a number of autonomic computers that does not share primary storage, but cooperates through asynchronous messages over a network.*

A distributed system allows objects to be distributed through a heterogeneous network, which allows every component to cooperate. A distributed system contains *nodes* that execute calculations. A node can be a PC, a mobile phone or any other device. The Internet is an excellent example of a distributed system.

One of the reasons for the increasing popularity of components and component architectures is the extended usage of distributed systems. Distribution raises numerous new challenging issues but, sometimes, applications and systems are simply distributed by nature, e.g. in mobile systems.

2.4.1 Why use distributed systems?

One of the reasons why distributed systems arose is the existence of decentralized organizations, that is, organizations or companies with offices in different locations, e.g. multinational companies.

A number of applications share (distributed) components; providers rent services from each other in order to achieve reuse and limit maintenance. Statistics shows that the maintenance costs of systems represent in average

twice the development costs. Maintenance is therefore a factor important to keep at a minimum.

Data can be distributed, often because of administrative reasons. E.g. data, which would be conveniently accessed from outside the system, has to be stored locally because of the security policy.

By using distributed systems, multiple processor usage can be exploited, hence increasing performance. Also, a given application may need the unique properties of one specific computer; and the distributed application can use the scalability and heterogeneity of the distributed system.

Users of current systems typically execute shared objects on one or more shared servers, see Figure 2-1. The users communicate through the same application.

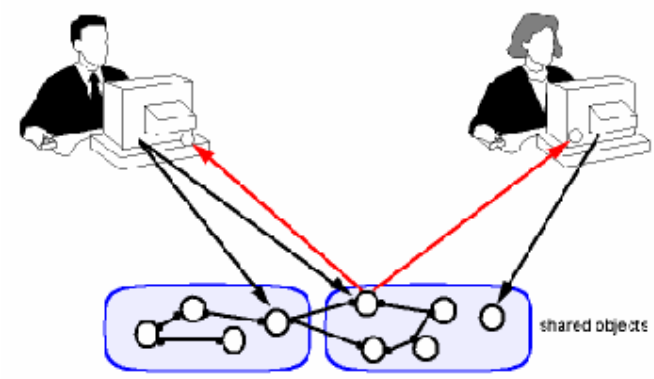

Figure 2-1: Communication on shared servers

2.4.2 Reality of distributed systems

In distributed systems, there exist some fundamental properties; they are taken for granted in a local program where all logic happens in the same operating system and in the same process.

Table 2-1 points to a few differences between objects that are local in the same process and objects that cooperate in different processes or machines.

	Local	Distributed
Communication	Fast	Slow
Error	Objects fail simultaneously	Objects fail separately
Parallelity	Only when multi-threaded	Yes
Security	Yes	No

Table 2-1: Local vs. distributed systems

The communication between two machines will be noticeably slower than between two local objects. If two objects are distributed in different processes, the objects can fail separately from each other, thus processes can execute independently (and unknowingly) of the other process success or failure. Distributed objects act as multi-threaded objects on a local system; all distributed objects operate on their own thread. While, with distributed objects on different machines, security mechanisms are often needed to authenticate the objects' identity, these are not necessary to consider if two objects are in the same process.

2.4.3 Object-oriented distributed systems

An object-oriented distributed system is the product of two technologies: networking and object-orientation. An application built in a distributed object environment means that "the network is the computer". Objects are distributed to different computers through a network, and still used locally within the application through an interface. In object-oriented distributed systems, the objects can be components that encapsulate their implementation and offer an interface outwards.

2.5 Background Terminology

After some basic concepts are examined, a closer study of the component models will be presented.

During the last 20 years there has been a change from having centralized servers to using distributed systems. The conception of communication between computers constitutes the foundation for distributed systems.

2.5.1 Transmission Control Protocol/Internet Protocol (TCP/IP)

There exist several network standards, and the family of Transmission Control Protocol/Internet Protocol (TCP/IP) stands out as the prevailing standard protocol. A protocol is simply a determined way of executing a task.

Communication protocols specify how computers (or other devices) cooperate by exchanging messages.

Innumerable TCP/IP-protocols exist and each protocol is usually represented as one to four layers, see Figure 2-2. Each protocol layer has a specific function, and functionality becomes more primitive in the lower levels. Typically, the upper layers are involved with the user needs, while the lower layers are more involved with technology.

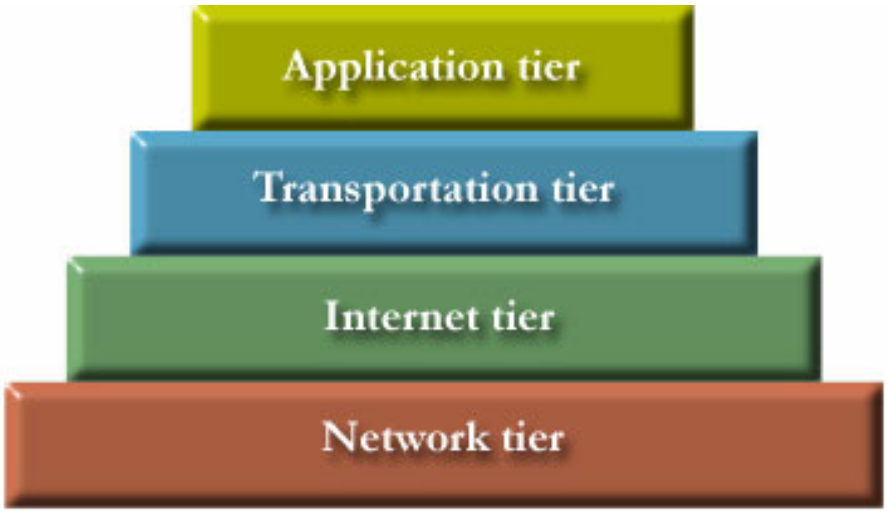

Figure 2-2: The four layers of the TCP/IP protocol

Because the functionality of the network layer is very primitive, this layer often becomes transparent to the user. This layer is responsible for the restructuring of data to a form suitable for network transmission, and for connecting logical addresses to physical addresses. A logical address is also known as an IP-address, or an IP-number that uniquely identifies a network device.

The Internet layer consists mainly of two protocols: The Internet Protocol (IP) and the Internet Control Messages Protocol (ICMP). IP standardizes the content and formats the data packages for transmission. It chooses a suitable route, fragments and reassembles the data packages for data forwarding to an appropriate higher level protocol. ICMP takes charge of transmission speed to insure that the receiver can keep up with the transmission pace. It detects if the receiver does not exist, reroutes network traffic dynamically and offers an echo service used for IP-protocol verification on external systems.

The most commonly used version of IP is the Internet Protocol Version 4 (IPv4). IP Version 6 (IPv6) is starting to be supported. IPv6 allows creating

longer addresses, hence increasing the number of Internet users. IPv6 includes all IPv4 services, and all servers supporting IPv6 packages also support IPv4.

The transport layer, similarly to the Internet layer, consists mainly of two protocols: TCP and User Datagram Protocol (UDP). TCP takes charge of error checking and retransmission in order to increase transmission reliability. Additionally, it collects packages from a continuous stream of data and puts it into a sequence. Finally, it delivers data to the processing application. UDP also offers delivery of a package to an application although it lacks the reliability and connectivity of TCP. TCP also guarantees data delivery in order, with no duplicates, and no data corruption, while UDP does not.

The application layer consists of all the applications that use the data delivered by TCP/IP. Some applications, such as e-mail, have been standardized, while other applications are specialized, e.g. the messenger service named ICQ [MIRABILIS].

TCP/IP is an important technology that lays the foundation for distributed programming. Nearly all architectures and technologies use TCP/IP as their foundation.

2.5.2 Sockets

The basic form of information exchange between devices are *sockets*; a facility offered by a TCP/IP network. Sockets allow sending and receiving messages, or datagrams, over a TCP/IP network. Sockets are the assembly language of TCP/IP data transfer, where Remote Method Invocation (RMI) and CORBA (explained in section 2.5.6 and 2.6.1) represent the high-level language. If the transmitted data are simple, such as an ASCII text, sockets are an excellent choice. When transmitting complex objects, socket programming becomes complicated and high-level solutions are preferred.

2.5.3 Architectures

A basic information system without a network consists of at least one unit such as a PC. A network system is composed of at least three parts: a client, a network and a server. The user operates the client through a user interface. The server holds the resources, such as data or programs needed to satisfy the client's demand. Finally, the network binds the client and server together.

The first of the two traditional architectures is the mainframe architecture. It arose in the early 60s, mainly as a consequence of expensive hardware. All computation is carried out on a server (mainframe). The other traditional architecture is the file server architecture. It is considered as a modern architecture but is traditional in the sense that it has existed for a long time. In the file server architecture, the clients do the computation, and a relatively small server functions mostly as a storage medium for joint client data.

The client-server concept surfaced in the early 90s. In this more balanced architecture, the client and the server share the computation, and replaces flat files with relational databases. The Structured Query Language (SQL) made the client-server systems more scalable than file server systems because it was no longer necessary to transfer large amounts of data over the network [SQL92]. Instead, only the necessary data are transferred, e.g. a database table, database row or database field.

The client-server architecture would face a major challenge if a client used several servers, usually servers with different operating systems and/or different database engines. As a consequence, the client had to be equipped with specific drivers for every configuration. To address this issue, the three-tier-architecture was introduced. It is a middleware solution put as a layer between the server and the client. Clients are equipped with a simple driver (thin driver) that communicates with a middleware server. The middleware server again communicates with the server. It makes the clients responsible for the user interface, the middleware servers responsible for computation and business logic. The servers have responsibility for storing data in one or more relational databases.

2.5.4 Object distribution architecture

Object distribution architectures apply the middleware concept by encapsulating data in object interfaces. Implementation details are concealed from the user of the object; distributed object architectures support location-, platform- and programming language transparence.

2.5.5 Remote Procedure Call (RPC)

Remote Procedure Call (RPC) abstracts the communication interface for a procedure call. Instead of working directly on sockets, it creates the illusion of calling a local procedure. The call's arguments are packed together and transmitted to the external object.

2.5.6 Remote Method Invocation

The successor of RPC is Java Remote Method Invocation [RMI 97] which is based on the principles of RPC. It has been adapted to distributed object systems, with the possibility of attaching one or more objects to an enquiry. Enquiry object serialization is how Sun terms it.

RMI is a language-dependent architecture that offers Java-to-Java distributed applications. One of the most significant advantages of RMI lays in the use of the Java object model, which provides language independence and platform heterogeneity.

RMI is the foundation for the distribution mechanisms in the Sun component model Enterprise Java Beans.

2.5.7 eXtensible Markup Language (XML)

XML [XML 98] is becoming an accepted standard of data exchange, especially between different platforms. It looks like Hyper Text Markup Language (HTML), although there is a major difference: HTML is the presentation of data, while XML is concerned with the specification of data. XML provides the tools to describe and deliver structured data from any application in a standard and consistent way. XML does not replace but rather complement HTML.

2.5.8 Simple Object Access Protocol (SOAP)

The increasing popularity of the Internet has created new problems with respect to security and firewalls. Microsoft made Distributed Component Object Model (DCOM) run on top of RPC using the TCP/IP protocol, to make it functional through a firewall. The idea is to communicate in an open and neutral way over port 80 (HyperText Transport Protocol (HTTP), the underlying protocol used by the World Wide Web, which is normally open in firewalls). SOAP running over HTTP does exactly this, by using XML as a language for passing parameters [SOAP99].

Certainly the most substantial characteristic regarding SOAP/XML is that it is an open standard driven mainly by Microsoft. In other words, objects communication with any other XML object running on any platform can be written.

2.5.9 Web Services

Web services is an emerging technology driven by the purpose to securely expose business logic beyond the firewall. Web services can encapsulate existing business processes, publish them as services, search for and subscribe to other services, and finally exchange information. Although there are many opinions as to what constitutes a "Web service," each definition shares some common ground; a Markup Language component transported over the Internet via HTTP [WHATIS].

The Gartner Group[GARTNER] defines a Web service as "*a software component that represents a business function (or a business service) and can be accessed by another application (a client, a server or another Web service) over public networks using generally available ubiquitous protocols and transports (i.e. SOAP over HTTP).*"

The Web Service specification is driven forward by Microsoft and IBM.

2.5.10 JINI

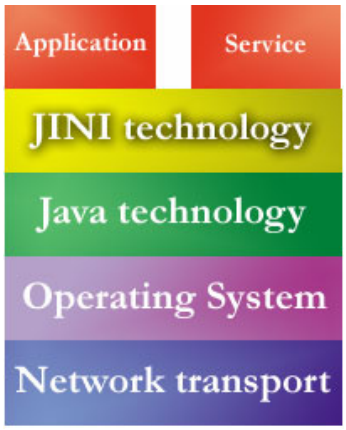

Figure 2-3: JINI layers

JINI [JINI] is a technology that enables devices to communicate without any form of planning, installation or human interaction. Each of these devices have a specific interface, which ensures compatibility and reliability. A device can be a PC, a refrigerator, a TV, etc.

The JINI technology utilizes a directory service in which all the devices and services are registered. When a device is enabled, it automatically goes through an add-in protocol called discovery and join. First, the device discovers the directory services, and then it sends to the directory server an object that implements the interfaces for the device services (join).

When a device or a person wishes to make use of a service, the object is copied from the directory service to the device. The directory service becomes a communicator of the service.

Java is JINI's programming language, and the devices in a Jini network are connected using Java RMI.

According to SUN, JINI is powerful enough to build a fully distributed system in a network of workstations, and at the same time compact enough to enable smaller consumer articles to communicate (e.g. a mobile telephone network).

2.5.11 Wireless Application Protocol (WAP)

WAP is a specification for a set of communication protocols. Its objective is to standardize the way wireless devices, such as mobile phones and radio transmitters, can be used with the Internet. WAP includes services such as e-mail, WWW, newsgroups, IRC and other. Such services have been available for a long time, although not standardized, which makes it practically impossible for a vendor to support all implementations. WAP is the result of cooperation between Ericsson, Motorola, Nokia and Unwired Planet.

2.6 Component architectures

There are currently three complete component models available: CORBA 3, COM+ and EJB.

2.6.1 CORBA

In 1990, Object Management Group (OMG) developed a specification for distributed objects that offers location transparency. CORBA is not an implementation, but rather a specification written by OMG, which means that there are several independent implementations of CORBA by different vendors. In any CORBA implementation, the communication between objects is handled by an Object Request Broker (ORB), which is present on both client and server sides. It enables the developer to perform calls to objects without knowing their exact location, what language they are written in or what operating system (OS) they are running on [CORBA97].

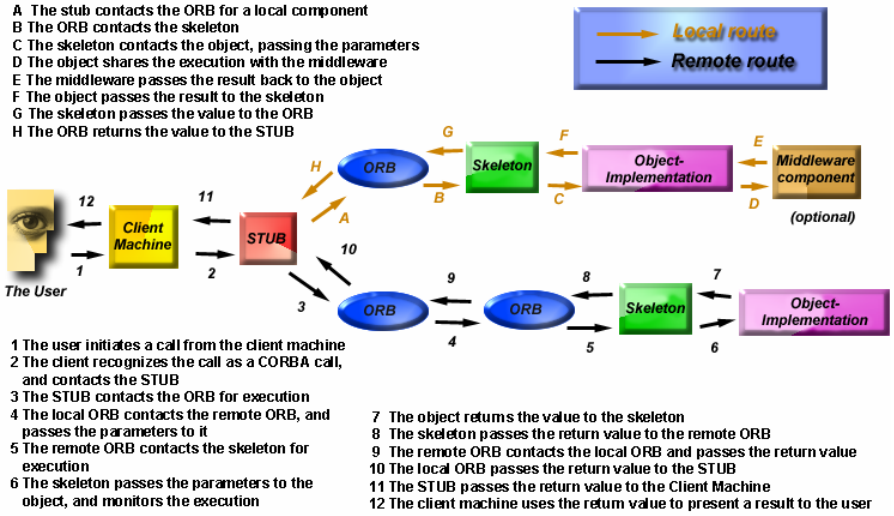

Figure 2-4: Client invocation of a server method using CORBA

2.6.2 Overview of the most significant components of CORBA

- **Object implementation**
  The object implementation is the server component. It is the application that contains the business logic.
- **Skeleton**
  The skeleton is the generic server side code that allows communication between different CORBA components.
- **Client**
  The client uses the server to perform services. Traditionally, this is the visual application that the user sees on a client machine.
- **Stub**
  The stub is the generic client side code that allows communication between different CORBA components. This generic code is generated for each function that the client wishes to perform on the

server. The stub performs the communication between the client and the ORB.

- **ORB**
  The ORB is a broker that handles the request from a distributed object, and makes sure that this request is carried out. To accomplish this, the ORB utilized the CORBA Interface Repository, where all the services of a component are stored. To present services to the world, the CORBA Interface Definition Language (IDL) can be used or the compiler can handle it. IDL is a generic term for a language that allows a program or object written in one language to communicate with another program written in a language unknown to the given program.
- **Internet Inter-ORB Protocol (IIOP)**
  IIOP is the a protocol designed and optimized for transmitting information from and to distributed components using CORBA. This protocol allows, as the name implies, that the Internet is used as the medium of transportation. IIOP also provides ORBs from different vendors with the ability to communicate. The ORB also handles persistency, transactions, security and the other services offered by CORBA.

2.6.2.1 Services and advantages

CORBA is an open specification. There are implementations for most platforms (UNIX, Linux, Windows, Solaris, etc.) and hardware configurations. The Java objects are platform independent, and communicate with EJB and DCOM/COM+.

CORBA offers security services such as encryption, authentication and authorization. CORBA supports nearly all programming languages on most platforms. This ensures availability for all projects.

2.6.2.2 CORBA 2 – a component model?

CORBA 2 objects can be considered as components where the interface and the communication between these objects are specified. The CORBA 2 specification does not include the implementation of the objects, and therefore cannot be regarded as a complete component model. CORBA and JAVA/RMI frame a complete component model.

2.6.3 CORBA 3

The specification for CORBA 3 [CORBA3] was completed in the fall of 2001. CORBA 3, the latest release, adds a Component Model, a Quality of Service control, a messaging invocation model, and tightened integration with Enterprise Java Beans and the Java programming language. In a press release on April 6, 1999, the OMG officially announced that the EJB model would

also serve as a subset to CORBA's Component Oriented Middleware platform [OMG99].

CORBA 3 adds a Java-to-IDL mapping, which defines IDL interfaces for Java objects. This permits Java programmers to use the OMG standard protocol IIOP for their remote invocations. EJB is based on the same CORBA 3 specification: EJBs interoperate on the wire using IIOP.

In CORBA 2, the implementation of the objects was outside the specification; hence, CORBA was not a component model. CORBA 3 adds the CORBA Component Model (CCM), which provides the integration with EJB. EJBs are Java-language basic level CORBA components, and applications can be built from any combination of EJBs and CORBA components. Indeed, the required application programming interface (API) for Java CORBA Components is EJB 1.1.

The CCM specification includes a comprehensive forward and reverse mapping of EJB and CCM operations, not only method invocations, but also container, factory, finder, and other infrastructure operations. CORBA components supply a container that integrates with EJB, handles transactions, security, persistence, interface, and events. This means that EJBs can function as basic CORBA components, and that Java-language level basic CORBA components can function as EJBs. By *basic* CORBA components is designated a model that corresponds nearly exactly to EJB, accompanied with a higher level that adds multiple interfaces, navigation, event handling, and advanced persistence.

The new features in CORBA 3 include:

- CORBA 3 handles communication over SOCKS, a protocol used by proxy servers.
- There will be two new methods of finding an object instance through the Interoperable Name Service. The name service has two URL-based methods, iioploc (that refers to the location of the object) and iiopname (which refers to the name of the object). An example would be: `iioploc://www.ifi.uio.no/NameService`
- CORBA 3 allows asynchronous messages. The client can set the timeout for a desired reply and the priority of a message.

The few implementations that are currently available are still beta versions, and no major software house has yet committed to implementing the CCM.

2.6.4 Distributed Component Object Model (DCOM)

Microsoft's distributed object protocol DCOM [DCOM98] is an extension of the Microsoft COM architecture [COM95]. DCOM offers interaction between objects registered in a network on different servers.

COM was introduced as a method of letting clients dynamically use and share objects implementation. The Dynamic Link Library (DLL) was introduced as packages containing the implementation. The COM interface appears to the client as a pointer to a virtual function table in a block of memory and hides the details in the implementation.

To meet the growing demands for distributed systems, Microsoft developed DCOM, which is an extension of COM. Since DCOM is an extension of COM, everything formulated about COM in this thesis applies to DCOM as well.

According to Microsoft, all distributed object architectures should have the following properties:

- **Interface definitions.** In DCOM, the objects communicate with each other through interfaces. An interface in DCOM is a collection of methods that define a contract. The interface also defines the behavior of an object, regardless of what language it is implemented in. COM objects can be implemented in the most common languages: Ada, C, C++, Java, Modulo-3, Pascal, etc. The Microsoft Java Virtual Machine (JVM) can be used to obtain a natural access to COM objects from Java; however this JVM is no longer updated since the last Sun's lawsuit against Microsoft. JVM is a platform-independent programming language that converts Java byte code into machine language and executes it. The interface used in COM, Interface Definition Language (IDL), is language independent. The interface description can be written manually, however most tools can make type libraries that include the IDL interface. These tools include Visual Basic (VB), Visual C++, Visual J++ and Inprise's Delphi. The fact that the tools handle so much of the work for the developer has contributed to COM's success. Unlike CORBA, DCOM is not tied to IDL.
- **Catalog services.** When a COM-client knows the name of the component that it wishes to utilize, it can use the COM catalog service to look up the class ID of the component. It can then find whether the component is run locally or remotely. The combination of the COM catalog service and the Windows registry constitutes the catalog service for COM.
- **Marshalling**. Marshalling is a key concept in localization transparency, and is the process of putting information into packets before sending them to another component, which in turn unmarshalls these and routes them to the destination component. The marshalling is done with a proxy and stub DLL.

- **Object persistence.** COM objects are normally stateless objects. If the developer chooses persistence, it has to be implemented programmatically.
- **Security.** DCOM is closely tied to the NT security model, both for administration and development.

COM and DCOM offer the benefit of being released on all computers running any Windows version newer than Windows 3.11. All the 32-bit Windows systems have DCOM support.

2.6.4.1 DCOM and Microsoft Transaction Server (MTS)

MTS is a container for DCOM components, and offers services to these components. DCOM combined with MTS represent a complete component model, as DCOM communicates with MTS that provides services on its behalf. MTS is DCOM's server environment; DCOM is MTS's protocol. In the same way as EJB utilizes RMI as its protocol, MTS functions as an application server but offers additional services that make DCOM a complete component model.

2.6.5 COM+ - a new generation of COM

COM+ [PLATT99] is the successor of the COM architecture, with a new generation technology. COM+ was made for Windows 2000 – it is COM with multiple inheritance, a new runtime environment and extensions for the languages, which enables implementations in more languages.

COM+ can be described as the combination of COM and MTS, with the addition of a series of new services.

COM+ is integrated in Windows 2000 and its improvements over COM can be categorized in two sections: improvements/updates and new services.

The three most important improvements/updates are:

- **Transaction services.** A mechanism to keep data integrity in a distributed system despite communication or hardware failure.
- **Security services.** While COM used the Windows NT security model, COM+ approaches the issue in an administrative manner. Mostly everything can be done administratively and little code is required.
- **Synchronization services.** One of the problems encountered with distributed services is the concurrent use of multi-threaded objects. COM+ offers services to synchronize components in an administrative manner: no code is required. However, to achieve complex concurrency control, a developer may choose to do it programmatically.

The four new services offered in COM+ are:

- **Queued components.** This service represents a means of communication that allows COM+ clients to call COM+ components that are not necessarily available at the time when the call is made. When the COM+ component being called becomes available, the system ensures that the call is carried out.
- **Event services.** This service is built around a subscriber and a publisher. The publisher is a component that offers information (e.g. updates of stock prices). A subscriber is a component that receives these updates and then publishes it to its subscribers.
- **In-memory database.** This service offers a way to improve the performance on frequently used database tables.
- **Load balancing.** This service enables the load of a clustered server solution to be as even as possible by directing each call to the server with the least central processing unit (CPU) load. Load refers to the amount work being carried out by the CPU.

It is still possible to write standard COM components with the same tools, the main difference being the tight integration to the operating system.

2.6.6 Microsoft .NET

This new product from Microsoft is the "successor of COM+", and was released in December 2001. The .NET framework is therefore not included in this thesis.

The goal of the Microsoft .NET framework is to simplify the process of building Web applications and Web Services. Web Services allows communication over the HTTP protocol, usually with the aid of the Simple Object Access Protocol (SOAP) [SOAP99]. Since Web Services and the SOAP technology are already supported in WebLogic's EJB implementation, communication between these two technologies is being facilitated.

2.6.7 Enterprise Java Beans

Enterprise Java Beans is a specification from Sun Microsystems, and already exists, at the time of the writing of this study, in version 2 final draft [EJB2.0]. The first draft specification, version 1.0, was released in December 1997.

Like CORBA, EJB is an open specification, not an implementation. There are several implementations available today, the most known are:

- Bea System – WebLogic Application Server
- IBM – Websphere
- Inprise – Inprise Application Server
- Lotus – Notes/Domino
- Netscape – Netscape Application Server (Kiva)

- Oracle – Oracle Application Server
- Sun – NetDynamics
- Sybase – Enterprise Application Server

Since EJB is language dependent to Java, it presents the advantage of using the java object model and the Java 2 Enterprise Edition (J2EE) [J2EE] framework for many of its services. J2EE is beyond the scope of this thesis; however the parts of J2EE that are used by EJB are naturally explained.

The objective of EJB, as stated in the introduction of the specification is:
"Applications written using the Enterprise JavaBeans architecture are scalable, transactional, and multi-user secure. These applications may be written once, and then deployed on any server platform that supports the Enterprise JavaBeans specification."

To meet this objective, the key features of EJB are:

- **Transaction management**. It may be vendor specific although EJB uses Java Transaction Service (JTS) /Java Transaction API (JTA). JTS is an API to ensure data integrity across several systems and their databases using two-phased commits and rollbacks. JTA specifies standard Java interfaces between a transaction manager and the parties involved in a distributed transaction system: the resource manager, the application server, and the transactional applications.
- **Security**. The Java object model security is used.
- **Portability**. EJB supports most platforms including Windows, Linux and UNIX.
- **Event-driven messaging.** The Java Message Service (JMS) [JMS] is used to implement this feature. JMS provides a reliable, flexible service for the asynchronous exchange of data and events.
- **Naming and catalog service**. Java Naming and Directory Service (JNDI) is utilized in order to perform this property. It is an API destined to naming-service-independent resource location. It provides Java applications with a unified interface that allows access to multiple naming and directory services in the enterprise.
- **Interoperability**. The EJB specification uses Java RMI as the default protocol to invoke Enterprise Beans over a network. Additionally, the specification refers to CORBA/IIOP mappings to enable CORBA clients to invoke Enterprise Beans. However, EJB is not tied to these solutions and can be run over any protocol (e.g. HTTP or DCOM) in order to support a multitude of clients (see Figure 2-5).
- **Scalability**. Mechanisms for scalability, such as load balancing and object pooling are included. An EJB component instance from a pool of shared instances may be used when a client makes a call. As soon as the request is serviced and the reply is sent back to the client, the

actual EJB component is returned to the pool. It may not be destroyed. This process is called object pooling.

- **Stateless, stateful and persistent objects.** A bean, the name of an EJB component, falls into three categories: the Stateless Session Bean, the Stateful Session Bean and the Entity Bean.

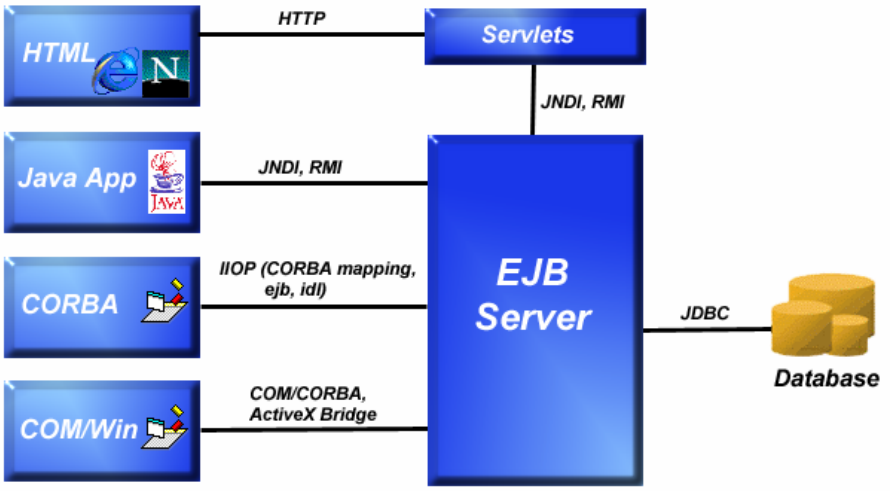

Figure 2-5: EJB interoperability

An EJB container creates, manages, and destroys EJB components. The EJB specification [EJB2.0] uses the terminology EJB Container: *An EJB Container (Container) is a system that functions as the "container" for enterprise beans. The Container is a part of the target operational environment, its runtime provides the deployed enterprise beans with transaction and security management, network distribution of remote clients, scalable management of resources, and other services that are generally required as part of a manageable server platform.* The EJB Container provider is an application server implementation, e.g. BEA with WebLogic server. This is the EJB Container provider utilized by the authors in the test implementation. The Container expression can also be used in a COM+ or CORBA context, i.e. the runtime environment in COM+.

Stateless Session Bean

The Stateless Session Beans are designed to be easy to implement and to have a low resource usage. If any state is to be held, it will be done on the client side, which leaves the server scalable. Because this type of Enterprise Bean is not stateful, it does not have a tie to a specific client; hence any client can use the first instance of this bean that it can find.

Stateful Session Bean

The Stateful Session Beans imply that the server has to keep track of which specific bean every client uses. Consequently, every Stateful Session Bean is

created exclusively for a specific client, and is to be considered as a private resource for that specific client (even though it can be shared). A Stateful Session Bean is a logic extension of the client, but the load is shared between the client and the server.

Stateful Session Beans will not survive a server crash or other Byzantine errors (some implementations of EJB have mechanisms to allow this).

Stateful Sessions Beans have access to persistent resources (like databases and files) but, unlike the Entity Beans, they do not represent the data. A Session Bean can access these persistent resources through a database API or an Entity Bean.

Entity Bean

Entity Beans are persistent objects that represent data in a permanent storage. An Entity Bean lives in an EJB container in the same way than a table instance lives in a database. The EJB containers exist for different data sources (Oracle, CISC etc.), but this does not represent an issue the developer needs to address. However, entity beans can be bean managed or container managed, or more specifically bean/container managed persistent and/or have bean/container-managed transactions. In bean-managed transactions, the developer implements the database code or chooses when to start and stop a transaction programmatically. In container-managed transactions, this is up to the container.

Unlike the Stateful Session Bean, the Entity Bean can be accessed by several clients simultaneously. Because an Entity Bean lives in a permanent storage, it will not be affected by a server crash or other Byzantine failures.

2.7 Common conceptions (hypothesis)

By reading various literatures and researching the subject of this thesis, the authors developed the following common conceptions concerning application server differences between the component-based models COM+ and EJB.

H1: COM+ and EJB should have identical (linear) performance curves, although EJB should be slower because of the Java runtime overhead. COM+ has stronger ties to the operating system; hence it should offer shorter response times. In addition to that, Java is considered as slow.

H2: EJB presents more features (such as state and persistence handling) for the developer, but is more arduous to learn properly.

H3: COM+ is less reliable than EJB, because of the history of instability of the Windows operating system.

These statements will be corroborated or invalidated in the discussion, Chapter 5.

2.8 Performance benchmarking

With performance as one of the key features of a system, the term *benchmarking* needs to be introduced.

> A benchmark can be defined as *a set of conditions against which a product or system is measured.*
>
> [WHATIS].

In this thesis, performance benchmarking is introduced as a method of obtaining trusted measurable results. A benchmark conducted on two application servers, namely COM+ and EJB, will be exposed to the reader in section 3.4.

The industry standard benchmark for transactional throughput is specified by The Transaction Processing Performance Council (TPC) [TPC]. It is a non-profit organization which defines transaction processing, database benchmarks and delivers objective, verifiable, results to the industry.

TPC-tests fall into several categories which are briefly described in Table 2-2.

TPC-test	Status	Description
TPC-A	Obsolete as of 6/6-95	TPC-A measures performance in update-intensive database environments, typical in online transaction processing applications.
TPC-B	Obsolete as of 6/6-95	TPC-B measures throughput in terms of the number of transactions per second that a system can perform.
TPC-C	In use	TPC-C is an online transaction processing benchmark.
TPC-D	Obsolete as of 4/6-99	TPC-D represents a broad range of decision support (DS) applications that require complex and long running queries against large complex data structures.
TPC-H	In use	TPC-H is an ad-hoc, decision support benchmark.
TPC-R	In use	TPC-R is a business reporting and decision support benchmark.
TPC-W	In use	TPC-W is a transactional web e-Commerce benchmark.

Table 2-2: The TPC Benchmarks

In an application server context, it is natural to look at the TPC-C test. TPC-C simulates a complete computing environment where a population of users executes transactions against a database. The benchmark is centered on the principal activities (transactions) of an order-entry environment. These transactions include entering and delivering orders, recording payments, checking the status of orders, and monitoring the level of stock at the warehouses.

TPC-C involves a combination of five concurrent transactions of different types and complexity, which are characterized by:

- The simultaneous execution of multiple transaction types which span a breadth of complexity
- Online and deferred transaction execution modes
- Multiple online terminal sessions
- Moderate system and application execution time
- Significant disk input/output
- Transaction integrity (ACID properties)
- Non-uniform distribution of data access through primary and secondary keys
- Databases consisting of many tables with an extensive variety in sizes, attributes, and relationships
- Contention with data access and update

For the performance benchmark, this thesis uses a simplified TPC-C test; it is described in detail in section 3.1.3.

Chapter 3

APPROACH

The objective of this chapter, as described in the introduction, is to present an unbiased comparison of the currently available component models by directly comparing their qualities and properties.

The most significant source of information comes from various forms of research, where the study of literature prevails. The authors' project experience gained from commercial projects gives a legitimate starting point and a fairly comprehensive view of the big picture, and is therefore playing a central role in the understanding of the theory and the complexity of component technology. In order to organize the qualities and properties into rational categories, a well-known development methodology [DW99] has been utilized as the starting point.

At the end of this chapter, the authors will present the implementation and the complete configuration of the test environment.

3.1 Research Methods

A fair amount of research had to be conducted in order to comprehend the underlying concepts of component models. In order to achieve a most complete understanding of a component model, a thorough comprehension of the entire process, from design to implementation, test and deployment is necessary. In view of this, there is a strong focus on understanding all the steps of the process.

3.1.1 Literature study

The main methodology used for collecting information is the study of literature. The authors initiated their research by gathering information on both the Internet and at the library, then sorted out the relevant pieces of literature and finally studied them. The different categories of gathered literature sources are:

- *Published articles or similar work,* in the respect of research which is not currently available, only bits and pieces from other sources and not always unbiased written sources can be identified.
- *Functional specification*, a formal document used to describe, in detail, intended capabilities, appearance and interactions with users [WHATIS]. This is factual information on how the system should- or has been implemented to- function. It is considered as unbiased.

- Many *published books* are written by authors affiliated with companies that have vested interests in either technology. Even when it is not so, the author often has preferences and is somewhat biased.
- *A whitepaper* is an article that states an organization's position or philosophy about a social, political, or other subject, or a summary technical explanation of an architecture, framework, or product technology [WHATIS]. This is often biased information but presents important facts and knowledge about a product.
- *Articles on the Internet* comes in various formats: contents from homepages, Usenet discussions, debates in communities, etc. They are publicly accessible on the Internet. The nature of the Internet itself makes it difficult to evaluate the seriousness and validity of these sources. Nevertheless, these personal opinions and statements are of interest and should be taken into consideration.
- *Newsletters.* Several companies publish newsletters; this is a (e.g. monthly) subscription service open to anyone's participation, where the ones of interest to this thesis discuss different aspects of the technologies in question.
- *An important debate* is going on concerning the subject discussed in this study. In some occasions, transcripts are published. These debates will of course reflect the opinion of the participants, but will nonetheless provide useful input.

The rule of thumb is that nearly all information gathered from literature is biased, with the exception of serious research. Obviously, this must be taken into account when the direct comparison is performed, and literature is the basis.

3.1.2 Design

It was essential that the design covered all aspects of what would be implemented in the next stage. Therefore, the realized case is a real life example to fully illustrate the mechanisms in the component models. The authors found that a simple Internet record shop (see 3.4.3) contained the different types of components needed to make the comparison.

By choosing the Unified Modeling Language (UML) [BRJ99a] as the modeling language, the latest standard of modeling presently available has been used. Working closely with Genera AS [Genera AS] naturally led the authors to choose a model-close-development strategy, that is, an iterative process where as much code as possible is generated in the early stages of the development. Later, the developer can easily go back, change the model, and regenerate the necessary code.

3.1.3 Implementation and methodology

In order to measure a system's runtime qualities, an implementation is imperative. The companies promoting their respective component models give a highly positive pitch about "their" technology. The only way to put their claims to the test is to actually check their technology by implementing important parts of each component model.

There is a need for a consistent method in order to test the behavior of applications with accuracy. The methodology used here was borrowed from [GZ00], which originally tested web-applications. In their methodology, they use a tool called The Grinder. Two test clients, which can be described as simplified Grinder tools, were implemented; one for each technology.

The primary motive for not using existing and renowned tools such as LoadRunner from Mercury Interactive [LOADRUNNER] is cost-related. Self-made tools cannot replace commercial products in this field, but can be used as a start to test applications. Additionally, the thorough understanding gained from developing similar custom-made tools is highly valuable.

The tests are performed by so-called business transactions. A business transaction is a collection of methods that model or emulate expected behavior of the system. An instance of one of the business transactions used in one test implementation (see Figure 3-1) illustrates a collection of use cases; when executed in a certain order, they form a business transaction. According to the UML, a use case constitutes a set of functionality, represented by an oval. An actor, represented by a matchstick person, performs actions towards a system [BRJ99a]. In this example, the actor in the use case diagram can retrieve a list of orders, modify the order, and finally create a new order. To simulate a real-life course of events, a business transaction with the following flow of events is created:

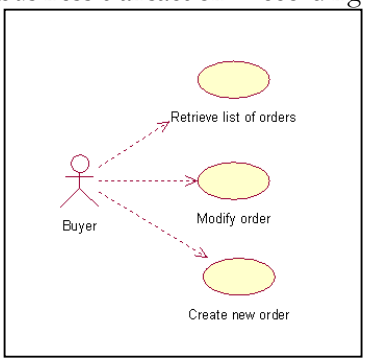

Figure 3-1: Sample business transaction

- `ShoppingCart.findAllOrders ();`
- `sleep (10); // Seconds`
- `ShoppingCart.setOrder (myOrder);`
- `sleep (12); // Seconds`
- `ShoppingCart.createOrder (myOrder);`
- `sleep (10); // Seconds`
- `ShoppingCart.findAllOrders ();`

First of all, the actor lists the available customers. The actor then views the results and modifies some data concerning a given customer (the whole process is estimated to take 10 seconds). Then, a new customer is added and finally, all the customers are listed out. This is the same business transaction as in section 3.4.4.3 referred to as the New Order business transaction.

3.1.4 Informal interviews

Throughout the research process necessary to this thesis, several informal interviews have been conducted. The authors have been in touch with BEA, Microsoft, and IBM on several occasions regarding questions about products, preferences, technicalities, etc.

The project leaders in the reference projects (see next paragraph) have also contributed in communicating information and has been taken into account in this thesis.

Taking part in professional projects, which involve the technologies in question, lead the authors into discussions about these technologies, their strengths, as well as their weaknesses.

3.2 Development project experience

Personal development experience gained from commercial development projects was very helpful with providing real life examples of how these technologies work in practice.

Both authors are part-time employees with Genera AS, and have several years of professional development experience. In the course of the past two years, there have been two projects of particular interest and they are used as reference projects within this thesis.

The authors' reference COM+-project is Rikstoto's [RIKSTOTO]

- A web system for betting on horses, the project was implemented using Visual Studio, COM+, and MS SQL Server 2000. The architecture of the system involves database replication, load balancing, high transaction volumes (up to 500 complete transactions every second) and very high security (as money is involved).

The authors' reference EJB 2.0-project is The Norwegian Railway's new ticket system, called NSB-LISA [NSB]

- This project uses Rational Rose, Genova, BEA WebLogic application server and an Oracle database. The project is due to be released 2nd quarter, 2002.

Diverse ideas and experience were collected from these projects, both from the design and implementation phases. First hand experience in developing

and designing distributed applications was of undeniable help when deciding on how to approach this implementation.

3.3 Development methodology

The Catalysis approach [GZ00] is the chosen platform of the authors' development methodology, where qualities are partitioned into runtime and development qualities; the traditional categories are functional and non-functional requirements.

The runtime qualities correspond to the task of measuring the application server runtime functionality. The development qualities, however, relate to the design structure and how it can be manipulated. In this particular case, these qualities must be measured up against the respective component model specification.

3.3.1 Runtime qualities

Runtime qualities are measured on a running system, and related to the dynamic behavior of the deployed system. The runtime qualities are:

- *Functionality* measures how well the system assists its users in performing tasks.
- *Usability* measures how intuitive the interface is for all kinds of users.
- *Performance* measures many different metrics concerning the speed of the system.
- *Security* measures the ability to prevent systems' unauthorized access or misuse.
- *Reliability* measures how the system performs over extended periods of time.
- *Availability* measures how the system handles failure.
- *Scalability* measures how easily the system can be scaled up to handle greater loads.
- *Upgradability* measures how easily the system can be upgraded.

3.3.2 Development qualities

Development qualities represent how easy it is to design and maintain the system. The development qualities are:

- *Modifiability* measures how easy it is to modify single components in the system while not interfering with the rest of the system.
- *Reusability* measures how easy it is to reuse components as well as the systems ability to integrate with legacy systems.
- *Portability* measures how easy it is to change runtime platform and/or vendor.
- *Buildability* measures how easy it is to implement and build the system.
- *Testability* measures how easy it is to test and debug the system.

- *Conceptual integrity* measures the system's elegance and practicality in a single quality. It is renamed to external qualities, and described in chapter 3.3.3.

3.3.3 External qualities

External qualities are allocated under the *Conceptual integrity* metric or fall outside of the Catalysis approach, which only views the system from a developer's/end user's perspective. The external qualities are meaningful to the person in charge of choosing technologies. Subsequently, these five metrics are included in this comparison. The external qualities are:

- *Time to market* measures the ability to deliver solutions fast, as the market evolves.
- *Cost of system* measures the actual price of the development platform.
- *Maturity* measures how long the technology has been on the market and how much it is being used.
- *Simplicity* measures how difficult it is for the application server users to gain an understanding, and to be able to develop applications for the server.
- *Future plans* are an overview of the roadmap for the technologies.

3.4 Implementation

For a proper measurement of the qualities listed in section 3.3, an implementation in both EJB and COM+ was needed. As far as the implementation is concerned, many things had to be figured out: the implementation, the hardware, and the software that should be employed. This called for a decision in order to make the benchmark as up-to-date and as neutral and fair as it possibly could.

3.4.1 Hardware

The hardware available for this benchmarking was limited; hence the setup of the hardware was not optimized to run big complex applications with a high transaction volume. The configuration featured only one physical server running both the database server and the application server. In a commercial solution, it would most likely be constituted of (at least) two physical servers. However, the hardware available was sufficient to perform these tests adequately, considering that the database server did not have a heavy load, since most of the load was on the application server.

3.4.2 Software

Choosing the appropriate software to use for the benchmark and analyzing the properties and qualities of these products represented important necessary steps, as they would directly affect the comparison. In the following

subsections, the choice of the operating system, programming language, application server and database is explained.

3.4.2.1 Operating System

The choice of platform, in order to perform neutral tests in which both application servers had exactly the same environment came quite naturally. As COM+ only runs on Windows 2000 and Windows XP, and Windows XP was still in beta at the time when the tests were commenced, there was no other choice than running all tests on the Microsoft Windows 2000 platform. While EJB is best known for running on powerful UNIX servers, BEA claims to have high performance on Windows 2000 as well. Having both application servers running on the same configuration for both solutions ensured a neutral "battleground". The operating system (OS) used was Microsoft Windows 2000 Server Service Pack 2.

As for the three client PCs, they all ran Windows 2000 Professional Service Pack 2.

3.4.2.2 Programming languages

The programming language for COM+ was initially planned to be Visual Basic (VB). Owing to a minimal amount of business logic, the choice of programming language was not of prime importance. Visual Basic became the most natural choice, because it is one of the most commonly used programming language on the Microsoft platform and has a high learning curve, in addition to being the choice of the COM+ reference project (see section 3.2). However, components written in Visual Basic unexpectedly do not support object pooling, due to the simple VB threading model. In the latest VB release, Visual Basic.NET, object pooling is supported. In order to achieve object pooling for one of the stateless components in COM+, the component had to be implemented in C++. As Microsoft Visual C++ is part of the Microsoft Visual Studio development suite which also contains Visual Basic, the authors chose it as their C++ compiler. Using C++ also demonstrates heterogeneity of programming languages on the COM+ platform. Both Microsoft Visual C++ and Microsoft Visual Basic were used with Microsoft Visual Studio version 6 Service Pack 5.

The EJB server does not give much option but to use Java. Since the application server was shipped with Sun JDK 1.3.1, it was the chosen version of the Java Virtual Machine throughout the implementation.

3.4.2.3 Application Servers

The choice of application server for COM+ is obvious as there was only one implementation available at the time the tests were commenced, that is, Microsoft Windows 2000.

The choice of application server(s) for EJB is not as obvious, as there are so many implementations to choose from. It soon became fairly manifest that the only two contenders were IBM WebSphere and BEA WebLogic. These two market leaders were the most complete and up-to-date implementations of the EJB specification, and had been on the market for a long time. At first, it was considered to test at least two application servers, since the EJB specification should make the implementation portable. However, the decision to test only one, in this case BEA WebLogic, was retained because it was the only implementation that supported the latest EJB 2.0 specification [EJB2.0]. It is incidental that one of the authors had former professional experience with BEA WebLogic as well.

3.4.2.4 The databases

The choice of the database server was not an easy task as both application server vendors had their preference. In addition to that, both application servers support all Open DataBase Connectivity (ODBC) for COM+/Java DataBase Connectivity (JDBC) for EJB compliant database servers, so the choices for database server were numerous. ODBC and JDBC are APIs for accessing a database. However, some of these database servers do not support all the transaction mechanisms and/or the locking mechanisms that EJB and COM+ utilize, and therefore they were dismissed in order to have access to all the mechanisms available in EJB and COM+.

BEA preferred the use of Oracle for optimal results, while Microsoft naturally recommended the use of MS SQL Server for best results. The objective of this thesis is not to measure the speed of any database server in any way, and thus the benchmark was made with very limited use of the database. Since Microsoft preferred SQL Server and BEA preferred Oracle, and these two servers being the marked leaders, it seemed natural to include both of them in this benchmark, so as to make the benchmark as neutral as possible. Unfortunately, there was a problem obtaining a version of Oracle, with the resources available to the authors. MS SQL Server supporting all the necessary standards that EJB and COM+ need in order to be fully functional, the choice was once again straightforward.

The database server was used with the default configuration, that is, the configuration of the server was not changed after the installation. While one can configure and tweak a database server for optimal performance, such tweaking was not the goal of this thesis and therefore ruled out.

The version used was SQL Server 2000.

3.4.3 Model

The methodology chosen for conducting the benchmarks called for a real life application to be implemented (see section 3.1.3 for implementation details).

Creating an ecommerce application seemed like a reasonable approach, and the test ended up being a simple Internet record shop.

The first thing that needed to be done was a UML class diagram of the application, in order to plan how to implement it, in the most suitable fashion for the benchmark purposes. It was necessary to have all three classes of components in the system: stateless, stateful and persistent, to see how all these features make the application servers behave under normal and loaded conditions.

3.4.3.1 Class modules

This simple model contains four classes: Artist, MyRecord, MyOrder and Customer.

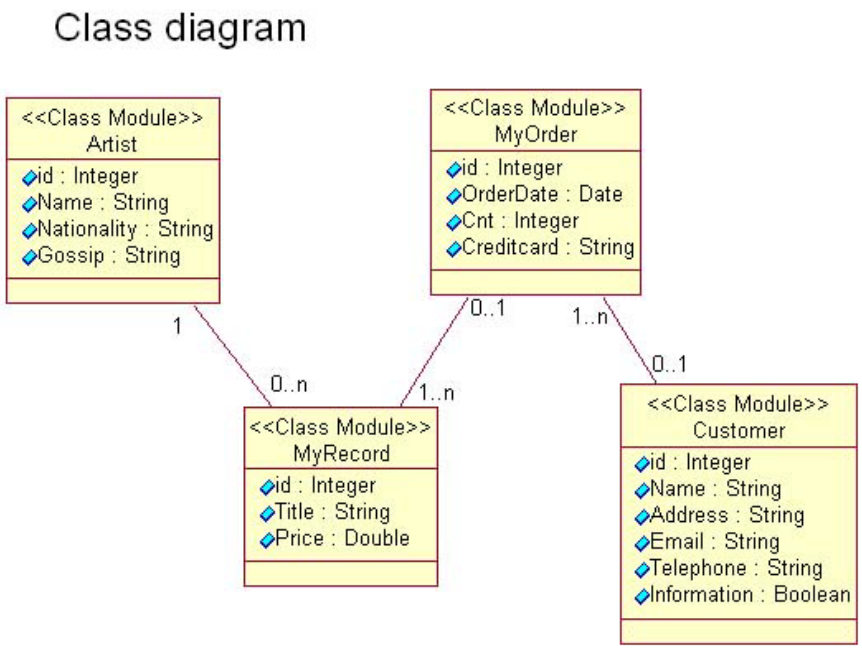

Figure 3-2: UML class diagram of the Record Shop

Figure 3-2 illustrates the four module classes present in the database as tables as well as their respective attributes (Boolean is represented by a short). The classes have been designed as small simple classes in order to keep the database traffic to a minimum.

3.4.3.2 Components

Figure 3-3 shows the component diagram of the record shop. The MyOrder, ShoppingCart and RecordServices components each expose a business interface to the client tier. The record shop client is only communicating directly with the three interfaces. The four business cases are represented with their main correspondent business logic components.

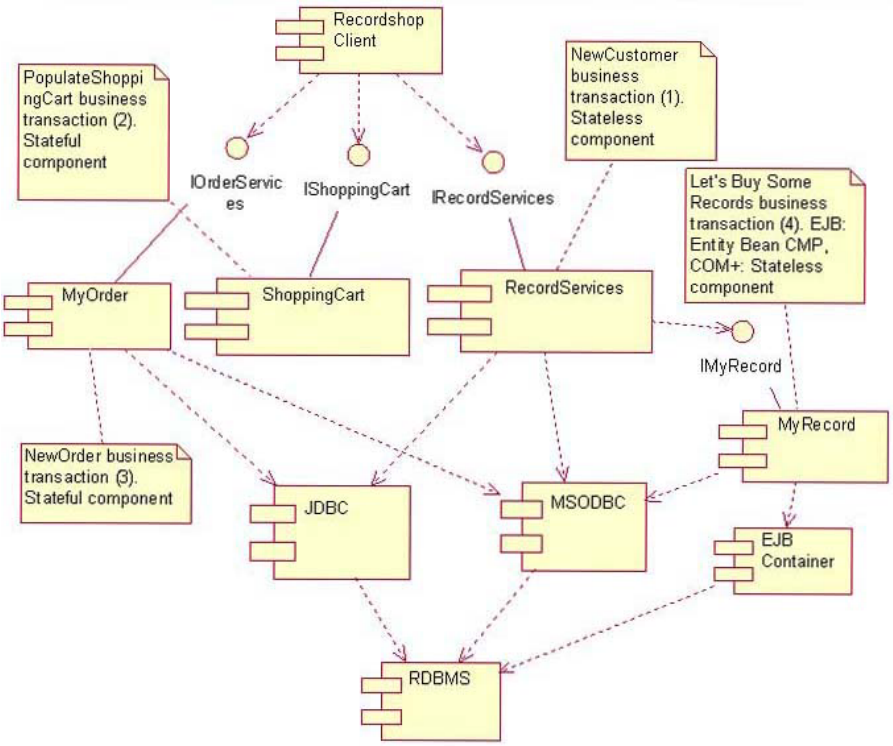

Figure 3-3: Component diagram of Record Shop

In the New Customer business transaction, the stateless RecordServices component uses an API (JDBC or MS ODBC) to communicate with the Relational DataBase Management System (RDBMS). A RDBMS is a program that allows creating, updating, and administering a relational database. The RecordServices component is implemented using C++ in order to achieve object pooling in COM+ as discussed above. The RecordServices component handles the business and the database logic in this business case.

In the Populate Shopping Cart business transaction, the RDMS is not utilized and state is stored internally in the object instance.

In the MyOrder business transaction, a stateful component uses the same API's for RDBMS connections as in the first business transaction. The main difference from the New Customer case lies in that, this time, the component handling the business logic is stateful.

In the case of MyRecord, a persistent component, the RecordServices component is placed in front of the persistent components and makes the client tier independent of the persistent implementation. The interface IMyRecord is also represented in the diagram. With EJB, it represents the remote interface of the MyRecord entity bean. The entity bean is container managed persisted, meaning that the database logic is handled by the EJB container, as illustrated by the diagram. In the EJB world, wrapping entity beans with session beans such as this represents a structural design pattern known as the facade pattern [GHJV95] or distributed facade pattern [BEP99]. With COM+, this type of component is simulated by implementing a stateless component with programmatical database logic.

The class modules of Figure 3-2 also function as value objects in the implementation. Value objects encapsulate information needed by the presentation logic. For example, the MyOrder value object, as indicated in the code below, encapsulates the information from a persistent MyOrder-component and ensures a clean separation between the persistent tier and the presentation tier. In this way, the persistent object is never returned to the client tier; this pattern is known as the Replicate Object [BEP99] or Data Transfer Object [FOW01] pattern. When a `findOrders()` function call is issued from the client, the stateless component (RecordServices, see Figure 3-4) creates the collection of MyOrder value objects and returns them to the client.

```
//Source file: o:\\Hovedfag\\src\\no\\henrik\\domain\\Myorder.java
package no.henrik.domain;
import java.io.Serializable;

public class MyOrder implements Serializable{
         private Integer Id;
         private String OrderDate;
         private Integer Cnt;
         private String Creditcard;
         private java.util.List theRecord;

         public getOrderDate() {
           return this.OrderDate;
         }
         .
         .
         .
}
```

Figure 3-4: MyOrder.java class

This can only be seen as a snapshot of the actual values in the persistent storage; however, it is a commonly used method for distributing information.

3.4.4 Clients

The clients are based on The Grinder, a test client application tool for web applications [GZ00]. By implementing these test clients, they became as identical as technically possible; a great deal of effort was necessary in order to obtain a common foundation for the two client implementations. Naturally, with different programming languages and several differences in application server architecture, the clients differ programmatically, but not in functionality. To investigate qualities, and to load - and stress test the application servers - simulation of a set of actions, which a user would normally perform on the client, was conducted. One sequence of actions is called a business transaction. By running many instances of the client on several machines in a network, a simulation of the average every-day usage of a complete system is archived.

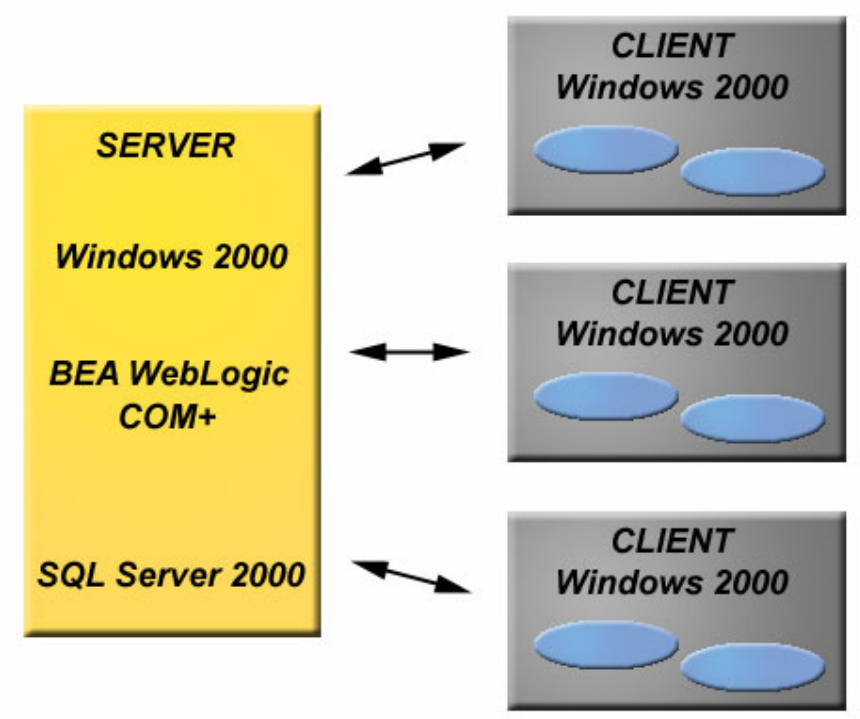

Figure 3-5: Test environment

As illustrated by Figure 3-5, the clients (the gray boxes) were distributed on the network, running several instances (the blue boxes) and iterations on different machines.

In order to simulate real EJB clients, multiple JVMs (or instances) must be started, with only one thread each. When several Weblogic clients run in one

JVM, they use one socket to communicate with the server because it optimizes performance [GZ00]. That is why the clients, on both technologies, run only one thread on each instance.

The first time that a component is activated from a client, a lookup/activation process is initiated. With COM+, the application is started (if not running) and, in this case, the COM+ stub is already on the client. The EJB client uses JNDI to return a RMI-stub to the client; this step is necessary owing to portability reasons. Transferring the stub could take a considerable amount of time, and is not included in the measurements of the business transaction response time. However, the creation time of the component is included.

3.4.4.1 Client options

To easily simulate various loads on the application servers, the clients support several tuning options. The values actually used in the tests can be found under section 3.4.6. The tuning options are:

- *No. of instances*. This number indicates how many instances of the program run on one machine. In the Java world, it corresponds to the number of JVM's to be started. Windows 2000 starts the program in the background with normal priority.
- *No. of iterations*. This is the number of times that each business transaction is repeated. Reiterating the business transaction several times allows making an analysis of the server load over an extended amount of time. Application server optimizing and caching need a few iterations to improve the performance of the code. Stress tests usually have 3 to 10 iterations of the business transaction [GZ00].
- *Initial sleep*. This number states, in milliseconds, the maximum wait there can be before the business transaction starts. The instance of a program waits for a random number of milliseconds, comprised between 0 and this determined value, before it starts the business transaction. This option is necessary because a great number of simultaneous connections to the application server will not be a realistic situation.
- *Start time*. The clients are able to start running at a given time (hh:mm:ss). This optional parameter can be left blank if the clients are to start immediately. In this way, clients on different machines can be synchronized to start simultaneously.
- *Log file wait*. In order to prevent unnecessary CPU-usage on the client machine, the log file is not produced until a given period of time has elapsed since the completion of the client process.

3.4.4.2 Recording data on the client

The client is not only expected to perform the business transaction, but to also report reasonable data for interpretation. To retrieve a summary of the

data, a script was written in order to collect and calculate the average and peak numbers needed for the interpretation. This script simply takes all log files available in a directory and gathers them so as to present sensible numbers that could be worked with.

Applying the test methodology described in [GZ00], was collected the following data, important for the further interpretation of the results:

- *HostId, jvmId and IterationId.* Together, these variables present a unique reference clarifying from which host, instance, and iteration the measurement originated.
- *Total Successful Transactions (TST).* This number is a counter incremented by one, every time a successful business transaction is completed and no errors occurred during the entire business transaction.
- *Total Processing Time (milliseconds) (TPT).* This is the total of milliseconds accumulated for the entire business transaction, waiting time not included.
- *Average Response Time (milliseconds) (ART).* This is the average response time of the individual method call. It represents the total average of all iterations in all instances on all client machines.
- *Transactions Per Second (TPS). The* number of transactions is logically calculated as 1/ART.
- *Total Unsuccessful Transactions (TUT).* This number is a counter incremented by one, every time any method in a business transaction fails. The remaining methods will not be executed.

A sample output from the client log file is presented in Appendix 2.

In addition to this, CPU usage on the client machine was recorded by using Windows 2000 performance monitor. This recording was done in order to ensure that the client machines were capable of handling the amount of clients required by the test. Had the client CPU been overloaded, the test results might have been considered as void since the delays from the client could have affected the test results.

3.4.4.3 Client business transactions

At least one business transaction for each type of component is defined: stateless component, stateful component, and persistent component.
The different business transactions are typical for the component in question. A more detailed view of the business transactions follows.

No.	Business transaction name	Component type
1	**New customer**	**Stateless**
2	**Populate shopping cart**	**Stateful**
3	**New Order**	**Stateful with persistent storage**
4	**"Let's buy some records"**	**Persistent**

Table 3-1: Overview of business transactions

1. **New customer business transaction.**
   A stateless component is relatively short-lived and typically provides a single-use service, independent of which client is calling the service, e.g. adding a customer to the record shop. Also, stateless components often function as a layer between the client and one or more persistent components [GWE01], which will be tested in business transaction no. 4 and 5.

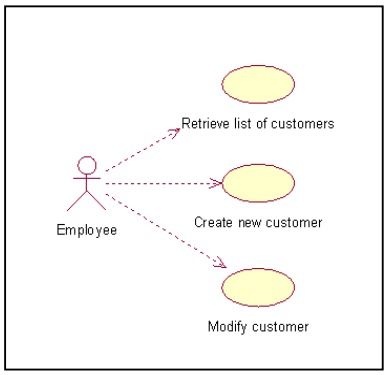

Figure 3-6: New Customer business transaction

   In this business transaction, the client simulates a typical user's course of events. First, it lists the available customers, secondly, it views the results and then, it modifies some data on a customer (the entire process is estimated to take 10 seconds). Lastly, the client adds a new customer and retrieves all customers.

   The stateless component developer will manage the persistence programmatically.

```
1. RecordServices.findAllCustomers ();
2. sleep (5); // Seconds
3. RecordServices.setCustomer (customer);
4. sleep (5); // Seconds
5. RecordServices.createCustomer (customer);
6. sleep (5); // Seconds
7. RecordServices.findAllCustomers ();
```

2. Populate shopping cart business transaction

A stateful component is session-oriented, meaning that it maintains state across methods calls and transactions. This state is kept in the applications server's memory, and no database operations are necessary. A typical example of a stateful component is the shopping cart, which life depends on the life of the client.

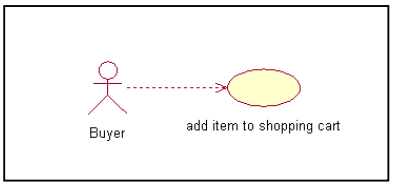

Figure 3-7: Populate shopping cart business transaction

In the stateful business transaction, the way the shopping cart is handled by a real life user is simulated by adding records to the shopping cart at regular intervals. No calls to the persistent storage are made, but all state is kept in the memory of the application server.

- `ShoppingCart.addItem (myRecord);`
- `sleep (5); // seconds`
- `ShoppingCart.addItem (myRecord);`
- `sleep (5); // seconds`
- `ShoppingCart.addItem (myRecord);`
- `sleep (5); // seconds`
- `ShoppingCart.listItems ();`

3. New Order business transaction

The second business transaction of the stateful component contained persistent storage to the database in order to compare the performance with the stateless component. It is a similar flow of events as in the aforementioned business transactions.

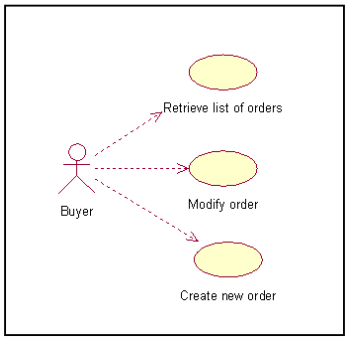

Figure 3-8: New order business transaction

- `ShoppingCart.findAllOrders ();`
- `sleep (5); // Seconds`
- `ShoppingCart.setOrder (myOrder);`
- `sleep (5); // Seconds`
- `ShoppingCart.createOrder (myOrder);`
- `sleep (5); // Seconds`
- `ShoppingCart.findAllOrders ();`

4. **"Let's buy some records" business transaction**

Persistent components are a representation or a view of the data from a data store (typically a relational database). In EJB, they are known as Entity Beans. Because they represent a data store, persistent components are transactional, and their transaction setting has much to do with the achieved performance, as mentioned in [GZ00] and as the further test results will indicate.

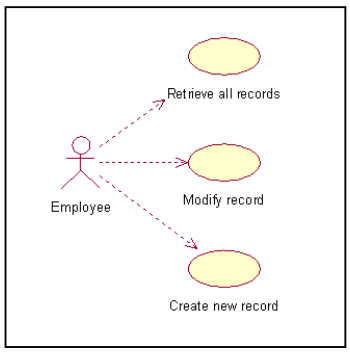

Figure 3-9: Let's buy some records business transaction

This business transaction is very similar to the business transaction defined by the stateless component. The difference lies in that another table (Record instead of Customer) was used. Indeed, this time, a stateless component is utilized as a layer between the persistent component and the client.

```
8. RecordServices.findAllRecords ();
9. sleep (5); // Seconds
10.RecordServices.setRecord (myRecord);
11.sleep (5); // Seconds
12.RecordServices.createRecord (myRecord);
13.sleep (5); // Seconds
14.RecordServices.findAllRecords ();
```

3.4.5 Application servers

There is a fundamental difference between the two application server implementations: WebLogic has emerged from a specification while Windows 2000 is a proprietary implementation of COM+.

BEA has added several features in WebLogic 6.1 not available in the EJB 2.0 specification. Time to market being a key issue, BEA does not necessarily have the time to wait for the EJB 2.0 specification to be completed before presenting their products to the market. Several of the features that BEA has added to their WebLogic 6.1 implementation, such as extensions to the QL-language and Read-Only Entity Beans [BEA 01b], are proposed for the next version of the EJB Specification. This was an interesting dilemma. Was the BEA server or the EJB Specification being tested? All special features of the BEA product should be taken into account when looking at the performance. When comparing the qualities and properties on a less technical level, it had

to be the specification that counted, not the specific implementation. After all, certain aspects, such as vendor neutrality, would then be void.

3.4.5.1 Server tuning

It is an extremely difficult, if not impossible, task to place both application servers side by side in every single technical setting. Where the settings on a very detailed level can be tuned, the default setting of the application server is maintained and is not (also for delimitation purposes) mentioned here. It is important to remember that the test implementations are as identical as possible on both technologies, and not necessarily optimal.

Options referenced in the official tuning and the performance papers from BEA [BEA 01a] and Microsoft [PLATT00] are considered substantial and are taken into account when adjusting the parameters and the setting of the application servers.

Table 3-2 presents a listing of important tuning parameters with their initial setting at the start of the test period. Some of the parameters were changed during the tests, as further described in section 3.4.6.

No	Setting	EJB	COM+
	Application server	BEA WebLogic Server	MS Windows 2000
	Version	6.1 Service Pack 1	Service Pack 2
1	Runtime environment	JDK1.3.1	Windows 2000
2	Heap size of JVM	128MB	N/A
3	Transaction handling	Container managed, required	Required
5	Transaction isolation level	SERIALIZABLE	SERIALIZABLE
6	Database Connection pool size min/max	25/100	Handled by ODBC driver
7	Transaction timeout	60 sec.	60 sec.
8	Initial and maximum component pool	0/limited by memory	0/limited by memory
9	Creation timeout	60 sec.	60 sec.
10	Just in time activation	Yes	Yes
11	Activation type	Call by reference where possible	Library application where possible
12	No. of execute threads	100	Handled internally
13	Database driver	Type 4 (MSSQL) driver	ODBC and ADO 2.6
14	Security	None	None

Table 3-2: Tuning parameters

1.
The default JVM for Weblogic 6.1 is used: JDK 1.3.1 [SUN]. According to [GZ00], this generation of JVM with hotspot optimizer performs remarkably better than other JVM's. Being also Weblogic default, JVM was the natural choice. As mentioned, Windows 2000 is the only runtime environment for COM+.

2.
The JVM heap size determines how often and how long the VM collects garbage (de-allocating unused Java objects from memory). When a JVM runs out of memory in the heap, all executions in the JVM stop, while a garbage collection algorithm frees space that is no longer required by an application. This process affects performance because server-side work cannot proceed during garbage collection. If a large heap size is set, full garbage collection is slower but occurs less frequently. If heap size is set in accordance to the memory needs, full garbage collection is faster, but occurs more frequently. The goal sought in tuning heap size is to minimize the time spent doing garbage collection, while maximizing the number of clients that can be handled by the server at any given time. This value is raised to 128 from the default of 64MB, in a case of trial and error.

3.
Transaction handling. The setting REQUIRED specifies that all objects created by the component will be transactional. It is the preferred setting for an object that performs resource activities, because it guarantees transaction protection for these activities [GWE01]. The EJB specification allows a session bean to choose between either container-managed or bean-managed transactions. In container-managed transactions, the transactions automatically start and commit as requested.

5.
Transaction isolation level. The EJB 2.0 specification supports explicit setting of the transaction isolation level (how the database handles the issued concurrency) for any or all transactions. In COM+, the transaction isolation level must be set manually or in the database. The SERIALIZABLE isolation level was chosen, because it is the only supported isolation level for the MTS in COM+.

6.
Database connection pool. The application server opens connections and puts them in a connection pool accessible to all clients. When a client closes a connection from a connection pool, the connection is returned to the pool and becomes available for other clients; the connection itself is not closed. The best performance occurs when the connection pool has as many connections as there are concurrent users [BEA 01a]. The COM+ connection pool is handled by the ODBC driver.

7.
Transaction timeout. It sets the default timeout for the transactions initiated in this component. If the duration of a transaction is longer than this default value, it will be rolled back. 60 seconds is the default setting.

8.
Initial and maximum component pool size. The nature of stateless components allows applications servers to maintain a pool of components for every stateless component class. There is, per default, no upper limit except for the available memory. Setting the initial value to a number different from the default (zero) populates the component pool at startup, and improves the initial response time of the application server.

9.
Creation timeout. It represents the maximum time that the creation of a component can remain active before it times out. Creation processes that remain active beyond this period of time are automatically aborted by the system. 60 seconds is the default setting.

10.
Just In Time (JIT) activation. The purpose of JIT activation is to save resource. It achieves this by ensuring that a component lives exactly as long as needed. When JIT activation is activated for a component, the instance is not created before a call is made to the component, and the component is terminated immediately after the call is done.

11.
Pass by value is always necessary when the component is called remotely (not from within the server). Components that are called from within the server should be library applications (COM+ terminology) or call-by-reference-components (EJB-terminology). Passing by reference increases the performance of the method invocation since the parameters are not copied.

12.
No. of execute threads. This value equals the number of simultaneous operations performed by the server. As a job enters the application server, it is placed in the execute queue. This job is then assigned to a thread that does the work on it. Threads consume resources, so a value too high could degrade the performance. The value is set to 100 for the WebLogic server.

13.
Database driver. WebLogic supports several types of JDBC drivers [GWE01]. The JDBC driver for Microsoft SQL server is a type 4 driver, for lack of available type 2 drivers. This is a 100% Java implementation of the JDBC API. It provides direct access to MS SQL Server, and requires no vendor-supported client libraries.

COM+ uses Windows 2000 Microsoft SQL server driver.

14.
Security. The security is set to none for all components. While being an important part of a distributed system, it is cheaper to handle some of the security aspects with hardware [PLATT00].

3.4.5.2 Recording test data on the server

Several measurements are recorded on the application server or in the server operating system.

During a test run, Windows 2000 performance monitor logs the server CPU usage, thread and processes count, number of connections to the database and the network usage. The metrics are sampled every second and written to a log file on the server. The network usage is measured in order to detect if the network is a bottleneck for the system.

3.4.6 Conducting the tests

The tests are conducted with the test clients running on three client machines (as shown in Figure 3-5). The COM+ test and the EJB test are naturally not run simultaneous. The performance tests are conducted in a clean environment; when the tests are not running, the network load is none. All tests are run with a synchronized start time. To synchronize the starting time on all computers, the Network Time Protocol (NTP) [NTP] is used. NTP synchronizes the clocks of hosts and the routers in the Internet.

In order to stress test a system, the stress tester starts by analyzing individual response, verifying that the response time falls within an acceptable range, and that the application actually functions as desired. This is called a *functional test* [GWE01]. After a successful testing of the application functionality, the baseline case [GZ00] test is conducted: A small discrete number of simultaneous clients are executed in order to understand how the application behaves when the server is not stressed. The baseline case is typically defined with about 50 users in an application such as this one.

The increase in the load is stepped in such a way that a meaningful performance chart can be drawn. At first, it is a case of trial and error but eventually permits to identify the limits of the application.

The test client options available have previously been presented. The mutual settings for the test runs are indicated as follow in Table 3-4.

No. of client machines	No. of iterations in each instance	Initial sleep (milliseconds)	Log file wait (seconds)
3	5	5000	120

Table 3-3: Mutual test parameters for all tests

12 tests per application server were conducted, which is three per business transaction as shown in Table 3-4.

Business transaction	Nr. of instances on each machine (test1/test2/test3)
New Customer	20/50/100(50)
Populate Shopping Cart	20/50/100(50)
New Order	20/50/100(50)
Let's buy some records	20/50/100(50)

Table 3-4: Test cases

As indicated in Table 3-4, the tests are run with 60 (20 * 3 client machines), 150 (50 * 3 client machines) and 250 (100 * 2 client machines and 50 on the last client machine) simulated clients. As presented in the following chapter, the upper limit for both applications server is reached at 250 simultaneous clients. Three test runs per business transaction are sufficient to observe the application behavior in typical, loaded and stressed (atypical) conditions [GZ00].

Chapter 4

IMPLEMENTATION

This chapter presents the allocation of component model services into runtime, development and external qualities [DW99]
. The key aspects of a component-based application are listed and organized for a thorough comparison in the next chapter. Also presented are the results from the conducted tests.

See Appendix 3 for a summary and high level comparison of COM+ and EJB properties.

4.1 Determining properties

In order to decide what was a significant property or quality, a close attention was paid to what the component models had to offer, or more specifically what the implementations had to offer. The selected aspects come from various definitions of component models (see chapter 2) and from other work conducted on the subject.

Several articles have been written about component models head-to-head comparison, with a strong focus on their entirety. Anne Thomas [THOMAS98] looks at basic family values such as language support, platform support, protocol support, etc. Two of the most prominent persons in the EJB vs. COM+ debate are Ed Roman and Roger Sessions. Each of them wrote a book on this subject: [SESSIONS00] and [RO99]. A transcript from a debate [RS99] between these two personalities has been an important source of research. The Serverside [SERVERSIDE] is an Internet community discussing component technology with a focus on J2EE. The Middleware Company wrote a whitepaper [RO99] presenting what they call "Technical Benefits of EJB and J2EE Technologies over COM+ and Windows DNA". Microsoft also published an article about the benefits of MTS vs. Enterprise Java Beans

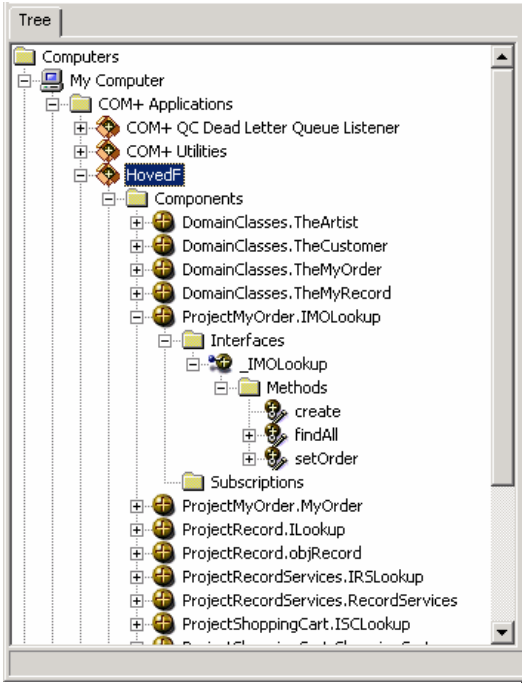

Figure 4-1: MMC treeview

[MICROSOFT98]. In addition to that, Roger Sessions published a more recent paper that compares the technical aspects of .NET vs. J2EE [SESSIONS01]

In COM+, the configuration of the components and the transaction statistics are presented to the user with a graphical user interface. This tool is available from the Microsoft Management Console (MMC), which is included in all Windows 2000 versions (see Figure 4-1). To locate functionality and services provided by COM+, a thorough examination of the functionality available in the MMC has been carried out.

EJB available functionality and services provided have to be taken from the current available specification [EJB2.0].

4.2 Runtime qualities

In this study, the runtime qualities correspond to the task of measuring the application server runtime functionality. In this case, they must be measured up against the respective component model specification. Diversity in behavior of different application server implementations can be expected with EJB. In this case, the WebLogic implementation is therefore the only one to be taken into account.

4.2.1 Functionality

Functionality measures how well the system assists its users in performing tasks. The main user in an application server context is the developer, but some tasks can be distributed to other actors in a project. The developer performs many tasks, such as tuning the server, compiling against it, deploying components, etc. A list of such tasks is compiled and listed in this section.

4.2.1.1 Event management

According to the definition of a component model from SUN, a component model should contain event management, or asynchronous event-driven communication.

An example usage scenario would be an event-driven process that operates asynchronously. An intranet workflow application fits the profile wherein business objects (examples are leave requests, travel reimbursement requests, etc.) send asynchronous messages. After the requests have been processed (it may even take a day or two), the application can further invoke another event to inform the user via email, mobile phone, or pager that the request has been processed.

Both technologies have their own solution to asynchronous event-driven communication.

Event management in EJB
Event management is supported in EJB by an implementation of message-driven beans. This type of enterprise bean is asynchronously invoked to handle the processing of incoming Java Messaging Service (JMS) messages. JMS [JMS] is a standard vendor - a neutral API that is part of the J2EE platform and can be used to access enterprise messaging systems. A typical message-driven object is stateless, can be transaction aware and executes upon receipt of a single client message. JMS is a prerequisite in the EJB specification.

The message driven bean was not implemented in the test implementation because the implementation was then unavailable. At the time of the writing, WebLogic 6.1 (among others) fully supports message driven beans.

Event management in COM+
COM+ handles messages through the Microsoft Message Queue (MSMQ). On top of the MSMQ, there is an infrastructure called Queued Components (QC). It abstracts the details of what is happening behind the curtain in MSMQ, away from the developer. As a result, the developer feels as though programming regular COM+.

The queue listener can be disabled / enabled as appropriate on the application level.

No tests of MSMQ and QC were conducted in the implementation. Indeed, they would have had no relevance since the corresponding feature was not available in the EJB implementation.

4.2.1.2 Component packaging

According to the definition by SUN of a component model, a component model should offer the possibility of packing files belonging to a component (such as icons, or graphics files) to a distributable format.

Component packaging in EJB
The EJB specification states that the EJB-jar file should be the standard format for the packaging of Enterprise Beans. It contains one or more enterprise beans, plus application assembly information describing how the enterprise beans are combined into a single application deployment unit. The EJB-jar file must also contain, either by inclusion or by reference, the class files for all the classes and interfaces, which upon each enterprise bean class as well as the home and component interfaces depend, with the exception of the system classes. Client stubs should not be included in the EJB-jar file, and are typically generated at runtime or deployment time.

The details about the packaging of components for deployment are specified in the J2EE specification [J2EE] and are too extensive to be presented here.

Component packaging is fully supported by WebLogic, and the test implementation uses a single EJB-jar file for the entire application.

Component packaging in COM+
In COM+, the standard for packaging is the Dynamically Linked Library (DLL) file. This DLL file can contain any number of components, as well as a lot more than just the implementations of the components. It includes the type library, configuration information, and class factories. Client stubs are generated from the MMC and exported to the client computers. These client proxy stubs can be automatically exported to a single server through the Application proxy, which specifies a remote server where to export the stubs.

The test implementation consists of 5 DLL files.

4.2.1.3 Instance and life cycle management

A component model runtime typically manages creation, management, and destruction of components; there are similarities in existing component models.

Instance management is about giving the client the impression that a dedicated component is waiting to service its request. The Container enters its instance management algorithm when a call is made from a client. In COM+, this is called Object Pooling and in EJB, Instance Pooling. For COM+, it is quite common to use JIT activation with Object Pooling, so that a component instance is placed back into the pool immediately after the execution of a function.

EJB instance and life cycle management
The EJB specification presents two main types of objects: session objects and entity objects.

According to the EJB specification, a typical session object is relatively short-lived and *executes on behalf of a single client. It does not represent directly shared data in the database, although it may access and update such data. The object is removed when the EJB Container crashes, if so the client has to re-establish a new session object to continue computation.*

The specification also states that a typical EJB Container provides a scalable runtime environment to execute concurrently a large number of session objects. Session beans are intended to be stateful. The EJB specification also defines a Stateless Session Bean as a special case of a Session Bean.

All session objects of the same Stateless Session Bean within the same home have the same object identity, while a stateful session object has a unique identity that is assigned by the container at creation time. The EJB specification recommends stateless beans regarding scalability, as mentioned in section 4.2.6.

The second object type in EJB, the entity object, provides an object view of the data in the database. It allows shared access from multiple users and can be long-lived (it lives as long as the data in the database). The entity, its primary key, and its remote reference can survive the crash of the EJB Container.

In EJB, an object instance of a component may be pooled; this is called instance pooling, as described in section 2.6.7.

WebLogic supports both object types and follows the specification. To prevent long-lived stateful components from monopolizing too many resources, the EJB container passivates idle components by temporarily persisting them to the disk. The components are reactivated when necessary. In the implementation, the business cases test the various components. The first business case, New Customer, tests the stateless bean. Populate Shopping Cart and New Order business transactions tests the stateful bean. Finally, Buy Some Records tests the Entity Bean.

COM+ Instance and life cycle management

COM+ has one type of object. Statelessness or statefulness is not an issue– it is left to the sole discretion of the developer as to decide which is the most reasonable way to implement an object. Against popular belief, it is possible to implement stateful components in COM+.

COM+ has several ways of influencing the life cycle of an object. A developer can:

- Enable JIT activation. It will make a component deactivated as soon as it completes its task(s), and will be activated when needed. This process is transparent to the programmer (but still needs to be taken into account as it means that the next object instance the client acquires, will most likely not be the same).
- Control the life cycle manually by creating and releasing the component programmatically. If JIT activation is disabled, the stateful components will become available.
- Ensure that a minimum number of components are ready at all times, and sets a limit of how many of them can be active at one given time.

A COM+ component can be stateful in various ways [PLATT99]. The state can be stored in the client, in the component itself, in a resource dispenser, or in a resource manager. A resource manager can be any type of storage, from a flat file to a database. All of these four approaches to state management present pros and cons.

Storing the state with the client is convenient. Indeed, the component is not concerned at all with the state, although this approach calls for advanced clients.

Storing the state in the component means fast access to the state, but this solution does not scale as the component will be tied to one client, for as long as the client deems it necessary – thus the component can not be used by other clients while it is inactive waiting for the first client to complete execution.

Having the state in a resource dispenser, such as the shared property manager (SPM), is a mediocre solution, but it scales better as the component is not tied to the client. The SPM is a resource dispenser that can be used to share state among multiple objects in a server process

A forth place where to keep the state is in a resource manager. It means that the state is persistent, but the persistence comes at a price. Access to the resource manager is very slow.

Of course, the state can also be stored in other numerous ways, such as text files, excel spreadsheets, etc. All these methods are too costly, both speed-wise and implementation-wise, and do not represent serious options for a business system.

In a beta version of COM+, Microsoft had implemented support for this kind of component, but made a strategic decision by removing all state from the middle tier.

In the implementation, different approaches are tested. The New Customer business case is implemented as a stateless component. The Populate Shopping Cart and the New Order business case are both implemented as stateful components, keeping the state within the component itself. The last business case, Buy Some Records, is implemented as a component with persistent state.

4.2.1.4 Query language

Every component model needs a language to communicate with its persistent storage. An example of usage can be found in Appendix 4.

Query language in EJB
In container-managed persistence, unlike in bean-managed persistence, the developer does not write database access calls in the methods of the Entity Bean class. EJB QL is a query specification language for the finder and selects methods of Entity Beans with container-managed persistence. EJB QL can be compiled to a target language, such as SQL, to a database or to any other persistent store. EJB QL is a subset of SQL, but substantially less mature.

Entity Beans, with bean-managed persistence, enable the developer to write database access calls with JDBC (SQL).

WebLogic fully supports QL and is used for the Entity Bean in the test implementation.

Query language in COM+
Microsoft provides a library for database access called ActiveX Data Objects (ADO). ADO uses SQL [SQL92] as its query language. A developer is free to use any third party library for database access as well.

ADO was used for all database access in the test implementation.

4.2.1.5 Naming or directory service

Both EJB and COM+ support location transparency, meaning that it is not necessary for the client of a component to know the physical location of the component. A client, at best, may only need to know the Domain Name Service (DNS) name to a server to get a reference to the component. An example on how to invoke a server object instance can be found in Appendix 4.

A description of how to locate the service of a component follows.

Naming and directory service in EJB
The specification states that a client can locate an enterprise bean home interface through the standard Java Naming and Directory Interface (JNDI) API.

A remote client may also obtain the metadata interface of an enterprise bean. The metadata interface is typically used by clients who need to perform dynamic invocation of the enterprise bean (dynamic invocation is needed if the classes that provide the enterprise client view were not available at the time the client program was compiled).

The specification also states that containers may optionally support runtime downloading of stub and value classes needed by the referencing container.

The CORBA 2.3.1 specification and the Java Language to IDL Mapping specify the way stub and value type implementations are to be downloaded.

WebLogic supports JNDI and metadata interfaces, but not the runtime downloading of the stub and the value classes. JNDI is also implemented in the test implementation for all clients. The metadata interface is not tested.

Naming and directory service in COM+
With COM+, there are several options for locating an application and its components. From the MMC, a developer can choose to export proxy stubs to a file that should be distributed to the clients. The proxy stub contains information about the server computer, its application and the interfaces supported by the applications. These are loaded into the registry of the client and are available as though they were on the local machine. The components can also be reached programmatically.

For the test implementation, client proxy stubs were created with the MMC and executed on all three clients.

4.2.1.6 Synchronization services

The application server should properly synchronize access in order to keep track of current activities in different threads.

Allowing components to start threads would lead to serious problems. An example would be to image two concurrent threads running with the same transaction context and trying to access an underlying database. If one thread is reading the data while the other thread is updating the data, it is completely unpredictable to know which data the first thread would read.

Synchronization services in EJB
The EJB specification makes it illegal for an enterprise bean to start new threads. The Container ensures that the system is manageable, and must control all thread creations.

For session beans, section 7.11.8 of the EJB specification states that the container must ensure that only one thread can be executing an instance at any time. Note that a session object is intended to support only a single client. Therefore, it would be an application error if two clients attempted to invoke the same session object. One implication of this rule is that an application cannot make loop-back calls to a session bean instance.

Multiple clients can access an entity object concurrently. The Container, in which the Entity Bean is deployed, properly synchronizes access to the state of the entity object by using transactions.

Synchronization services in COM+
COM+ features a service called activity-based synchronization [PLATT99]. This service provides locking and transaction features that keep track of the current activities in the different threads, that is, it provides automatic synchronization by the use of process-wide locking. There are four different settings available for this service (see Table 4-1).

Synchronization Setting	Creator in an activity	Creator not in an activity
Disabled	None	None
Supported	Activity of the creator	None
Required	Activity of the creator	New activity
Required New	New activity	New activity

Table 4-1: COM+ synchronization settings

4.2.1.7 Transaction handling

Transactions free the application programmer from dealing with the complex issues of failure recovery and multi-user programming. If the application programmer uses transactions, the programmer divides the work of the application into units called transactions. The transactional system ensures that a unit of work either fully completes, or the work is fully rolled back.

Transaction handling in EJB
In chapter 18 of the EJB specification, the following is stated: *One of the key features of the Enterprise JavaBeans architecture is support for distributed transactions. The Enterprise JavaBeans architecture allows an application developer to write an application that atomically updates data in multiple databases which may be distributed across multiple sites.*

EJB transaction attributes
The specification states in section 17.4.1 that the transaction attribute specifies how the Container must manage transactions for a method. Some attributes are not supported by container-managed Entity Beans and message-driven beans (see the specification for further details on this).

The specification lists the following transaction attributes in its section 17.6.2, listed in Table 4-2.

Transaction attribute	Client's transaction	Transaction associated with business method	Transaction associated with resource manager
NotSupported	None T1	None None	None None
Required	None T1	T2 T1	T2 T1
Supports	None T1	None T1	None T1
RequiresNew	None T1	T2 T2	T2 T2
Mandatory	None T1	Error T1	N/A T1
Never	None T1	None Error	None N/A

Table 4-2: Transaction attributes in EJB

The figure is quite self-explanatory and provides a summary of the transaction context. T1 is a transaction passed with the client request, while T2 is a transaction initiated by the Container. If the bean's business method invokes other beans, the transaction indicated in the "Transaction associated with business method" column will be passed as part of the client context to the target bean.

EJB Transaction modes

The EJB specification presents two different types of transaction models: programmatic (bean-managed) and declarative (container-managed). With bean-managed transactions, the enterprise bean code demarcates transactions. With container-managed transaction, the Container demarcates transactions per instructions provided by the settings of the component.

According to [GWE01], container-managed transactions should always be used.

EJB Isolation levels

The specification provides guidelines for implementing isolation levels (section 17.3.2 in the specification), but does not define the API to manage them because isolation levels are resource specific (e.g. not all persistent storages have support for all isolation levels). The isolation level describes the degree to which the access to a resource manager, by a transaction, is isolated from the access to the resource manager, by other concurrently executing transactions.

EJB Nested transactions
EJB does not support nested transactions, because it allows vendors of existing transaction processing and database management systems to incorporate support for Enterprise Java-Beans. If these vendors provide support for nested transactions in the future, Enterprise Java-Beans may be enhanced to take advantage of nested transactions (the EJB specification, section 17.1.2).

WebLogic fully supports all transaction modes proposed by the specification. In the test implementation, declarative transactions are used on all components with the Required transaction attribute and the Serializable isolation level set.

Transaction handling in COM+
COM+ supports distributed transactions through MTS and the MS DTC.

EJB	COM+
Never	Disabled
Supports	Supported
Requires New	Requires New
Not supported	Not supported
Required	Required
Mandatory	N/A

Table 4-3: Transaction mapping of EJB and COM+

When a component is added to an application, it is analyzed and MMC finds and sets the transaction attributes. COM+ transaction attributes is a subset of the EJB services, as indicated in Table 4-3.

The component has its default (set programmatically by setting a variable, or set manually in the MMC). The Microsoft Distributed Transaction Coordinator (MS DTC), a part of MTS, handles the coordination of the transactions. In a transaction, an object has to inform the transaction manager of the transaction's success or a failure before exiting the transactional context. The MS DTC can be run on a different server if desirable.

The MS DTC requires a resource manager to function. However, there are still too few RM's on the market that have all the functionality required by MS DTC and MTS, in order to enable all their features. The list of available resource managers can be found in Table 4-4.

MTS supports only the SERIALIZABLE isolation level, which is the strictest form of isolation available. The SERIALIZABLE isolation level guarantees

data integrity, however the prices to be paid is performance as it is the slowest of all isolation levels.

In the test implementation, MTS is used for all transactional activity. MTS fully supports nested transactions with the RM mentioned in Table 4-4.

RM	Version
Microsoft SQL Server	6.5 or higher
Microsoft Message Queue Server	
Oracle	7.3 or higher
Informix	
Sybase	
CA Ingres	

Table 4-4: Resource managers that fully support COM+

In addition to that, COM+ supports the transactional handling of non-database operations through the Compensating Resource Manager (CRM). If a developer wants to write a file as part of a transaction, the CRM, available from the MMC as an option on the component level, can be used to ensure that the transaction as a whole is rolled back, even if only the non-database operation failed [MSDN CRM] .

4.2.2 Usability

How can the general user interface help the users performing their tasks? Is the interface intuitive for all kinds of users?

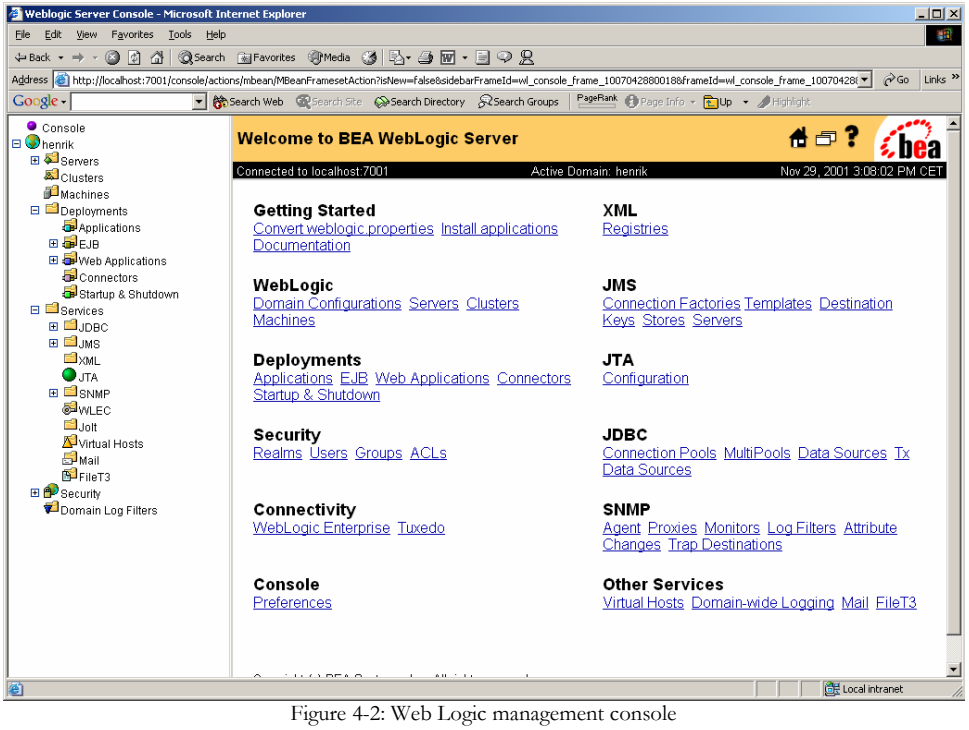

Figure 4-2: Web Logic management console

EJB usability

As a specification, EJB does not give any guidance on how the user interface should look, as this naturally will be specific to every implementation. WebLogic has solved this issue with a web-interface to runtime tuning of the application server (see Figure 4-2). The WebLogic internal web server must be up and running in order for this solution to function. Alternatively, the WebLogic configuration files in .xml-format are editable when the server is offline.

Component parameters are put in xml-descriptors and packed with the component and must be edited by a text editor or other third-party tool.

COM+ usability

Microsoft traditionally makes fairly intuitive graphical user interfaces for all their applications, and the management console for COM+

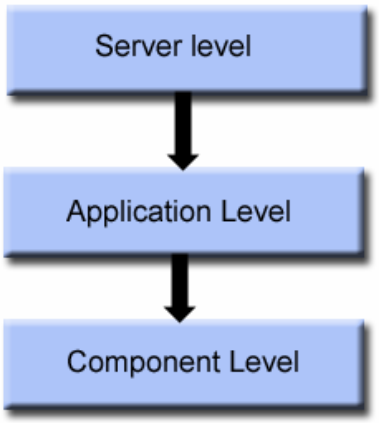

Figure 4-3: Hierarchy in COM+

(MMC) is, in this matter, no different (see Figure 4-1).

This is a hierarchical structure, where the lower layers inherit the defaults set in the higher ones (see Figure 4-3), and where some of the settings from the Server level can be overridden on both the Application and the Component level.

All these settings can be made programmatically as well, so that big operations, e.g. changing the security settings for several applications, will not require hours of work.

The MMC is used throughout the implementation.

4.2.3 Performance

Performance relates to how the application server performs, and is thoroughly described in section 4.5.

4.2.4 Security

The issue of security is becoming more important as more sensitive systems become distributed. The author of [PLATT99] goes as far as saying that "any system should have an excellent reason for not implementing a high degree of security".

EJB security

EJB uses the security mechanisms of J2EE, which are based on two separate security models. The first is called *declarative security* model, and expresses the security structure of an application, including roles, access control, and authentication requirements. All the latter can be changed without modifying the application. The second model is called *programmatic security*, and is about adding explicit security checks within the application code. Declarative security is preferred wherever possible in order to separate application code and security constraints.

Section 21.1 of the EJB specification encourages the developer to implement the enterprise bean class without hard-coding the security policies and mechanisms into the business methods. Because not all security policies can be expressed declaratively, the EJB architecture provides a simple programmatic interface that the developer may use to access the security context from the business methods.

Web clients can be authenticated over a Secure Sockets Layer (SSL) in Java Server Pages (JSP), and servlets. J2EE also enables integration with existing security systems. For instance, WebLogic can interoperate with Windows security.

According to the EJB specification (19.8: security interoperability), EJB supports the secure interoperable mechanisms based on the CORBA/IIOP protocol. It also supports Kerberos-based secret key mechanism and X.509 certificate-based public key mechanisms. Kerberos is a secure method for authenticating a request for a service in a computer network.

WebLogic Server relies on the standards-based technologies just explained for its security services.

COM+ security

The Windows Distributed Internet Applications Architecture (Windows DNA) security model is quite analogous to the one of J2EE. Web clients can be authenticated over Secure Sockets Layer (SSL) in Active Server Pages (ASP) or Internet Server Application Program Interface (ISAPI) code. Application code typically accesses credentials in Microsoft's Active Directory, and authorization can be either programmatically or declaratively controlled in COM+ components.

Authentication Level	Description	Security
None	No authentication	None
Connect	Authenticates only at connection	Low
Call	Authenticates for every call to the component	Medium
Packet	Authenticates and verifies that all call data is received	Medium
Packet Integrity	Authenticates and verifies that none of the data has been modified in transit	High
Packet Privacy	Authenticates and encrypts the packet, including the data and the sender's identity and signature	Very high

Table 4-6: Authentication settings in COM+

The COM+ can enforce security in several layers. All components in an application are set to run as a given user or as the user currently logged on. This helps ensure that the component process does not have access to e.g. files on the server. There are two other settings and one additional feature that can be set for security in COM+.

First of all, there is the authentication layer, which sets the standards for the level of identification that the client must provide to the component process. This setting has been divided into six authentication levels, see Table 4-6.

Then there is the impersonation layer, which sets the standards for the way how a component process can impersonate the client. It is useful for paranoid database access, where the different users have different access rights to the database, and so the component process can use the identity of the client to access the database. This setting has been divided into four different impersonation levels as described in Table 4-7.

Impersonation level	Description
Anonymous	The client is anonymous to the server application.
Identify	The server application can obtain the client's identity, and can impersonate the client to do access list checks.
Impersonate	The server application can impersonate the client while acting on its behalf, but with restrictions.
Delegate	The server application can impersonate the client while acting on its behalf, whether it is or not on the same computer as the client. During impersonation, all of the client's credentials can be communicated to any number of computers.

Table 4-7: Impersonation settings in COM+

Last, there is the ability to define roles. The administrator can set that e.g. Jim is a member of the Managers and is therefore able to access the administration component.

Every component in COM+ can enable object construction, this in order to pass a string to the component as it is being created. Typically, a string is inaccessible to the client. Kerberos comes with Windows 2000, and is fully supported by COM+. All these settings can be done either programmatically or manually in the MMC.

The test implementation does not include the security features as it is outside the scope of this thesis.

4.2.5 Reliability and availability

This section tries to answer whether the application servers perform correctly over extended periods of time, and how they handle failure. Are there possibilities for fault-tolerance with duplicate hardware and/or software?

Clustering & Load Balancing in EJB

The EJB specification does not mention these properties, and hence leaves the issue up to the application server implementation. However, it does, at some point, take into account the fact that the clustering of application servers is a widespread phenomenon.

A WebLogic Server cluster is a group of servers that work together to provide a more reliable application platform than would a single server. It also improves the application scale (see section 4.2.6). A cluster appears to its clients as a single server but is, in fact, a group of servers acting as one. If one server fails, another can take over. The ability to fail-over, from a failed server to a functioning server, increases the availability of the application to clients.

Several load balancing algorithms are supported by WebLogic: Round-Robin, weight-based and random. The round-robin algorithm cycles through a list of WebLogic Server instances in sequence. The weight-based algorithm improves on the round-robin algorithm by taking into account a pre-assigned weight for each server. Finally, the random algorithm chooses the next replica at random.

Reliability in EJB

When it comes to reliability, the extensive tests and the often used trial and error approach put the technologies up for a challenge. The WebLogic server crashed two or three times, including times when reboot of the operating system was a necessity. The most probable cause was the trial-and-error eccentric parameter settings in the server, and cannot be taken into account. The WebLogic server behaved exemplary and showed no signs of unreliability. NSB-Lisa, the authors' EJB reference project, had problems when the database became unavailable; WebLogic did not continue operations before it had been restarted. At the time of the writing of this thesis, the developers and administrators in the project are unsure if the servers are correctly set up.

Handling failure in EJB is analogous in Java, and exceptions from the server can easily be caught on the client side.

Clustering & Load Balancing in COM+

COM+ solves the issues of clustering and load balancing by adding at least two servers to create a Component Load Balancing (CLB) cluster. The first server, the one that the clients see as the only server in the network, is formally known as the Application Cluster Router. This server is in charge of dynamically balancing the load of the servers in the network. This is done by the other servers which report back to the load balancing server regularly with reports on their current load. The rest of the servers are the "slaves" and do the actual work (while the Application Cluster Router takes all the credit). Now, the administrator can add as many "slaves" as needed in order to maintain an acceptable performance.

Reliability in COM+

The Rikstoto project had an overall excellent experience with COM+. It was running smoothly all along. The only problem the project encountered was a taste of the "DLL hell", a situation that arises in the Windows registry when

changing the interface of a component and the lack of DLL versioning support. With this single exception, everything was running very well.
As for the MSMQ functionality, if a message fails and is to be sent to the dead letter queue, as an option component can be launched to correct the error that has occurred. It serves as an exception class for the component.

4.2.6 Scalability

As a system grows, the response time of the application will become higher and higher, until the system is left unusable. One solution to this problem is adding more hardware (clustering) and making sure that the hardware is used properly (load balancing). Other mechanisms are also implemented in applications server in order to make a system scale.

Scalability is defined by Roger Sessions in his *Objectwatch* newsletter, issue #26:

> I consider a system to be "scalable" if we can add more workload to the system without increasing the cost of the system per unit of workload. The common unit of workload for a commerce system is a transaction.

The industry standard benchmark for transactional throughput is specified by a consortium called the Transaction Performance Council (TPC) and the benchmark is called the TPC-C benchmark [TPC].

On its website, TPC presents the Top Ten TPC-C by performance. Two different tables are presented, one with clustered solutions and one with non-clustered solutions. On the clustered list, COM+ is the only participant offering eight different solutions. It is reasonable to believe that no other vendors have delivered TPC numbers for clustered solutions.

However, on the non-clustered list, Websphere, being the only EJB-vendor with publicly released TPC-C numbers, ranks in at number 5, ahead of all COM+ solutions. Because this has much to do with the kind of hardware in use, the number of CPUs, and the number of servers put in a cluster, the results appeared as fairly irrelevant to this particular comparison context.

Therefore, the TPC-C has to be seen together with the rate of the new order transactions (tpmC), which gives COM+ monopoly of the top ten list. More comments on this can be found in section 4.4.2, where the cost of system quality is presented.

Scaling mechanisms
With the features implemented in both technologies, scalability is primarily a question of good design. A lot of hardware cannot compensate for an application that does not scale much.

The joint mechanisms for scaling are:

- Object pooling
- Database connection pooling
- Load balancing in a cluster

EJB

The fact that an implementation should scale has nothing to do with the EJB specification. However, the specification section 7.8 states the following: *Because Stateless Session Beans minimize the resources needed to support a large population of clients, depending on the implementation of the container, applications that use Stateless Session Beans may scale somewhat better than those using Stateful Session Beans. However, this benefit may be offset by the increased complexity of the client application that uses the stateless beans.*

WebLogic supports all mechanisms mentioned: load balancing in a cluster, as described in section 4.2.5, and object pooling and instance management, as described in section 4.2.1.3.

Database connection pooling is supported as of JDBC version 2.0, and hence is supported by WebLogic. This is described further in section 3.4.5.1.

COM+

COM+ also supports all mechanisms mentioned: load balancing in a cluster, as described in section 4.2.5, and object pooling and instance management, as described in section 4.2.1.3.

Database connection pooling is built into the ODBC, and MTS uses ODBC for database access.

4.2.7 Upgradability

Upgradability brings to the question of whether the system at runtime can upgrade new features or versions without bringing operations to halt. It is an important issue to the systems that demand continuous operation.

EJB upgradability

The EJB specification states in 3.1.5 that "*The Container Provider typically provides support for versioning the installed enterprise Bean components. For example, the Container Provider may allow enterprise Bean classes to be upgraded without invalidating existing clients or losing existing enterprise Bean objects."* This is more a suggestion than a demand to the Container Provider. Because the automatic redeployment feature in WebLogic uses dynamic deployment, the server can only redeploy EJB's implementation classes. Redeployment of EJB's public interfaces can be done without restarting the application server. However, changing interfaces requires the application server to restart.

The issue might be solved with a clustered solution by taking down one server for the redeployment of Entity Beans, while another server is up serving the

public. This solution is not tested because it is beyond the scope of this thesis. But, according to postings in the BEA newsgroups, it is possible and has been implemented. WebLogic has no feature allowing the upgrade of the WebLogic version while it is running.

When taking down a server in a clustered environment, local state (in stateful components) needs to be replicated to another server to preserve the state. WebLogic has functionality to ensure state preservation across servers in a cluster.

COM+ upgradeability
According to [RS99], a COM+ application server can bring down any machine in a cluster for system upgrade.

COM+ has a system of CLS (class ID's) that uniquely identifies a component. Since COM+ supports inheritable multiple interfaces and each interface have its own unique Globally Unique Identifier (GUID), not all clients have to be updated simultaneously when an interface is modified. The old clients use the old interface, while the updated clients use the new one. After all clients have been updated, the old interface will be phased out.

COM+ has no functionality to ensure state preservation when taking down a server in a cluster.

4.3 Development qualities

Development qualities represent the level of easiness in designing and maintaining an application developed for the respective application server. The qualities must be observed during development and maintenance activities, because they relate to the design structure and the way it can be manipulated.

4.3.1 Modifiability

This is the ability to modify a component without having to rebuild all other components related to the deployed application. The redeployment of modified components is covered in section 4.2.7. When designing a deployable application, it is important to consider the component packaging and possible limitations that may arise. This is covered in section 4.2.1.

4.3.2 Reusability

In distributed systems, reusability is a key concept. It reflects the ability of the system to reuse other components, that is, components from other vendors and/or platforms as well as third-party components.

Interoperability is an important part of reusability. Indeed, new systems might need to communicate with existing ones, potentially huge and implemented with outdated technology, in order to avoid rewriting of the existing systems.

Interoperability is defined in [WHATIS] as the ability of a system or a product to work with other systems or products without particular effort on the part of the customer. In this section, the focus is on interoperability.

EJB reusability and interoperability
As mentioned in 4.2.1, the EJB specification states that containers may optionally support the runtime downloading of the stub and the value classes needed by the referencing container. At the time of the writing, however, this is not supported by WebLogic.

According to the EJB specification, new in EJB 2.0 is a defined interoperability protocol based on CORBA/IIOP to allow remote invocations on session and Entity Beans from J2EE components, which are deployed in products from different vendors. CORBA clients can be written in a variety of languages and use the Interface Definition-Language (IDL) to interact with a remote object.

Clients wishing to use the COM+ protocol communicate with the server component through a COM-CORBA bridge. EJB supports web clients through servlets, Java Server Pages, or similar Web extensions.

The specification does not address interoperability issues between enterprise beans and non-J2EE components.

As of version 6.1 of WebLogic, Web Services is supported, which makes communicating with e.g. COM+ web services straightforward. Web Services is not part of the EJB 2.0 specification.

All interoperability issues discussed above are supported by WebLogic as of version 6.1.

COM+ reusability
COM+ does not have any fixed network protocol; it can utilize any protocol installed on the server. The default protocol to use is TCP/IP, but any other protocol can be chosen from the MMC. COM+ components are available from any computer running DCOM, or simply COM, for local access to components on the server.

HTTP tunneling is available through the COM Internet Services (CIS) [MSDN CIS], which allows COM+ components to interact through port 80 in order to enable access through proxies and firewalls.

COM+ components can be activated from the Internet Information Services (IIS), the web server that comes with Windows NT, Windows 2000, and Windows XP.

Another way of communicating with e.g. EJB is to exchange information through Web Services, a service that is part of Microsoft .NET but is already implemented for BEA WebLogic, Windows 2000, and Windows XP. Web Services exchange data over SOAP/XML and can therefore easily communicate with other systems.

COM+ components can communicate with CORBA/IIOP through a COM-CORBA bridge.

4.3.3 Portability

Does the system design permit easy porting to other platforms? Are there hardware and infrastructure dependencies localized in the implementation?

EJB is a specification and COM+ an implementation. As previously mentioned, both technologies are starting at two opposed ends and finishing where the other started. EJB is open for everyone to implement, COM+ has already been implemented.

EJB portability

The EJB specification strives to specify programming restrictions of portable enterprise beans in its chapter 24.

WebLogic is committed to the EJB 2.0 specification but, as do other vendors, has its own extensions to the product. Features, such as extensions to the QL-language and Read-Only Entity Beans [BEA 01b], are proposed for the next version of the EJB Specification but are already implemented in WebLogic.

In order to be a portable cross-vendor, the developer must use only features described in the specification, and possibly be willing to sacrifice useful and convenient extensions. In addition to that, several parameters are set on the application server, such as tuning parameters and component database field-to-field mapping, which are WebLogic specific and probably need manual porting.

Java, and hence EJB, are platform independent; therefore only an implementation of EJB needs to exist in order for a platform to be EJB-ready.

WebLogic is currently available at the following platforms: ***NT, Solaris, HP-UX, AIX, Tru64, Windows 2000, OpenVMS, AIX 4.3.3, Sequent Dynix 4.4.4, OS/400 V4R4, Linux, SGI Irix 6.5, SNI Reliant 5.44C, Unisys OS1100, Unisys Burroughs, OS/390 V2R6***

COM+ portability

COM+ was designed and implemented by Microsoft. This is not based on a specification, but on Microsoft's proprietary ideas.

COM+ is currently available only for Windows 2000 and Windows XP running on either Intel or Alpha hardware. Microsoft does not have any plans of releasing any UNIX versions of COM+ in the near or distant future.

Since COM+ is by nature operation system close, it can have access to the memory above 2 gigabyte in Windows 2000 Advanced Server.

4.3.4 Buildability

This measures whether the system is easy to implement and to build, and what third-party components or libraries it takes advantage of.

EJB buildability

The choice of the programming language is very straightforward. EJB is based on, and supports Java and only Java.

The J2EE platform supports other languages through the Java Native Interface (C++) and through CORBA interoperability. SUN recommends the latter approach [CATTELL00].

The development environment is provided by third-party tools. Editors, Java Virtual Machine, Debuggers, etc. are all third-party tools. The choice between different tools and vendors is large and some good tools are even free of use.

COM+ buildability

The COM+ platform natively supports several languages, and leaves it to the developer's discretion to choose the language best suited for the needs of a component. COM+ has been designed to run especially with Visual Basic, C++, ASP (VBScript), but can be invoked from various languages.

COM+ has common tools, editors, runtime environments, and tools all wrapped up in an integrated development environment (IDE) called Visual Studio. This package tool is included when buying either of Microsoft's Visual Tools. Any language on the platform that can access the Windows API can be used for developing COM+ components. Hence the developer does not necessarily have to use the Microsoft suite of tools. A developer is also free to use a third-party editor.

4.3.5 Testability

The testability metric measures how easy it is to demonstrate defects in the system, or in the application server context, the deployed application. This is determined by the degree of easiness in accessing the internal state and inputs of the components so they can be stimulated and observed.

EJB testability
Testing is beyond the scope of the EJB specification. WebLogic comprehensive log files from the server are available and can be tuned runtime to contain all debug messages through only fatal errors.

In EJB, there is a need for third-party tools to debug java code inside the components. NSB-Lisa uses BugSeeker from Karmira [KARMIRA] and Visual Café from WebGain [WEBGAIN]. Other third-party tools both for measuring performance and pinpointing performance bottlenecks are available for Java from e.g. Rational. Some tools, such as JProbe from Sitraka [SITRAKA], specifically support BEA WebLogic server.

COM+ testability
The IDE that comes with Visual Studio has a built-in debugger that allows stepping through code line by line, setting breakpoints, watching variables, debugging components that run under MTS, and many other features. Visual Studio and Windows 2000 come with a set of performance testing and general system monitoring tools. Each component can be enabled for statistics and event reporting.

When launched, a COM+ application can be set started in a debugger. The developer can specify which debugger to use, as well as which command line options to use for the latter.

4.4 External qualities

External qualities are important to the person in charge of choosing technologies, e.g. the project leader.

4.4.1 Time to market

Today, the technology strides ahead and delivering projects fast is alpha omega.

As mentioned earlier, EJB has a specification that is time consuming to implement. Sometimes, the vendors impatiently advance faster than the specification itself.

In NSB-Lisa, when using the first version of WebLogic 6.0, the implementation came out only a few days after the specification was released to the public. The finder-methods did not compile, even though it was consistent with the specification. This was corrected in a service pack later on.

Microsoft's technologies have a short time to market as Microsoft does not have to wait for specifications to be accepted by several parties. As they are the masters of their domain, they do not have to wait for anyone else but themselves.

The time to market for developing projects for a customer depends mainly on simplicity issues and domain knowledge, which are described in section 4.4.4.

4.4.2 Cost of system

The cost of system metric reflects the total cost of the entire platform on which to run the application server. Not included are the costs related to teaching developers how to use the application server properly. The latter is covered in section 4.4.4.

According to BEA Norway, a license on a BEA WebLogic server starts at approximately USD 8000. The database or persistent storage is not taken into account. In addition to that, an operating system and appropriate hardware on which to run the server must be purchased. Third-party components are necessary to develop against the server.

However, other free implementation exists: JBOSS is available under a public license, alternative operating systems, such as Linux, are available for free [JBOSS].

The current price of Microsoft Windows 2000 Advanced Server is USD 3999 with 25 client licenses. It includes the operating system but excludes the programming languages, which translates into an additional USD 1690. The supported hardware is either Intel (or compatible) or Alpha servers.

The Transaction Performance Council (TPC) has published metrics for Top Ten Non-Clustered TPC-C by Price/Performance. When looking at the associated price per transaction (USD/tpmC) of all submitted non-clustered configurations (see Table 4-8), Microsoft positively stands out.

HW Vendor	System	tpmC	Price/tpmC	System availability	Database	Operating system	TP Monitor	Date submitted
DELL	PowerEdge 2500/1.13/1P	11,320	USD 4.38	10/31/01	Microsoft SQL Server 2000 Standard Edt.	Microsoft Windows 2000 Server	Microsoft COM+	10/31/01
IBM	IBM eServer pSeries 660	57,346	USD 28.47	06/19/01	Oracle9i Database Ent. Edition 9.0.1	IBM AIX 4.3.3	Webshpere App. Server Ent. Edition V.3.0	04/23/01

Table 4-8: TPC-C by Price/Performance for COM+ and Websphere

As mentioned in section 4.2.6, the top ten list of Price/Performance is monopolized by COM+. As of now, no numbers about clustered solutions are available for non-COM+ technologies.

4.4.3 Maturity

The first EJB specification was released in 1998 and the first beta implementations in 1999. It was three years after the first implementation of MTS, the forerunner of COM+. The major EJB application servers use transaction monitors that existed long before the introduction of MTS. WebLogic uses Tuxedo as its transaction monitor. Tuxedo was first introduced in 1978 [TUXEDO].

4.4.4 Simplicity

The simplicity measures how easy it is for the application server users to gain an understanding, and to be able to develop applications for the server.

According to [THOMAS98], important simplicity factors are a number of development options and automation for the application programmer. Thomas calls this a draw between the two technologies, because COM+ (or MTS) might be considered easier due to fewer development options. As tools available for EJB automate a lot for the application programmer, EJB development will probably be the easiest solution for most applications.

4.4.5 Future plans

What are the plans for the two technologies in the nearest future?

The features deferred to future releases according to the current EJB specification are as follow:

- support for other types of messaging in addition to JMS
- aggregate operations and other extensions to EJB QL
- read-only Entity Beans with container-managed persistence
- specification for the pluggability of Persistence Managers
- support for method interceptors
- support for component-level inheritance

Everything implies that Java remains EJB's language. COM+, on the other hand, will probably support every new language. EJB will probably cooperate closely with CORBA in the future, as the technologies, after all, are pretty similar.

As for Microsoft, they have recently released their .NET platform for development [MS .NET]. The most interesting service in .NET is their innovation named Web Services, which allows applications to communicate and share data over the Internet.

There is, to this date, no information on what Microsoft plans to do next.

4.5 Performance

In this section are presented the results of the performance test. The focus is on the server CPU load and on the average response time (ART) [GZ00]. The section is divided into the four business cases, and results from both technologies are presented in the subsections.

The final subsection presents two other interesting findings discovered while looking at the result logs. Firstly, how the application server handles RDBMS connections, and secondly the connection between the CPU load and the number of threads allocated by the application server.

It is important to note that the results indicate the performance on the hardware (see Appendix 1) and the configuration used (see section 3.4.5).

4.5.1 New Customer business transaction

The New Customer business transaction is implemented, as described in section 3.4.4.3, as a stateless component and the developer handles the persistence programmatically.

4.5.1.1 COM+ results

Figure 4-4 shows the server CPU load for COM+ when it runs 60 (blue), 150 (red) and 250 (yellow) clients. The graph of the three different cases form a similar pattern that indicates that the higher the load is, the more time it takes to complete the task.

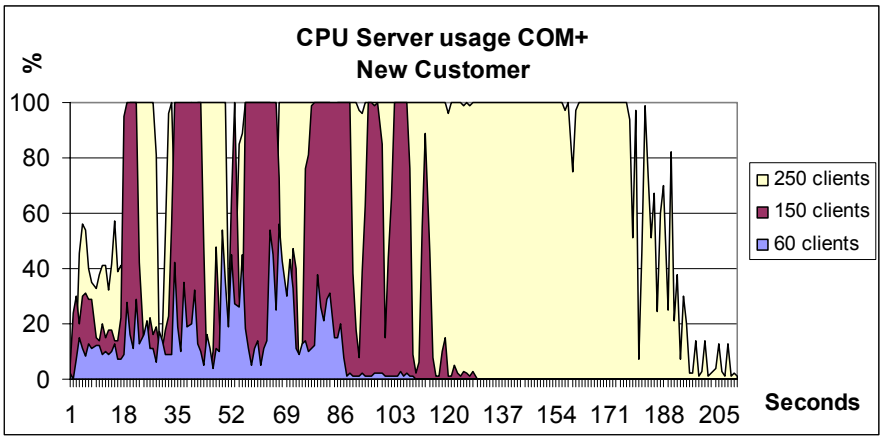

Figure 4-4: CPU usage in the COM+ "New Customer" case

At 150 clients, the server load is maximized for shorter periods, while, at 60 clients, the server load never exceeds 60%. It takes the server about 85, 121 and 197 seconds to complete the task at hand for 60, 150 and 250 clients.

To keep the server load at less than 100%, the number of concurrent users creating new customers should be less than 150.

Clients	Average Response Time in milliseconds (ART)	Transactions per second (TPS)
60	16	62,5
150	365	2,7
250	3043	0,3

Table 4-9: Average response time in the COM+ New Customer case.

Table 4-9 shows that by adding 90 clients to a total of 150, the average response time increases radically by 2181%. Table 4-9 combined with Figure 4-4 indicates that once the server load reaches 100%, the response time increases dramatically.

4.5.1.2 EJB results

Figure 4-5 is the EJB counterpart to Figure 4-4, hence it describes the server load while running the EJB "New Customer" business transaction. In this figure, there is a longer stretch on the X axis, the time axis, since the EJB solution takes a longer time to complete, especially in the case with 250 clients.

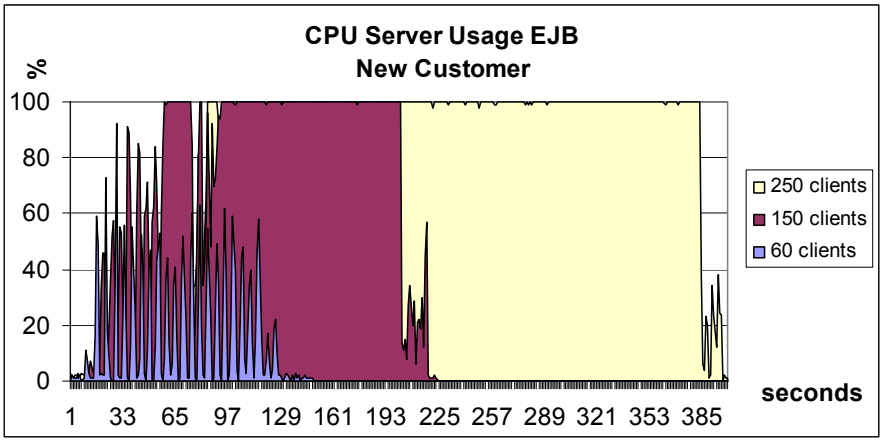

Figure 4-5: CPU usage in the EJB "New Customer" case

It appears that, when the server had the time to start up all the clients in the cases of both 150 and 250 clients, it hits 100% load and stays there until completion of all the clients. It takes the server about 133, 215 and 397 seconds to complete the three cases.

Clients	Average Response Time in milliseconds (ART)	Transactions per second (TPS)
60	168	5,95
150	4088	0,25
250	8756	0,11

Table 4-10: Average response time in the EJB New Customer case.

Table 4-10 shows the EJB result for this business case, and is the counterpart to Table 4-9. The leap from 60 to 150 clients caused a 2333% increase in the average response time.

4.5.1.3 EJB vs. COM+ results

The previous graphs, which show EJB and COM+ CPU usage, have similar patterns in the way that they both look alike for 60, 150 and 250 clients: the higher the client count is, the more the graph stretches on both the X and the Y axes.

The EJB solution uses more time to complete the transaction (see the X-axis on the two CPU usage graphs), independently of the number of clients. It also presents a higher server load and processes fewer transactions per second.

Figure 4-6 shows that the difference in transactional throughput per second is nearly ten times in favor of COM+ for 60 and 150 concurrent clients. However, at 250 clients, when the CPU load is staying at 100% for both technologies, the difference in transaction throughput per second is only about 170%.

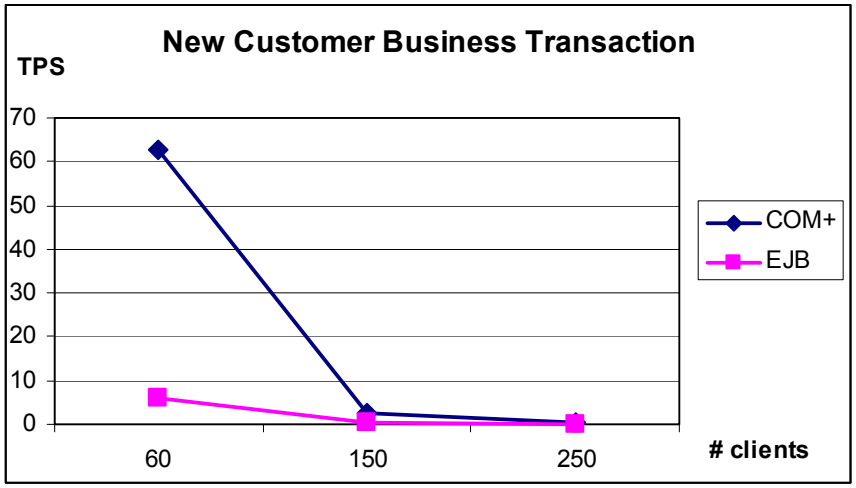

Figure 4-6: Transactions pr. second comparison for the New Customer Business Transaction

4.5.2 Populate Shopping Cart business transaction

The "Populate Shopping Cart" business transaction is implemented as a stateful component with no access to persistent storage (see section 3.4.4.3).

4.5.2.1 COM+ results

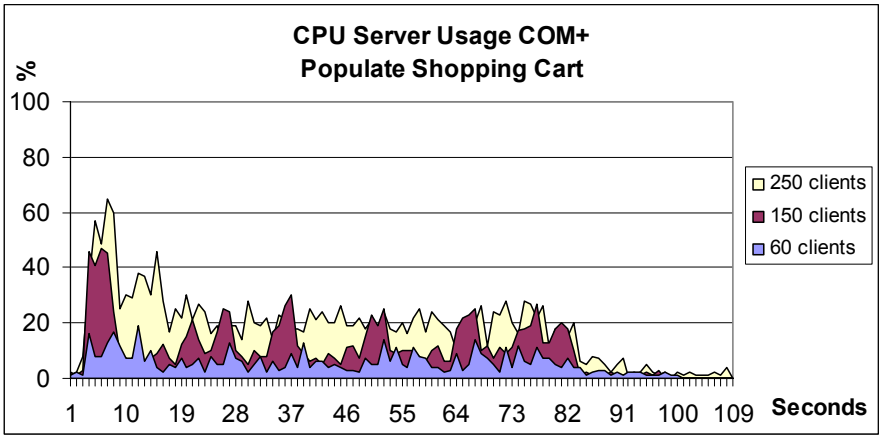

Figure 4-7: CPU usage in the COM+ "Populate Shopping Cart" case

As indicated in Figure 4-7, even with 250 clients pounding the server, the load on the server never exceeds 70%. Another interesting fact is that all the configurations use approximately the same amount of time, about 85 seconds, to complete their tasks.

Clients	Average Response Time in milliseconds (ART)	Transactions per Second (TPS)
60	1,8	555,6
150	2,67	374,5
250	1,97	507,6

Table 4-11: Average response time in the COM+ Populate Shopping Cart case.

In the "Populate Shopping Cart" business transaction case, the increase in ART from 60 to 150 clients is only 48%, and drops by 35% when the number of clients is increased from 150 to 250.

4.5.2.2 EJB results

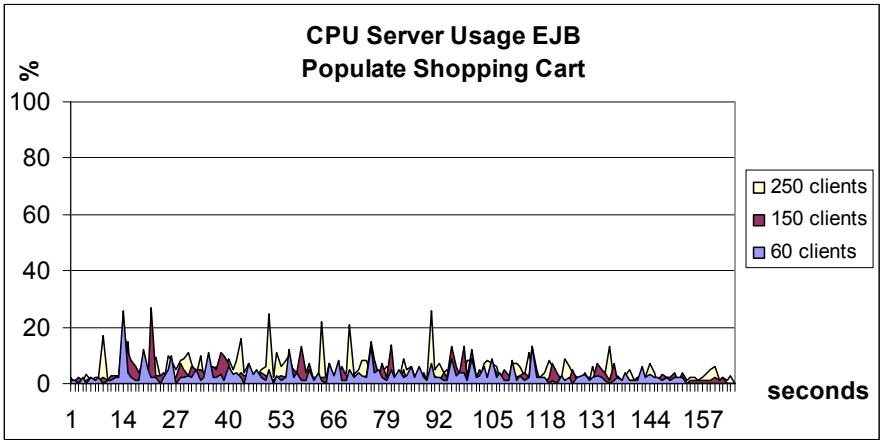

Figure 4-8: CPU usage in the EJB "Populate Shopping Cart" case

As shown in Figure 4-8, the server load never exceeds 30% with the EJB implementation. All three configurations of this business transaction case take about 145 seconds to complete their tasks.

Clients	Average Response Time in milliseconds (ART)	Transactions per second (TPS)
60	2,23	448,4
150	3,19	313,5
250	3	333,3

Table 4-12: Average response time in the COM+ Populate Shopping Cart case.

For the EJB solution, the increase in ART from the 60-client configuration to the 150-client configuration is as low as 30%. When the number of clients is increased from 150 to 250, the ART drops by 6%.

4.5.2.3 EJB vs. COM+ results

While the server load using the EJB implementation is next to nothing, and certainly much lower than the one of the COM+ implementation, the ART is longer for the EJB implementation and the EJB implementation also uses a longer time to complete its task. As indicated by Figure 4-9, the difference in transactional throughput for 60 and 150 clients is about 20%. However, when running with 250 clients, the difference increases to more than 50%. Interestingly enough, the transactional throughput is higher when running 150 clients than it is when running 60 clients.

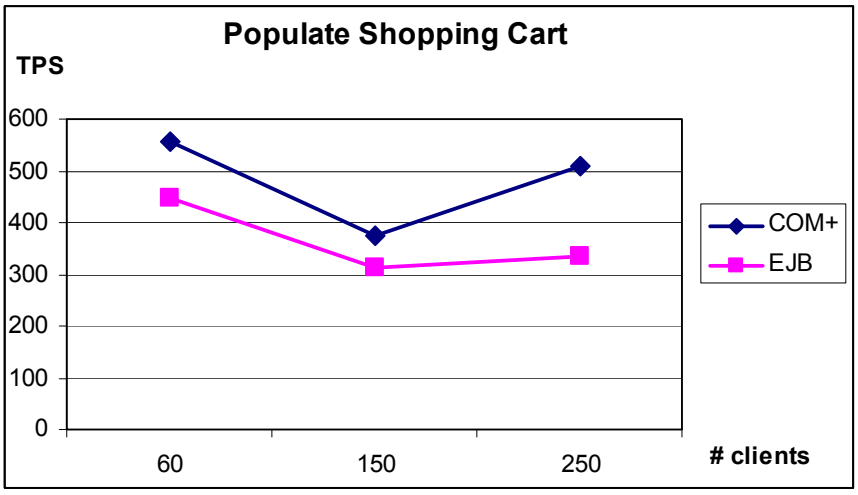

Figure 4-9: Transactions per second comparison for the Populate Shopping Cart Business Transaction

4.5.3 New Order business transaction

The New Order business transaction is implemented, as described in section 3.4.4.3, as a stateful component with persistent storage.

4.5.3.1 COM+ results

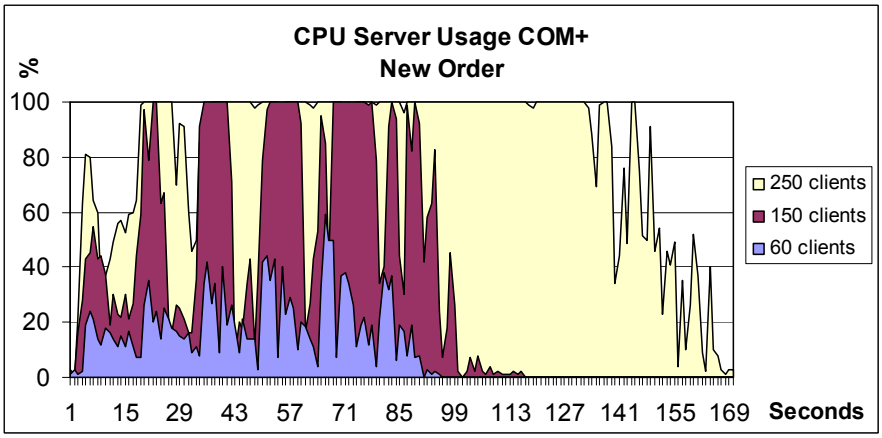

Figure 4-10: CPU usage in the COM+ "New Order" case

Figure 4-10 shows that running this business transaction with 150 clients, the CPU load of the server hits 100% repeatedly, while it stays below 60% with 60 clients. Running 250 concurrent clients puts the server load to 100% for most of the time that it takes to complete the task. The time spent completing the task increases from the 90-100 seconds compared with the time that it

takes with 60 and 150 concurrent users, about 165 seconds with 250 concurrent users.

Clients	Average Response Time in milliseconds (ART)	Transactions per second (TPS)
60	18	55,6
150	386	2,6
250	1932	0,5

Table 4-13: Average response time in the COM+ New Order case.

As indicated in Table 4-13, the TPS drops from 55,6 when running 60 concurrent clients to 2,6 when running 150 concurrent clients. The ART increases by 2044% from 60 to 150 clients, and increases by another 401% when running 250 concurrent clients.

4.5.3.2 EJB results

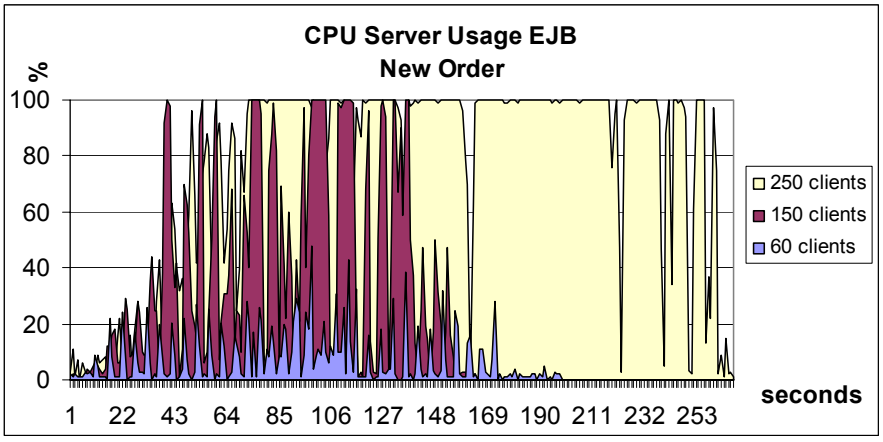

Figure 4-11: CPU usage in the EJB "New Order" case

As seen in Figure 4-11, while the CPU load runs 60 concurrent clients, it remains well below the 50% mark. Adding 90 clients makes the server reach 100% for shorter periods of time. Adding another 100 clients, in order to reach 250 concurrent clients, causes the server CPU load to be at 100% for more than half the time that it takes to complete the task. In spite of this, the 150 client configuration used about 10 seconds less than the 60 client configuration to complete its task.

Clients	Average Response Time in milliseconds (ART)	Transactions per second (TPS)
60	33	30,3
150	568	1,8
250	4040	0,2

Table 4-14: Average response time in the EJB New Order case.

As for the ART, as indicated in Table 4-14, the increase from 60 to 150 clients is 1621%, while the increase from 150 to 250 concurrent clients is 611%. There is also a massive drop in TPS from both 60 to 150 clients and from 150 to 250 clients.

4.5.3.3 The EJB vs. COM+ comparison

The EJB solution does not load the server as much as the COM+ solution does, but it uses substantially more time to complete the tasks for all 3 client counts. COM+ has longer ART for all client counts as well. Figure 4-12 shows that when running 60 clients, COM+ processes almost twice as many transactions per second.

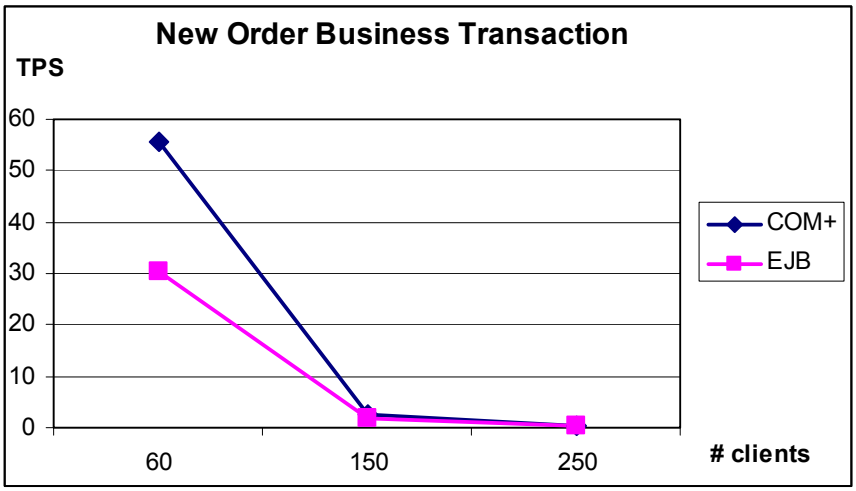

Figure 4-12: Transactions per second comparison for the New Order Business Transaction

4.5.4 Let's Buy Some Records business transaction

The "Let's Buy Some Records" business transaction was implemented with a persistent component, as described in section 3.4.4.3.

4.5.4.1 COM+ results

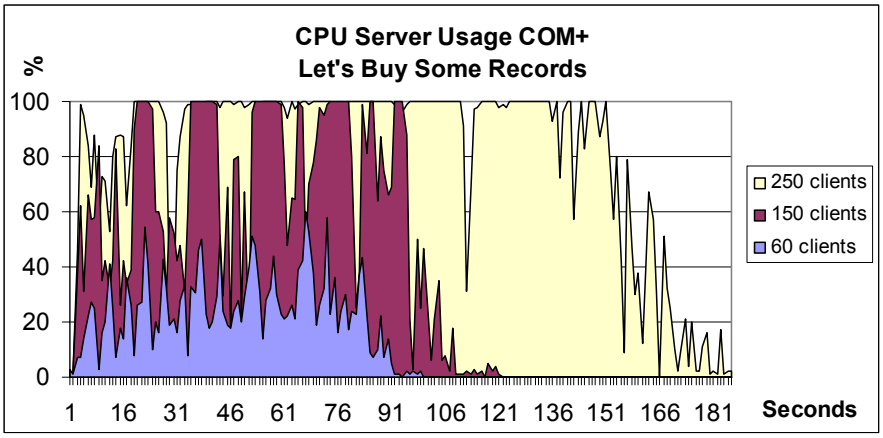

Figure 4-13: CPU usage in the COM+ "Let's Buy Some Records" case

The pattern noticed in the three previous business transactions (sections 4.5.1, 4.5.2 and 4.5.3) repeats itself for this last one as well. As shown in Figure 4-13, the server load peaks at about 60% when running 60 concurrent clients, while peaking at 100% several times when running 150 concurrent clients. When running 250 concurrent clients, the server CPU load is at 100% more than half of the time that it uses to complete the tasks.

Clients	Average Response Time in milliseconds (ART)	Transactions per second (TPS)
60	30	33,3
150	548	1,8
250	2284	0,4

Table 4-15: Average response time in the COM+ Let's Buy Some Records case.

The ART increases by 1727% when the number of concurrent clients is increased from 60 to 150, as calculated from the numbers presented in Table 4-15. The increase in the ART from 150 to 250 concurrent clients is 317%.

4.5.4.2 EJB results

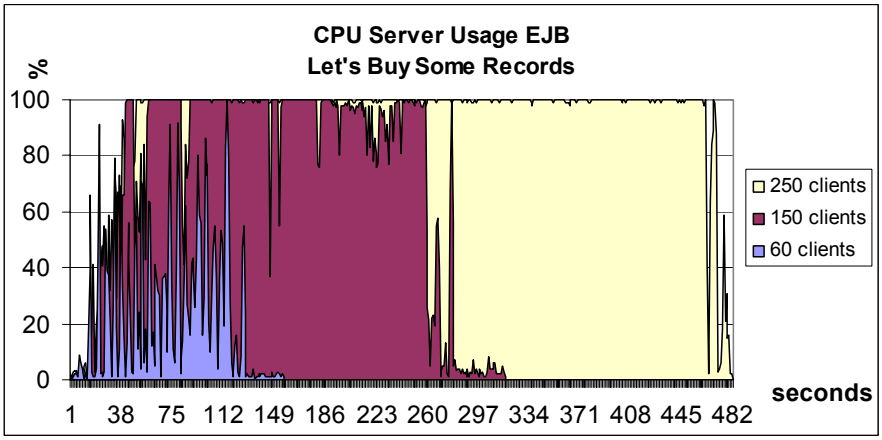

Figure 4-14: CPU usage in the EJB "Let's Buy Some Records" case

Figure 4-14 indicates that, while peaking at about 90% with 60 clients, the CPU server load is at 100% most of the time with 150 clients. The time spent to complete the tasks for 60, 150 and 250 clients is about 120, 300 and 470 seconds, which make it the slowest business transaction to complete its job.

Clients	Average Response Time in milliseconds (ART)	Transactions per second (TPS)
60	223	4,5
150	4806	0,2
250	15707	0,1

Table 4-16: Average response time in the EJB Let's Buy Some Records case.

The average response time, as indicated in Table 4-16, is drastically longer than the response time in the three other tests. The increase in response time from 60 to 150 clients is about 1963%, and the increase from 150 to 250 clients is approximately 227%.

4.5.4.3 EJB vs. COM+ comparison

The EJB solution uses almost three times longer to complete the task when running 250 concurrent users. The EJB solution has 100% CPU server load for almost the entire test run, both for 150 and 250 concurrent clients. COM+ uses substantially less CPU time for both of these two test runs. When running 60 concurrent users, COM+ and EJB load the server equally

much, about 25% in average. As Figure 4-15 shows, at 60 clients the number of transactions processed per second is 6,5 times higher with COM+ than with EJB.

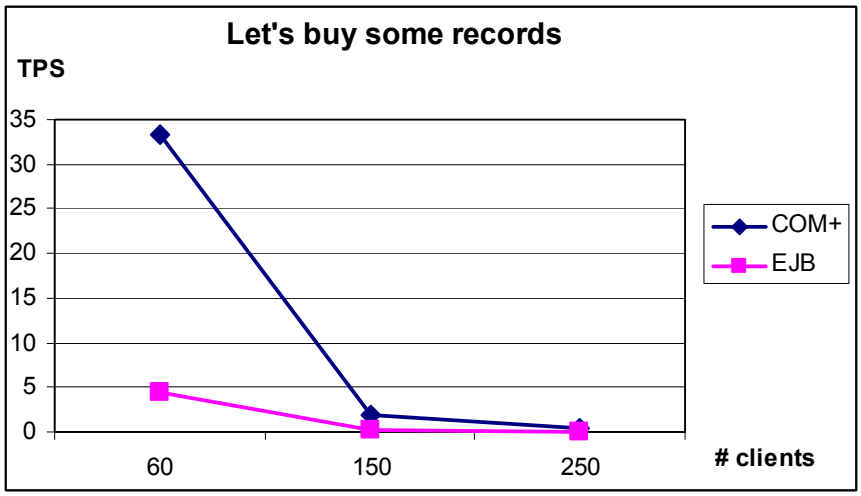

Figure 4-15: Transactions pr. second comparison for the Let's Buy Some Records business transaction

Chapter 5

DISCUSSION

In this chapter, the results and the facts presented in the previous chapters will be used to perform a direct comparison between the two technologies, as well as to carry out a discussion on the meaning of these results. The chapter is organized in a manner similar to the one of the previous chapter, in respect to headings and subsections.

5.1 Runtime qualities

A comparison of the runtime qualities in the two architectures is presented in this section. Only the WebLogic implementation of EJB will be taken into account.

5.1.1 Functionality

As described in section 3.3.1, functionality in this case represents the capability of the system to assist the user in completing its tasks.

5.1.1.1 Event management

Event-driven systems are important in distributed technology [COULOURIS01], because distributed systems are asynchronous by nature. As the Internet grows and more distributed services become available, event-driven systems become a matter of necessity.

Event management is supported in both technologies and complies with Sun's definition of a component model. Both technologies support asynchronous messaging. Asynchronous messaging is not tested in the implementation presented in the thesis, due to lack of vendor-implementation on the EJB platform.

The COM+ platform has a built-in queue manager within the MSMQ. The EJB platform fully relies on a third party product. However, both technologies will, in most cases, have MQ client support (EJB in all). On paper, the two technologies appear as similar.

5.1.1.2 Component packaging

The importance of component packaging presents significant differences between the two technologies. On one hand, Microsoft is not required to invest much into the technology since COM+ is only to be used on the Windows platform. On the other hand, EJB, because of their portability, ought to support a host of platforms. Sun has included the component

packaging service in their component model definition, for the sole and unique reason of the portability issue.

Both technologies have their respective component packaging solution. Microsoft has DLLs and EJB uses .jar.

5.1.1.3 Instance and life cycle management

This is an important issue. Indeed, the way how an object instance life cycle is managed reflects the application design.

While COM+ supports stateful components, although it does not have life cycle management support for them, it recommends not storing state in the middle tier. Microsoft and Sun have two completely different views on persistence in the middle tier. EJB supports life cycle management for stateful components (passivation and activation). This is mainly due to scalability reasons. The instance and life cycle management are closely related to scalability, which is briefly discussed in section 5.1.6.

Having the choice of where to put the state is an advantage for EJB. Introducing the container-managed bean eases the developer's task but compromises performance. With the Entity Bean, EJB takes a more object-oriented approach than COM+. The business logic is separated from the component, often into a stateless bean between the client and the Entity Bean. In COM+, the component manages the business logic and relies on ADO, or a third party product, to retrieve the data.

Instance Pooling in EJB is analogous to Object Pooling with Just In Time activation enabled. The Object Pooling in COM+ is supported only by programming languages with a supported threading model, currently C and C++. In the COM+ reference project, one critical component had to be implemented in C++, as object pooling was desired in order to decrease the response time. This contrasts with Microsoft's language heterogeneity. Just In Time activation still works as a scalable resource saving mechanism, as objects have the shortest possible life span. However, the optimal solution would be to have both JIT activation and object pooling, like in EJB.

5.1.1.4 Query language

The way the application server and the developer persist data is naturally important. Both technologies allow the developer to choose how to persist data. The most common choices are ADO for COM+, and JDBC for EJB, but the study of these two technologies is beyond the scope of this thesis.

WebLogic has their own extensions to the EJB QL language to make it more practical and user friendly. COM+ lacks the container-managed stateful component and has no use for a language such as EJB QL.

5.1.1.5 Naming or directory service

Naming is an issue that is easily overlooked, but is nonetheless fundamental in distributed system design. Names are needed to look up components, and assist the developer since knowledge of the component's physical location is no longer needed.

Both technologies have their own name and directory service, which are quite analogous. They both support the most common ways of obtaining server references: the compile time creation or the runtime downloading of proxy stubs.

5.1.1.6 Synchronization services

Thread management is difficult and confronts the developer with arduous problems. The handling of synchronization services by the application server allows the developer to focus on the business logic.

EJB has defined rules to handle synchronization for each component type. For session beans, only one client is allowed to enter an object instance concurrently. Entity Beans have transactional support, as described in section 5.1.1.7, because the Entity Bean reflects a database table. COM+ does not support this kind of component.

COM+ allows the developer to set synchronization at a component level. Each component can have a different rule for synchronization, which is left to the developer's sole discretion.

The forced serialization in EJB renders it thread-safe, but not necessarily efficient. The COM+ model leaves it up to the developer to choose the level of security, although handling threads manually is a very difficult task. Synchronization services are enabled by default in COM+.

5.1.1.7 Transaction handling

The handling of transactions is important in order to maintain consistent data. An application server abstracts this issue by providing transparent transaction services. Maintaining the integrity of the data of an enterprise system across multiple applications, users and machines is an essential and arduous task.

The transactions can be controlled declaratively or programmatically in both application servers. COM+ only supports the SERIALIZABLE isolation setting, while EJB supports all settings. While SERIALIZABLE guarantees concurrency in the database, it is curious that faster settings, such as READ_COMMITTED, are unsupported by COM+.

READ_COMMITTED is used as the default setting in the Oracle database. The EJB reference project also prefers this setting over SERIALIZABLE as

it prevents dirty-read, although it allows non-repeatable and phantom reads on the database table. The SERIALIZABLE setting waits until rows write-locked by other transactions are unlocked, which prevents the client from reading any "dirty" data. Because other transactions cannot update or delete the rows in the range, the current transaction avoids any non-repeatable reads. Because other transactions cannot insert any rows in the range, the current transaction avoids any phantoms. The SERIALIZABLE isolation level is stricter than necessary for most applications, and this represents a drawback for COM+.

Both technologies support the same transactional attributes in a similar manner.

COM+ supports nested transactions, while EJB does not. Nested transactions are particularly useful in distributed systems because child transactions may run concurrently in different servers. The lack of nested transactions makes it harder to perform complex distributed transactions, because the application design is less flexible.

5.1.2 Usability

For the developer, the usability is mostly involved with the graphical user interface and the easy maintenance of the application server configuration settings.

WebLogic uses a web interface, XML-based configuration files and XML-descriptors. More and more functions are removed from the text files, and relocated into GUI components as new versions of WebLogic are released. The text-based configuration makes generating configuration files easy, while the web interface is more intuitive and helps seeing the big picture. Still, most configuration files are textually based and not reachable from the web interface.

Microsoft is known for their intuitive user interfaces, and COM+ uses GUI actively. All configuration settings, both on the application server and the component level, can be viewed and updated graphically from the MMC. It is also possible to generate configurations.

The COM+ solution gives an excellent overview of the application and of its components in the MMC. All settings can be viewed nicely. WebLogic has much to accomplish in order to match the MMC user interface, and the web browser is currently not powerful enough to produce such an elegant interface. Additionally, using a web solution makes the server reachable from virtually anywhere. In COM+, the MMC is reachable from any remote client running Windows 2000 or Windows XP. The extensive use of XML-text files in the EJB solution can be an advantage. The developer may feel more comfortable editing text files, as developers generally work with text.

5.1.3 Performance

In the performance test conducted for the test implementation, COM+ comes out substantially better than EJB in all areas. As thoroughly discussed in section 5.4, it has probably to do with Java performance and COM+'s advantage of being closely tied to the operating system.

5.1.4 Security

Security is a key aspect in an application server and is carefully taken care of by both technologies.

EJB and COM+ both have good security models, which support the most common protocols and seem quite architecturally analogous.

5.1.5 Reliability and availability

Many current systems demand a 100% uptime and the modern application server provides solutions to achieve this goal. Clustering is the main method of increasing reliability and availability. Both technologies support clustering of the application server onto two or more servers. The clustering solutions look equally adequate on paper. Practical testing has not been conducted in respect of the scope of this study.

In order to achieve availability, also at potentially high loads on the server, support for load balancing is provided in the cluster.

As far as reliability goes, the uptime of the operating system is an important factor. It is a common conception that the Windows operating system is less reliable than other systems, but this reputation may have been earned in the early Windows versions. The authors have had some poor experience with the WebLogic server reliability, but good experience with BEA support. It is important to recall that WebLogic is only one of the many EJB implementations.

5.1.6 Scalability

Scalability is the sum of many aspects, which some depend on the application server and others on the developer.

The most obvious way of achieving scalability is through clustering, which is supported by both technologies. However, this alone does not ensure a scalable solution. Scalability is mostly a design issue, as both application servers have the necessary mechanisms to provide a scalable solution. For stateless components, object pooling is a method to maintain a low latency even with high server loads.

The load balancing service, provided by both servers, ensures that the load is shared between the servers in a cluster, assuring that no single server receives

all the requests. The load balancing service also makes the choice of the server transparent to the developer, who sees one application server. The COM+'s method of load balancing is to have, on each of the servers in the cluster, a service that reports back to the load balancing server with its current server load. The load balancing server then sends the request to the server with the least load. WebLogic, on the other hand, can choose from three different ways to load balance, but lacks what might be the most useful one – the one offered by COM+.

The stateless component scales very well, as its life span is that of the call from a client. The stateful component, on the other hand, is to be regarded as a private resource to a single client and may live as long as the client does. This poses a problem: the stateful component is monopolizing valuable resources on the server, while it may be accessed rarely. The WebLogic EJB implementation introduces some compensation to this problem by offering passivation of idle stateful beans, and it also can replicate the state of such a bean so that it works with a clustered load balanced solution. This enables some, however little, scaling to the Stateful Session Bean.

State in the middle tier should only be implemented if the system load is relatively stable. However, Stateful Session Beans have specific advantages: they are very convenient for the developer and easy to implement. As long as the application server has enough memory, and the load is stable, stateful components represent a good solution. On the other hand, in order to have potential for future growth, state should be moved out of the middle tier.

In the Rikstoto reference project, session specific state was stored both client side and in the RDBMS. To keep the RDBMS updated, a cleaning job was executed on the RDBMS every hour to delete inactive session objects. The alternative, to store the state in an object instance, would mean that one single client could hold that single instance alive as a personal resource until the client has been inactive longer than the allowed timeout value (which was 15 minutes). In addition to this, COM+ does not have mechanisms for scaling stateful components in a clustered environment and hence a solution with state in the object instance would not scale. Even though the stateful solution (with no RDBMS access) would perform substantially better on a single server, the solution would not scale and was for that reason not chosen.

EJB has some mechanisms to make stateful components scale to a certain degree, but the specification explicitly notes that the Stateless Session Bean "may scale somewhat better than Stateful Session Beans".

5.1.7 Upgradability

Both technologies support upgradability on a similar level. A computer, which is part of a cluster solution, can be brought down for maintenance and upgrade without bringing the system to a halt.

COM+ has the advantage of allowing inheritable multiple interfaces to one single component, so that the clients do not have to be updated instantly. This can also be a trap, and the developer should aim at keeping the number of interfaces for a component to a minimum. Upgrading a component can be done at runtime in COM+ as long as the interface is binary compatible.

The fact that COM+ allows for runtime upgrading of components is an advantage.

Preservation of stateful components, when taking down a server in a clustered environment, is possible in WebLogic only. When designing COM+ applications, a stateless architecture will normally be utilized. WebLogic offers an alternative stateful model.

5.2 Development qualities

This section covers how the two technologies handle the development qualities.

5.2.1 Modifiability

Modifiability has already been already discussed in the sections about upgradability (section 5.1.7) and component packaging (section 5.1.1.2).

5.2.2 Reusability

Interoperability with legacy systems and third-party components is important in order to have current systems interoperate with earlier generation and potentially huge systems. Rewriting earlier generation and fully functional systems can be time consuming, very costly and should not be necessary.

There exist bridges that allow communication across many technologies. EJB interoperate with CORBA through IIOP, and CORBA can talk to COM+ through a special CORBA-COM bridge; hence EJB can communicate with COM+. Java has a native interface to C (and C++), which eases communication with legacy systems. Both EJB and COM+ support MQ and Web Services, which can be used to interoperate.

Both technologies have support for web clients through ASP/JSP, which enables users to execute server code from any platform.

Regardless of the application server choice, communicating with other technologies is possible.

5.2.3 Portability

Portability is divided into two categories: platform and vendor. One of the EJB specifications primary concerns is: "An application may be written once,

and then deployed on any server platform that supports the Enterprise JavaBeans specification."

Since Microsoft supports only the Windows platform, and they are the only maker of COM+, the vendor and platform are provided.

Microsoft has the advantage of choosing what they believe should be the next step for the application server and simply implement it. They can see what ideas the consortium takes into consideration for the next EJB specification, but the consortium has no access to Microsoft's ideas concerning their next release, which gives Microsoft an edge. Some people believe that Microsoft has too much power on what the next step about component models should be, and disapprove of their lack of openness. The authors feel that further discussion has no place in a formal study such as this thesis.

As for EJB being portable, experience from the EJB reference project proves that developers are tempted to use the vendor-specific extensions to the EJB specification. In WebLogic, some configuration files are vendor-specific. As a result, it ties to one specific vendor, hence loosing the vendor independence. As regards WebLogic, new versions are first released for the most popular platforms, although these releases include some platform specific differences. A company often wants to contract with the vendor in order to have long term support for their product.

Changing platform and vendor every now and then seems highly unrealistic, and it may come at a high cost both in terms of time and money.

5.2.4 Buildability

The programming languages and the development tools available form the buildability properties. Buildability is important for the developer as it defines how comfortable it is to work with the application server.

Java is the most elegant programming language of the ones available to the application servers in question; it incorporates garbage collection and is a fully object-oriented language. Visual Basic has garbage collection, but is not fully object oriented. C++ is fully object oriented but the developer must handle the garbage collection manually. The two main advantages of C++ are its performance and the freedom it gives the developer. Java runs inside a virtual machine, therefore the language is not platform dependent. Performance is the price Java has to pay for its features. However, if a developer does not like Java, then staying away from EJB is a wise advice.

Java is the authors' preferred language among the available existing, but the performance is a serious drawback. A competitor to Java is the newly released C#, Microsoft's Java clone for COM+, which promises all the properties of

Java, combined with an improved performance thanks to its ties to the operating system. Also, Visual Basic.NET has become more object oriented, and is now a more modern and attractive architecture.

The primary COM+ development environment is Visual Studio, a tool that is included when Visual Basic or Visual C++ is purchased. Microsoft Visual Studio has been around for many years. Being very mature and having a rich set of features, it has grown extremely popular. So popular, in fact, that several companies have made clones of it for the Java platform. Other tools for developing COM+ components exist, but are not frequently used.

Java relies on third-party tools for their development environment: editors, Java virtual machines and debuggers. Some of these are freeware. Traditional UNIX development environment, such as 'make', are still in use, even on the Windows platform.

The rich functionality and the look-and-feel of Microsoft Visual Studio make it an excellent tool. The clones that exist for Java present the disadvantage of having Java poor performance.

5.2.5 Testability

The ability to demonstrate defects in the deployed application is significant for fault correction. To be able to debug and test applications in an application server environment is important to ensure the quality of the component.

The quality of the development tools is important for debugging and testing applications. The logging of running components can dynamically be turned on and off at runtime in both technologies, and there exist tools that take care of debugging and testing in a highly satisfactory manner.

5.3 External qualities

This section covers the qualities that did not fit into the Catalysis approach, but that the authors found relevant to include in the study.

5.3.1 Time to market

This metric measure how long it takes for the application server product to reach the market.

While EJB has to wait for the specification to be finalized before implementations can be made available, COM+ is already implemented (as discussed in section 5.2.3).

Microsoft has the application server as part of their OS; hence the application server is not released until the OS is released.

Which approach takes the least time? It is hard to say, but Microsoft, at early stages, gives developers free beta-versions of their operating system, including their application server. Such practice gives them an undeniable edge.

5.3.2 Cost of system

When choosing an application server technology, the cost, as much as the technology itself must be taken into consideration. It is an undisputed fact that Intel-based Windows servers are cheaper than UNIX servers. However, both technologies run on the Windows platform. Windows 2000 Server includes the COM+ application server, therefore when running EJB on a Windows platform, COM+ has been purchased as well, whether it is a personal preference or not.

The cost of an EJB server license from one of the major vendors is at best steep: it is about twice as expensive as the COM+ solution. The TPM-C benchmark includes price/performance metrics, and Microsoft emerges ahead in this regard. However, one could imagine running a freeware EJB implementation on Linux using Intel hardware. This would be the cheapest of all solutions, but it comes with the disadvantage of none, or expensive, support which has made companies reluctant to adopt these solutions as of now.

The experience from the reference projects clearly indicates that in order to scale the system further, it is necessary to add more hardware. This is a lot cheaper with the Microsoft reference project than with the EJB one.

5.3.3 Maturity

A system that has been around for some time is more likely to be more stable and to have fewer bugs than a newer system.

EJB is regarded as a newer technology than the COM technology, and this might be a reason why many skeptics do not dare using it yet, even though several major and successful projects are developed using EJB.

5.3.4 Simplicity

The simplicity metric measures the general easiness of the application server use. This has to do with the amount of training required in order to make a developer efficient, as well as the daily efficiency, when using the application server.

The development environment simplicity is discussed in section 5.2.4.
Visual Studio includes all necessary tools, while third-party products have to be obtained for EJB. Finding the right tools for EJB requires time but may be necessary in order to have specific needs met, whereas Visual Studio is more a general purpose tool.

COM+ presents a better design for configuration and fewer knobs to turn.

EJB has automated generation of code of an entire bean.

The reference projects gave clear indications as to the learning curve and the simplicity. In the EJB project, the developers had difficulties comprehending the full architecture and the possibilities that EJB had to offer. The COM+ project proceeded a lot more smoothly. One of the reasons is that few of the developers had prior knowledge of Java, and the project included a lot of people without prior work experience. In addition to this, finding the right development tools proved to be a time-consuming activity. The COM+ project, the first COM+ project for all participants, consisted of fewer and more experienced people. However, the simplicity of the tools and the use of Visual Basic in the COM+ project were certainly influential.

The authors believe that COM+ is easier owing to fewer development options

5.3.5 Future Plans

The future plans are important to take into account when choosing an application server platform since the technology is likely to remain for a significant period of time.

Microsoft has recently released .NET, but has not released any specific information about their future plans. .NET does not bring in a lot of new services to the COM+ application server, but includes one of Microsoft's primary focuses, namely Web Services.

The most important EJB future plans include the extension of the QL language, the introduction of other types of event management and other features already seen implemented by major vendors, such as WebLogic's implementation of read-only beans and web services.

The EJB specification has in the past held surprises close to final releases, e.g. the leap from the final draft to the final version radically changed the way how the local and the remote interfaces are used. What the future holds for COM+, only Microsoft knows, but surprises from them as well are likely to be expected. The authors assume that Web Services will make the world of components more uniform, as it allows communication of all languages on all platforms.

5.4 Performance

Clients often require short response times in order to take full advantage of a system. Additionally, users may end up not using the system, because the system is too slow. With the hardware and computational power that exist

today, an end-user expects short response times. A system has to be designed to perform for potential future high loads, as discussed in the section about scalability (see section 5.1.6).

First, in this section, a discussion of the stateless vs. stateful implementation is carried out. Secondly, the results from the test case are analyzed in respect to programmatically vs. declaratively persistence. A discussion on where to persist data is included. The two technologies respective performance and response times are directly compared in the following subsection.

5.4.1 Stateless vs. stateful implementation

Both a stateless and a stateful component, with all business logic and database code included, were implemented.

In the stateless version, the client makes a create() function-call to the server each time that the component is invoked. Additionally, state must be stored away from the middle tier, as the client is independent of the stateless object instance.

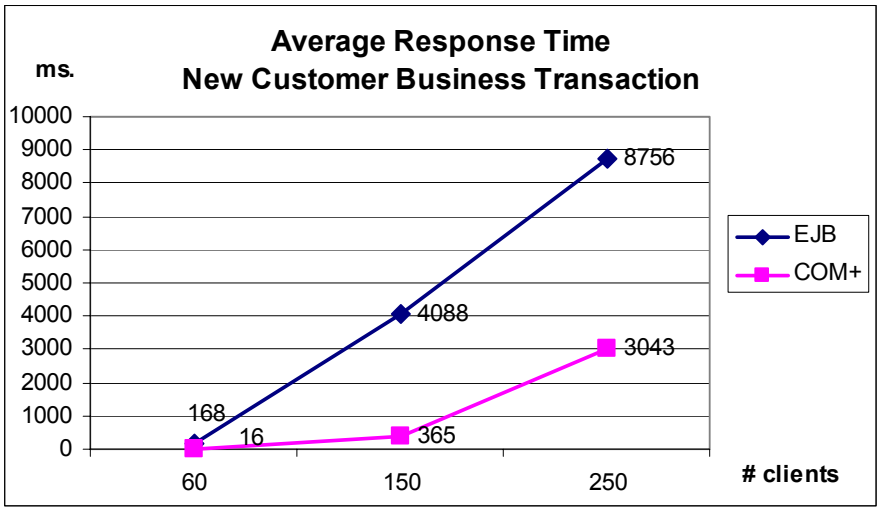

Figure 5-1: Average Response Time comparison in the New Customer business transaction

Figure 5-1 shows the average response time for both application servers running the New Customer business transaction, which is the stateless component.

For stateful components, the client has a reference to the same object instance and considers the component to be a private resource. State is, as the name implies, stored in the server component instance. Figure 5-2 indicates the COM+ and EJB average response time for the stateful component: the New Order business transaction.

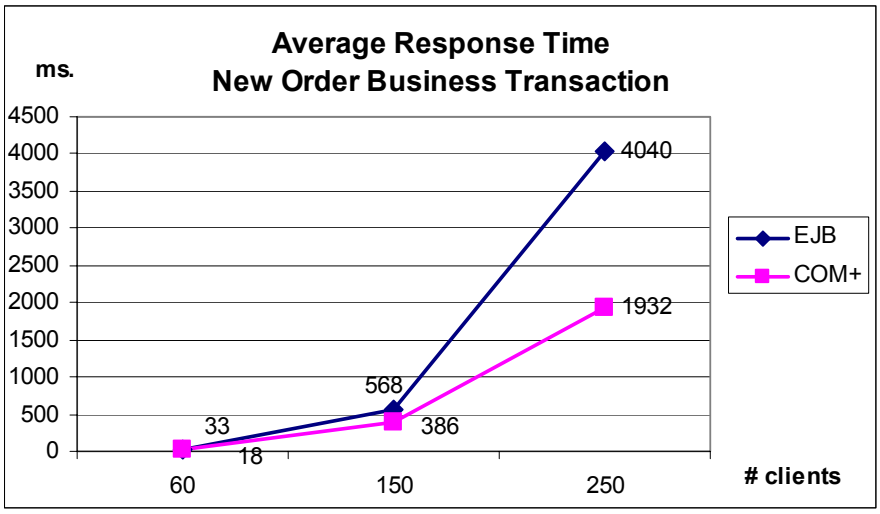

Figure 5-2: Average Response Time comparison in the New Order business transaction

5.4.1.1 COM+

In COM+, the stateless solution has a lower response time for both 60 and 150 clients, but is 40% higher when running 250 clients. Looking at the CPU graphs of the two implementations, Figure 4-4 and Figure 4-10, shows two almost identical patterns, so this would not explain these results. However, the CPU graphs indicate that when running 250 clients, the CPU load is 100% most of the time, and this explains why the stateful solution is faster at this test run. The CPU being loaded means that obtaining object instances demands unavailable resources, therefore the solution performs slower.

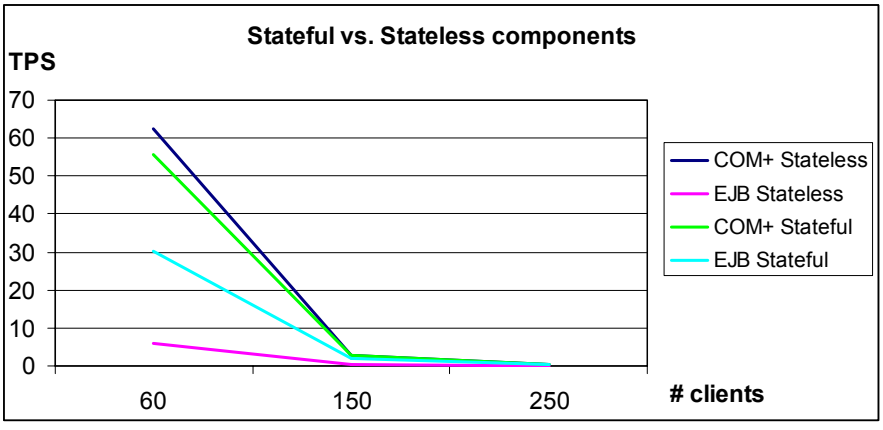

Figure 5-3: Stateful vs. Stateless components comparison

As Figure 5-3 indicates, the difference between the stateful and the stateless implementation in COM+ is a minor one when it comes to transactional throughput. Although it is impossible to notice it from the figure, Table 4-9 and Table 4-13 show that the transaction throughput, when running 250 clients, is about 66% higher for the stateful implementation. One of the reasons for this can be found in Figure 5-4: As the JIT activation is enabled for the stateless component, the components are not instantiated until needed, and the number of threads gradually increases until the CPU load reaches 100%.

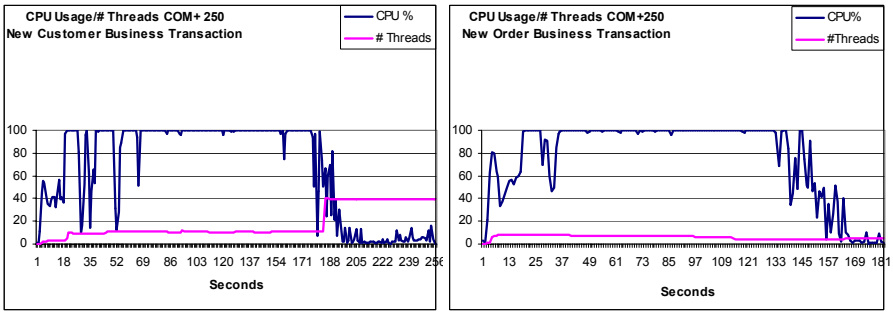

Figure 5-4: Comparison of CPU usage and thread usage for stateless (left graph) and stateful (right graph) components

The COM+ threading model evidently takes into consideration how much CPU time is available, and when the CPU load is 100%, there is no longer a need to create new threads. The objects are then put on hold in an execution queue, and wait for a thread to become available. Notice that for the stateless component, as soon as the CPU drops from 100% and after about 180 seconds, the number of threads almost triples. As for the stateful component, because clients hold the reference to the component instance until it terminates, the variation of threads is a lot less. The instances are created at the beginning of every client's execution and the number of threads decreases as the clients terminate.

5.4.1.2 EJB

Comparing the results indicates that the stateful solution has a shorter response time for all test runs in EJB. It is partly explained by the fact that the stateful component is already instantiated, since the client does not have to request a new instance on the server. Looking at Figure 5-5, the difference in transactional throughput, measured in percentage, between the stateful implementation and the stateless implementation is approximately 50% higher at 150 clients than when running 60 clients. Looking at the CPU usage in the two different test runs, Figure 4-5 and Figure 4-11, gives a reasonable explanation for this. The CPU is running at 100% for the stateless implementation when running 150 clients, but is more moderately loaded in the stateful implementation. The difference in CPU load and the fact that the

stateless solution has to obtain object instances for every invocation explains these results.

Because the EJB application server does not have as strong ties to the operating system as COM+, the way how it utilizes threads does not show on the measurements. Therefore, it is unknown whether the threading model of EJB has anything to do with the results or not.

An interesting aspect of Figure 5-3 is the difference between stateful and stateless components in EJB is as high as 400% when it comes to transactional throughput. In the COM+ solution, the difference is only 10%.

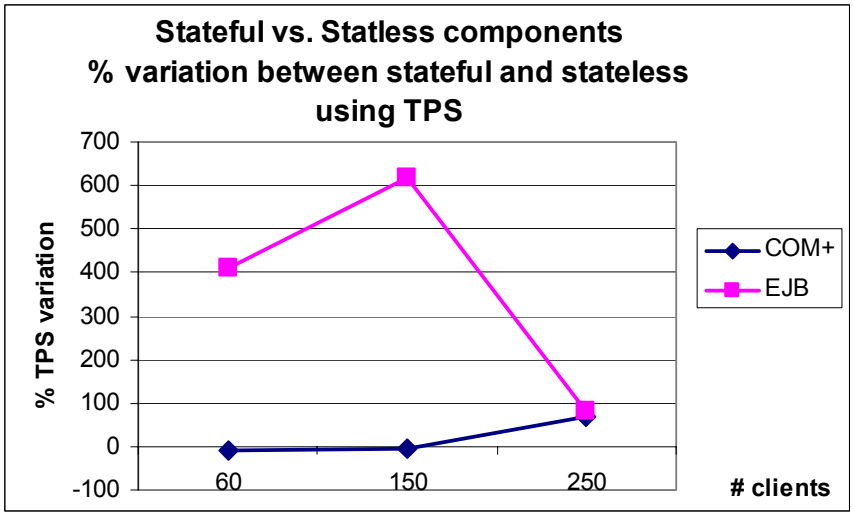

Figure 5-5: % TPS variation between stateful and stateless components in EJB and COM+

Many designers choose the stateless solution over the stateful solution, due to scalability issues. EJB has introduced mechanisms to scale stateful components. If the server CPU load is at 100%, the stateful component clearly performs better than the stateless one.

5.4.2 Persistence in the middle tier

The ability to have container-managed persistence in the middle tier is probably the single biggest difference between the two technologies. EJB supports it in their architecture, a field-to-field mapping from the component to the database. In the test implementation, this feature is emulated in COM+ by manually doing the required database operations.

Only minor differences can be observed over the stateless implementation in the New Customer business transaction, with the exception of the run with 250 clients. The results of the New Customer business transaction reflect

longer ART and higher CPU load than in the other business transactions. The "persistent" COM+ component utilizes a stateless component as a proxy between the client and itself. The results are better for the Let's Buy Some Records business transaction when running 250 clients; in the collected data no explanation for this was found.

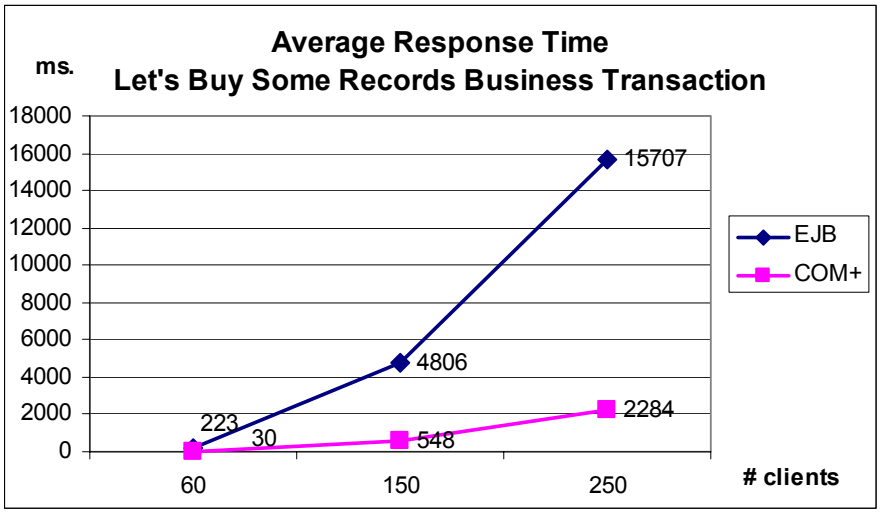

Figure 5-6: Average Response Time comparison for the "Let's Buy Some Records" business transaction

Comparing the ART results between EJB (see Figure 5-6) with the stateless component (see Figure 5-1) shows that the ART is longer and the CPU load higher. The increase in both the ART and the CPU load is about 50% in average on all test runs and must be considered as a substantial increase. Microsoft claims that this kind of component does not scale, while BEA and the rest of the EJB community claims it does. It has not been tested in this implementation, owing to the lack of clustering. The performance would be improved if the developer implemented the database logic manually, however this would make maintenance harder.

Many designers, including the designers of the EJB reference project, prefer the container-managed persistence model as it eases the developer's job and is a better object-oriented design. This comes at the cost of performance. However, additional hardware compensates the performance loss compared to having to code it manually and, therefore, can represent a reasonable solution.

5.4.3 COM+ vs. EJB performance

For the stateless component (New Customer business transaction), the COM+ server average response time is about ten times faster than EJB on low load (see Figure 5-1). The CPU load is somewhat higher for EJB, and, as

a combined result of these metrics, it also takes more time to complete the task.

The CPU usage for the stateful component is far lower for EJB than for COM+. However, the average response time is shorter and the time that it takes to complete the task is shorter for COM+ than for EJB. This is mainly the result of Java performance and the overhead of the virtual machine.

WebLogic is implemented in Java, and therefore runs inside a JVM. This indicates that there are more "layers" involved in this technology than there are for COM+ with its close ties to the Windows OS. Generally, Java is not a strong performer.

Figure 5-7 shows that the ART is reduced from 150 to 250 clients. This may be caused by instance lookup algorithms, e.g. using hash tables with binary trees where a balanced tree is the most efficient. It would be pure speculation to assert that the instance lookup algorithms are the cause of the decrease in ART from 150 to 250 clients.

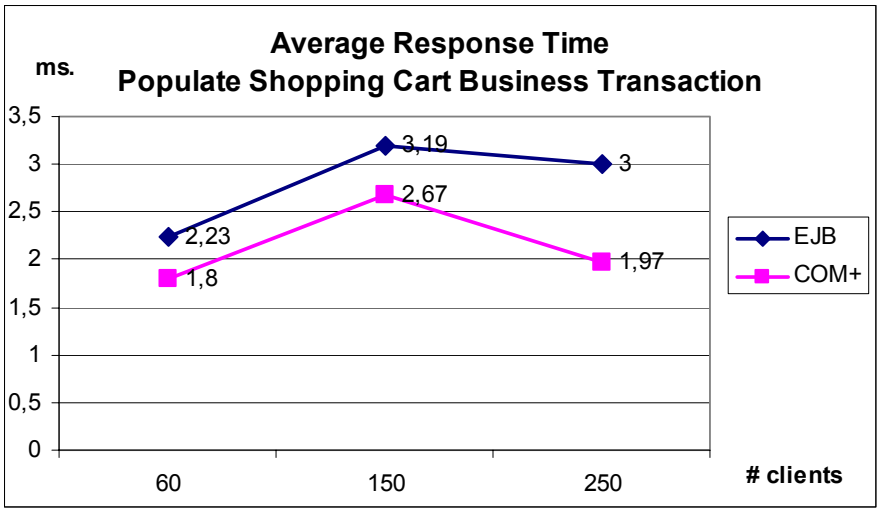

Figure 5-7: Average Response Time comparison Populate Shopping Cart business transaction.

The comparison between the container-managed persistent bean in EJB and the emulated "persistent" component in COM+ is questionable. Indeed, as the EJB container manages the persistence, the overhead naturally increases. The comparison can describe the performance advantages of handling persistence manually. Figure 5-6 indicates a considerable difference in the average response time of the two technologies; again COM+ emerges ahead.

Comparing the results of the Populate Shopping Cart business transaction, which is the only business transaction that does not use database storage, with

the other three business transactions, it is clear that the performance of EJB solutions is more heavily affected than the one of COM+ when access to database is introduced. The reason for this difference is unclear, but this could be related to the choice of a Microsoft RDBMS, the database drivers and the extra overhead that EJB has when performing tasks. This has not been closely examined owing to lack of data and resources.

All findings in the test implementation point to the same observation: COM+ performs better than EJB. It is important to remember that both servers were run using primarily the default configuration, and only tuned so that they would match each other in particular cases. The choice of OS and database also affects the results. The fact that COM+ is closely tied to the OS and that Java is a slow performing language would place COM+ ahead as far as the performance is concerned, regardless of tuning. The TPC-C benchmark results discussed in section 4.4.2 support these findings.

Chapter 6

EVALUATION

In this chapter, the author's estimate their work and look at what could have been done differently and what they believe was done correctly. A comparison with results of existing work is also conducted to evaluate how the findings presented in this thesis may differ from those of others. Finally, a look at the common conceptions regarding the two technologies is given.

6.1 Right approach?

The Catalysis approach that has been used throughout this thesis proved to be a valuable and lucid approach. It helped keep the different qualities of the technologies apart, and provided a natural mapping of the traditional functional and non-functional requirements.

The test implementation supplied important results so that a better comparison of the performance could be done. Additionally, it is easier to understand a technology that one has implemented in a project, than a technology that one has just read about. To have a reference project for each technology proved to be valuable in terms of real life experience and examples of what can go wrong.

During the research work for this thesis, it became manifest that clustering is important in order to obtain good results concerning scalability. Clustering should have been included to make the results of the thesis more complete. The available hardware did not permit clustering to be part of this study, and including clustering would have made this thesis too extensive.

In order to increase the reliability of the results of the performance comparison, at least some tuning of the servers could have been done. The tuning of application servers is actually a sophisticated task that some people perform for a living, which illustrates how complicated it can be.

The approach was satisfactory. However some delimited parts (see section 1.3) could have been included, although they would have made this thesis too extensive. Especially clustering and load balancing would give important results for scalability issues.

6.2 Comparison with existing work

Articles that compare versions of these two technologies head-to-head are available and have been used as a resource in the planning of this thesis. These articles are all based on specifications and white papers produced by SUN and Microsoft, and the persons who wrote these articles seemed biased.

This study, unlike the mentioned articles, bases its performance comparison on an actual implementation conducted in a similar environment. Even though the application servers were not tuned, the results reflect the reality.

The results in these articles, however different they may look, are actually quite similar to the ones presented in this thesis. The importance of the differences found between EJB and COM+ is open for discussion, and in this lays the main difference between the articles and this study.

That the results from the articles and this thesis are so similar is not all that surprising since they are facts for the most part.

6.3 Common conceptions

Whether or not the hypotheses are verified is summarized in this subsection.

The common conceptions stated in section 2.7 are:

> H1: COM+ and EJB should have identical (linear) performance curves, although EJB should be slower because of the Java overhead. COM+ has stronger ties to the operating system; hence it should offer shorter response times. In addition Java is considered slow.
>
> H2: EJB presents more opportunities for the developer, but is more arduous to learn properly.
>
> H3: COM+ is less reliable than EJB, because of the history of instability of the Windows operating system.

The first part of the first hypothesis is clearly falsified. The performance curves show that EJB performs gradually worse compared to COM+ as the number of clients increases. One of the reasons is that EJB is more CPU intense than COM+. The second part of the first hypothesis is verified, although if COM+'s strong ties to the operating system is the cause for it cannot be stated. COM+ had shorter ART for all tests conducted.

The second hypothesis concerns the learning curve of EJB and that EJB has more features than COM+. As EJB seems more difficult to learn, and it certainly has more features than COM+, this hypothesis is verified.

The third and final hypothesis concerns the reliability of COM+, and it claims that COM+ is less stable than EJB. According to the authors' experience, the reality is actually quite opposite, and this hypothesis is falsified.

Chapter 7

CONCLUSION

Presented in the thesis are findings of an attempt to objectively evaluate the COM+ and EJB component models.

Both models present all the necessary properties and qualities required of a component model, and hence this is not an issue in choosing technology. They are both mature enough, but the authors do not like the fact that WebLogic has a tendency to crash, even with final releases. However, this is mainly a WebLogic issue and not an EJB issue.

In the performance test, COM+ performs drastically better than EJB on all types of tests. This might be due to a non-tuned EJB server, but it is hard to explain such huge gaps.

EJB achieves platform and vendor independence, meaning that one will probably remain tied to one platform and one vendor in the end owing to vendor specific extensions.

Interoperability is covered in both technologies. However, EJB has strong ties to CORBA which eases the communication with CORBA components. As both technologies now support Web Services, interoperability should become a less cumbersome issue in the years to come.

In the case of EJB, a strong inclination for Java is preferred, if not necessary. COM+ gives the choice between several programming languages, and offers transparency while using them. However, in order to utilize the object pooling mechanism for COM+, Visual Basic can not be used because of its simple threading model.

The transactional support in EJB has the advantage of supporting all isolation levels, while COM+ only supports the strictest isolation which compromises performance. However, COM+ supports nested transactions, EJB does not.

As for development tools, Microsoft provides Visual Studio, which therefore will be most likely what the developer will eventually use. Finding a tool for EJB is more time consuming.

The external qualities of the application servers are important:: time to market, cost of system, maturity, simplicity and future plans. External qualities are Microsoft's prime domain. They particularly excel in cost of system and in simplicity. EJB offers more in terms of properties, by providing choice of

persistence in the middle tier and life cycle management of stateful components.

The bottom line is to know what qualities are considered important, since the application servers are quite analogous, more analogous than both communities like to admit!

Chapter 8

FUTURE WORK

While planning this thesis, one of the challenges was delimitation. Originally, the thesis was meant to be two separate theses, about performance and properties respectively. Because of the complexity and extent of the application server technology, the delimitation was made in order to keep the amount of work at a reasonable level. Event-driven communication, security and scalability are all services of component models that could serve as a basis for further research.

As the research advanced, the authors found delimitated aspects that should have been included (clustering) and aspects that could be further investigated. The following subsections describe the most important areas in which to expand the subject of this thesis.

8.1 .NET vs. J2EE, on a more extensive and higher level

This study compares the component models of the two major distributed technologies. Several recent papers are available in comparing the .NET and J2EE platforms, however these are again biased. By comparing on a more superior level, a wider understanding of the advantages and disadvantages is undeniably gained.

8.2 Portability

EJB claims to be portable across platforms and vendors. Porting a WebLogic implementation to Websphere, JBOSS or another application server that follows the EJB 2.0 specification would test the actual portability of EJB.

8.3 Performance - unbiased test with tuning

The performance test was conducted with default configurations on one set of hardware. Only one EJB implementation was considered, and SQL Server became the only RDBMS. Further research and testing could be done with respect to tuning and multiple choices of software and/or hardware. Running the EJB on the same hardware, utilizing another OS, e.g. Linux, would be interesting to determine whether the choice of the platform matters.

Tuning the application servers could drastically modify results presented in this thesis. Furthermore, it would be interesting to determine whether the choice of the RDBMS or EJB implementation matters. If the results found after optimal tuning remained analogous to the results in this thesis (far better COM+ performance), detailed logging and analyzing on several levels could be carried on to determine the cause of a gap in performance.

8.4 CORBA 3

Similar study can be conducted including the CORBA 3 CCM as an actor in the comparison. The first implementations are now available; however, the industry has not yet shown great interest for this technology.

CORBA could also be compared directly to .NET and J2EE, as in section 8.1.

8.5 Scalability

Further research can be conducted on clustering and load balancing. The consequences of the design and possible development disadvantages when using a clustered solution could be investigated.

Further research could be done on the different load-balancing algorithms to discern which one works better under different circumstances. Additionally, a measurement on how well the different component types scaled in a clustered environment could be done.

8.6 Life cycle cost of project

What technology gives the least expensive total project cost? Estimation of the life cycle cost of a project; including hardware, platform/project software, and development.

8.7 Inter-platform communication

Communicating cross platform with Web Services, bridges between and message queuing would give an interesting approach to research on interoperability.

BIBLIOGRAPHY

[ASU99]
Bharat Gogia , Max P. Grasso, Hoa Nguyen
Application Servers Unmasked, Oct 99 issue
http://www.adtmag.com/Pub/oct99/fe991001a.htm
ADTMag.com

[BEA 01a]
BEA WebLogic 6.1 Documentation, Tuning WebLogic Server.
http://e-docs.sbea.com/wls/docs61/perrform/WLSTuning.html
BEA, 2001.

[BEA 01b]
BEA WebLogic 6.1 Documentation
http://e-docs.bea.com/wls/docs61
BEA, 2001

[BEP99]
K. Brown, P. Eskelin, N. Pryce
A Mini-Pattern Language for Distributed Component Design.
PLOP 1999 Conference. Pattern Languages of Programs, August 1999.

[BLAIR97]
Gordon S. Blair, Jean-Bernard Stefani
Open Distributed Processing and Multimedia.
Addison-Wesley, 1997

[BRJ99a]
Booch, Grady, Rumbaugh, James and Jacobson, Ivar.
The unified modeling language user guide.
Addisson-Wesley, 1999

[BRJ99b]
Grady Booch, James Rumbaugh, Ivar Jacobson.
The unified software development process.
Addisson-Wesley, 1999

[CATTELL00]
Interview with Rick Cattell
JavaLive, 2000

[COM95]
The Component Object Model Specification
Microsoft Corporation, 1995

[CORBA3]
The CORBA 3 specification.
http://www.omg.net
OMG, 2001

[CORBA97]
"The Common Object Request Broker: Architecture and Specification, Version 2.1"
OMG, 1997

[COULOURIS01]
G. Coulouris, J. Dollimore, T. Kindberg
Distributed Systems, Concepts and Design.
Addison-Wesley, 2001

[DCOM98]
N. Brown, C. Kindel
Distributed Component Object Model Protocol.
Microsoft Corporation, 1998

[DMENTOR]
DevelopMentor's home page
http://www.developmentor.com/

[DW99]
Desmond Francis D'Souza, Alan Cameron Wills
Objects, Components, and Frameworks with UML. The catalysis approach.
Addison-Wesley, 1999

[EJB2.0]
Enterprise Java Beans 2.0 specification
http://java.sun.com/products/ejb/docs.hhtml
Sun Microsystems, 2001

[FOW01]
M. Fowler
Information System Architecture.
http://martinfowler.com/isa/index.html
Martin Fowler, 2001

[GARTNER]
The Gartner Group home page
http://www.gartnergroup.com

[Genera AS]
A company in the Software Innovation group.
http://www.genera.no

[GHJV95]
E. Gamma, R. Helm, R. Johnson, H. Vlissides
Design Patterns: Elements of Reusable Object-oriented Software.
Addison-Wesley, 1995

[GWE01]
Michael Girdley, Rob Woollen, Sandra L. Emerson
J2EE applications and BEA WebLogic Server.
Prentice Hall, 2001.

[GZ00]
Paco Gómez, Peter Zadrozny
Java 2 Enterprise Edition with BEA WebLogic Server.
Wrox press ltd, 2000

[J2EE]
Java 2 Enterprise Edition specification version 1.3
http://java.sun.com/j2ee/download.html#platformspec
Sun Microsystems, 2000

[JBOSS]
World class J2EE technology in open source
http://www.jboss.org

[JINI]
JINI Network Technology.
http://www.sun.com/jini/
Sun Microsystems

[JMS]
Java Messaging Services.
http://developer.java.sun.com/developer/technicalArticles/Networking/messaging/
Sun Microsystems, 2000

[KARMIRA]
Bugseeker for Java 2
http://www.karmira.com

[LOADRUNNER]
Load testing tool
http://www-svca.mercuryinteractive.com/products/loadrunner/
Mercury Interactive

[MICROSOFT]
http://www.microsoft.com

[MICROSOFT98]
Comparing Microsoft Transaction Server to Enterprise Java Beans.
http://www.microsoft.com/com/wpaper/mts-ejb.asp
Microsoft, 1998

[MIRABILIS]
ICQ Messenger Service home page
http://web.icq.com/
Mirabilis

[MS .NET]
Homepage for Microsoft .NET
http://www.microsoft.com/net/default.asp
Microsoft

[MSDN CIS]
MSDN
COM Internet Services
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndcom/html/cis.asp
Microsoft, 1999

[MSDN CRM]
COM+ Documentation Team
Compensating Resource Manager
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cossdk/htm/pgcrmcpt_9yr6.asp
Microsoft, 2001

[NSB]
Norges Statsbaner, the Norwegian railway company
http://www.nsb.no

[NTP]
Network Time Protocol
http://www.eecis.udel.edu/~ntp/

[OMG]
Object Management Group
http://www.omg.net

[OMG99]
EJB, CORBA and Application Servers Press Kit
http://cgi.omg.org/news/pr99.html
OMG, 1999

[PLATT00]
David S. Platt
http://msdn.microsoft.com/library/en-us/dnmag00/html/dnmag00/html/ComTips.asp?frame=true
Rolling Thunder Computing Inc., 2000

[PLATT99]
David S. Platt
Understanding COM+.
Microsoft Press, 1999

[RIKSTOTO]
Portal for horse-betting on the Internet.
http://www.rikstoto.no

[RMI 97]
Java Remote Method Invocation
http://java.sun.com/products/jdk/rmi/index.html
Sun Microsystems, 1997

[IBM]
Eliezer Dekel
http://www.haifa.il.ibm.com/Technion/beansp6.htm
IBM, 1997

[RO99]
Ed Roman, Rickard Oberg
The Technical Benefits of EJB and J2EE Technologies over COM+ and Windows DNA.
The Middleware Company, 1999

[ROMAN99]
Ed Roman
Mastering Enterprise Java Beans and the Java 2 Platform Enterprise Edition.
Wiley, 1999

[RS99]
Ed Roman, Roger Sessions.
EJB vs. COM+. Debate at Austin Foundation for Object Oriented Technology
http://www.objectwatch.com/eddebate.htm
AFOOT, 1999

[SERVERSIDE]
J2EE community
http://www.theserverside.com

[SESSIONS00]
Roger Sessions
COM+ and the Battle for the Middle Tier.
Wiley, 2000

[SESSIONS01]
Roger Sessions
Java 2 Enterprise Edition (J2EE) versus The .NET Platform Two Visions for eBusiness
http://www.objectwatch.com
Objectwatch, 2001

[SIEGEL]
Jon Siegel
CORBA 3 Fundamentals and Programming, Second Ed.
Wiley, 2000

[SITRAKA]
JProbe Performance Tuning Solutions
http://www.sitraka.com/software/jprobe/
Sitraka

[SOAP 99]
Simple Object Access Protocol (SOAP) 1.1 specification
http://www.w3.org/TR/SOAP/
Microsoft, et. al.

[SQL92]
SQL specification draft
http://www-2.cs.cmu.edu/afs/andrew.cmu.edu/usr/shadow/www/sql/sql1992.txt
Digital Equipment Corporation, 1992

[SUN]
Sun Microsystems, the creators of Java.
http://www.sun.com
http://java.sun.com
Sun Microsystems

[THOMAS98]
Anne Thomas
Comparing MTS and EJB.
Distributed Computing, 1998

[TPC]
Transaction Processing Performance Council.
http://www.tpc.org

[TUXEDO]
BEA Tuxedo product overview
http://www.it.iitb.ernet.in/~suresh/tuxedo/prodov/oview.htm
BEA

[WEBGAIN]
WebGain | Accelerating e-Business
http://www.webgain.com

[WEBLOGIC]
BEA WebLogic application server
http://www.weblogic.com
BEA

[WHATIS]
Definitions for thousands of the most current IT-related words.
http://www.whatis.com

[XML 98]
eXtensible Markup Language (XML)
http://www.w3.org/XML/

Appendix 1

HARDWARE

This appendix presents the hardware used for conducting the test on the performance. All hardware was connected using an Intel 10/100 Mbit switch, and CAT-5 TP cables.

The hardware used is as follows:

Server:

CPU:	1,4 GHz AMD Thunderbird
RAM:	512 Mb DDR Ram
HD:	60 Gb IBM 7200 rpm
OS:	Windows 2000 Server
Net:	3Com 3c905tx 10/100
Database	SQLServer 2000 with no alterations made

Test machine 1:

CPU:	500 MHz Intel Pentium III
RAM:	256 Mb SDRam
HD:	20 Gb IBM 4800 rpm
OS:	Windows 2000 Professional
Net:	3Com 3c905tx 10/100

Test machine 2:

CPU:	750 MHz Intel Pentium III
RAM:	256 Mb SDRam
HD:	20 Gb IBM 4800 rpm
OS:	Windows 2000 Server
Net:	3Com 3c905tx 10/100

Test machine 3:

CPU:	500 MHz Intel Pentium III
RAM:	512 Mb SDRam
HD:	40 Gb IBM 7200 rpm
OS:	Windows 2000 Professional
Net:	3com 3c905tx 10/100

Appendix 2

SAMPLE LOGFILE

```
<xxx.xxx.xxx.xxx><010> --- ------------------------------------------------
--------------------
<xxx.xxx.xxx.xxx><010> --- (TST) Total Successful Full Transactions
<xxx.xxx.xxx.xxx><010> --- (TPT) Total Processing Time (ms)
<xxx.xxx.xxx.xxx><010> --- (ART) Average Response Time (ms)
<xxx.xxx.xxx.xxx><010> --- (TPS) Transactions per Second
<xxx.xxx.xxx.xxx><010> --- Total Unsuccessful Transactions      : 0
<xxx.xxx.xxx.xxx><010> --- TST TPT   ART    TPS
<xxx.xxx.xxx.xxx><010> --- 5   139  27,8   35,9712230215827                   findCustomer()
<xxx.xxx.xxx.xxx><010> --- 5   15   3      333,333333333333                   setCustomer()
<xxx.xxx.xxx.xxx><010> --- 5   9    1,8    555,555555555556                 createCustomer()
<xxx.xxx.xxx.xxx><010> --- 5   183  36,6   27,3224043715847                   findCustomer()
<xxx.xxx.xxx.xxx><010> --- ------------------------------------------------
----------------------
<xxx.xxx.xxx.xxx><010> --- 20  346  17,3   0,238045629070514
<xxx.xxx.xxx.xxx><010> ---
=====================================================================
<xxx.xxx.xxx.xxx><010> --- TOTAL:;
5;139;27,8;35,9712230215827;5;15;3;333,333333333333;5;9;1,8;555,55555555555
6;5;183;36,6;27,3224043715847;20;346;17,3;0,238045629070514
```

Appendix 3

EJB VS. COM+ SUPERIOR COMPARISON TABLE

	DCOM/COM+	Enterprise Java Beans	CORBA
Application Server	Microsoft Transaction Server (MTS) COM+ is a	Websphere, Weblogic, Oracle	CORBA 3 application server.
Openness	Binary standards	Open standards, but SUN has veto	Open standards
Language support	Java, C++, C, Visual Basic, Delphi, PowerBuilder	Java. Bindings to C/C++ exist.	ADA, C, C++, COBOL, Java to IDL, IDL to JAVA, Smalltalk, Eiffel.
Web server	All that support ASP, most used is Internet Information Server (IIS)	All that support servlets	All web servers that support the language used
Platform support	DCOM: All 32-bit Windows versions COM+: Windows 2000 / Windows XP	There are implementations for most platforms	There are implementations for most platforms
Protocol support	DCOM	RMI over JRMP or IIOP	IIOP, SOCKS

	DCOM/COM+	Enterprise Java Beans	CORBA
Persistent object references	Have transient objects, but can have persistent objects programmatically	Persistent objects with persistent object references	Persistent objects that have persistent object references
CORBA-compatibility	COM and CORBA can access each other through bridges, but not with full functionality	Most EJB servers support access of EJB components from CORBA clients. It is possible because Java has a language binding to IDL.	100%

Appendix 4

SOURCE CODE COMPARISON

With precompiled proxy stubs on the client, the code for invoking a server components is as follows:

```
// Create new reference to server object

RecordServices rs = (RecordServices)
Naming.lookup("no.henrik.ejb.RecordServices");

// Invoke server object instance
rs.findAllCustomers();
```

```
'Create new reference to server object

Dim rs As RecordServices
rs = new RecordServices

'Invoke server object instance
rs.findAllCustomers()
```

The following example shows database code for the findAllCustomers() method.
The main difference is that the EJB code obtains the sqlsource from the application server.

```
// Obtain application server context
InitialContext initCtx = new InitialContext();

// Get application server sqlsource
DataSource ds = (javax.sql.DataSource)
              initCtx.lookup("SQLSource");

// Get database connection
Connection con = ds.getConnection();

// SQL-statement
PreparedStatement ps  =
con.prepareStatement("select Name, Address,
Email, Telephon, Information, Id from
Customer");

// Execute query
ps.executeQuery();

// Obtain resultset
ResultSet rs = ps.getResultSet();
```

```
// Declaration of variables
Dim myRecSet As adodb.Recordset
Dim myConnection As New adodb.Connection



// Open database connection
    myConnection.Open "Data
source=ejbvscomplus01;User ID=sa;Password=sa;"

// Execute SQL-query, and obtain resultset
Set myRecSet = myConnection.Execute("select *
from Customer")
```

The next example shows QL-database code for the findRecord(String title) method. The EJB-QL is from the ejb-jar.xml file included with the RecordBean. The database connectivity is managed by the container. The VB example is similar to the one above.

```
<query>
<query-method>
<method-name>findRecord</method-name>
<method-params>
<method-param>double</method-params>
</method-params>
<ejb-ql>
   <![CDATA[SELECT OBJECT(a) FROM MyRecordBean
AS a WHERE a.title = ?1]]>
</ejb-ql>
</query-method>
</query>
```

```
// Declaration of variables
Dim myRecSet As adodb.Recordset
Dim myConnection As New adodb.Connection



// Open database connection
    myConnection.Open "Data
source=ejbvscomplus01;User ID=sa;Password=sa;"

// Execute SQL-query, and obtain resultset
Set myRecSet = myConnection.Execute("select *
from Record where title = '" + title + "'")
```

I N D E X

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X