SCHOOL OF INFORMATION TECHNOLOGY

MASTERS IN INFORMATION TECHNOLOGY

Individual Assignment Cover Page

Name of Student	**Pitso Tsibolane**
Student Number	**24256162**
Name of Module	**Corporate IT Systems**
Module Code	**MIT 853**
Name of Lecturer	**Prof. AP Engelbrecht**
Date of Submission	**06 September 2004**
Declaration: *I declare that this assignment, submitted by me, is my own work and that I have referenced all the sources that I have used.*	
Signature of Student	

Date received	
Signature of administrator	

Mark	
Date	
Signature of Lecturer	

MIT 853

THE ROLE OF MIDDLEWARE IN ENTERPRISE APPLICATION INTEGRATION

06 SEPTEMBER 2004

ABSTRACT

Middleware fulfills a crucial role in opening up enterprise Information System environments to new access channels and application systems integration. This paper discusses the origins of the concept of middleware and the IT problem that led to the introduction of the concept. Furthermore, a discussion of various key middleware technologies and their usage is undertaken. The paper asserts that middleware has a primary role of enterprise application integration and the apparent convergence of middleware technologies will benefit distributed computing world.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

There is currently a marked rise in the interest of the subject of middleware in the IT world. Growth in the middleware market was projected to increase by an incredible 438% between 1998 and 2003 [IDC, 1999]. This represents a growth from $2.2 billion to $11.6 billion in this period. However, the concept of middleware remains very misunderstood and misapplied by the IT world in general. This is due to the fact that the middleware concept is a very technical subject in its nature which has grown so rapidly since its introduction.

Furthermore, the discussion around middleware is often over-complicated by the fact that middleware technologies often invoke passionate support from "techies" from various support camps in newsgroups and seminars alike. Perhaps the most notable is the conflicting views from the Java and Microsoft protagonists. Also adding to the over-complication of this subject is the fact that various middleware vendors keep renaming their products with each new version they release [Bye, 2003].

This paper introduces the concept of middleware, states the need for this technology and discusses various middleware technologies on offer in a way that is not aligned to a specific technology vendor. Furthermore, the discussion is aimed at business and technology people who want to understand the discussion and make decisions based on an impartial point of view.

2. THE ORIGINS OF MIDDLEWARE

The first usage of the word "Middleware" can be dated to as early as 1970 at the British European Airways IT group [Bye, 2003]. The group developed a software suite to make the development of transactional applications easier. The term was also used in the late 1980's to describe network connection management software [Bakken, 2002]. However, it was only in the mid-1990 when the network technology had stabilized that the usage of the term became widespread [Bernstein, 1996]. Even then, the concepts similar to today's middleware went under names of network operating systems, distributed operating systems and distributed computing environments.

The first major distributed object middleware system was named Cronus. Its contemporaries were known as Clouds and Eden [Bakken, 2002]. In 1982, Birrell and Nelson developed RPC (Remote Procedure Call). RPC systems that were widely used then were the Sun's Open Network Computing (ONC) and Apollo's Network Computing System (NCS). However the first middleware framework to provide general purpose and extensible quality of service for distributed objects was Quality Objects (QuO) [Zincky et al, 1997].

In 1998 the first major Common Object Request Broker Architecture (CORBA) system to provide quality of service, namely real-time performance directly in the ORB (Object Request Broker) was developed. It was named TAO[Schmidt et al, 1998]. A year later, 1989, the Object Management Group (OMG), which is comprised of a number of organizations with a collective purpose of standardizing object and component technology, was formed. It presently remains the largest industry consortium of its kind.

In 1993, the Message Oriented Middleware Association (MOMA) was formed. This led to the wide usage of the MOM middleware technology by the late 1990's. In the late 1990's the advent of the internet and browsers led to Hyper-Text Transfer Protocol (HTTP) becoming a major building block for various kinds of middleware due to its pervasive deployment and ability to get through most firewalls [Bakken, 2002].

3. THE NEED FOR MIDDLEWARE

The rapid technology developments over the last decade and currently has made networking systems together easier. This has made new ways of working possible and has directly changed the economics of various traditional methods of doing business. This has led to an increase in the need to adapt quickly to a more challenging business climate through quick and quality service and through cost savings. The growth of electronic business has opened options to many enterprises and more businesses are increasingly becoming reliant on IT to operate.

Thus an effective IT framework that allows businesses to respond to new pressures of change is vital. Electronic businesses require the ability to do the following [Bye, 2003]:

- Add new application services rapidly
- Deliver this applications through expanding variety of access channels
- Seamless integration between these applications as they could be distributed across different organisations and connected through private networks or across the Internet e.g. Web services.

A typical IT landscape of most organisations comprises of various critical systems developed over time. These legacy systems are often implemented using

different standards or no open standards at all. New systems being introduced are also not guaranteed to conform to the same standards. Thus the problem of heterogeneity persists.

To address the problem of heterogeneity companies have opted to create work-a-rounds like [Bye, 2003]:

- Discard legacy systems and rewrite those using latest technologies and standards. However this can be costly.
- Some organisations have decided re-use existing legacy applications with new ones to increase capability. This is a low risk approach however this approach can be ad-hoc and only increase complexity in future.

Thus enterprises realized the need for a well-organised IT infrastructure that supports the addition of new channels and applications in such a way that they can be integrated with the existing legacy systems in a controlled manner. This infrastructure should also allow progressive replacement of obsolete applications. Middleware is the key to building this infrastructure.

4. CLASSIFICATION OF MIDDLEWARE TECHNOLOGIES

Bakken (2002) describes middleware as a class of software technologies designed to help manage the complexity and heterogeneity inherent in distributed systems. Middleware can also be defined as a layer of software above the operating system but below the application program that provides a common programming abstraction across a distributed system, as shown in Figure 1.

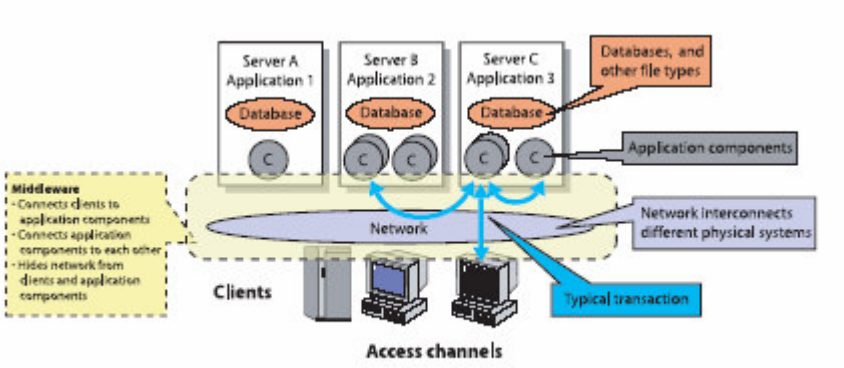

Figure 1. Middleware, component and distributed systems [Bye, 2003]

Modern middleware applications have characteristics that include [Campbell et al, 1999]:

- Masking heterogeneity in the underlying infrastructure by cloaking system specifics;
- Permitting heterogeneity at the application level by allowing the various components of the distributed application to be written in any suitable language;
- Providing structure for distributed components by adopting object-oriented principles;
- Offering invisible, behind-the-scenes distribution as well as the ability to know what's happening behind the scenes
- Providing general-purpose distributed services that aid application development and deployment.

A strict classification of middleware technologies is a challenging exercise as the categories are blurred in the market-place. Many middleware product vendors evolve their products rapidly to catch up with competing vendors which is resulting in a convergence of middleware technologies. For instance, many products have begun offering APIs (Application Programming Interface) for distributed objects and message queues managed in part by a Transaction Processing Monitor (TPM). TPMs in turn use RPC or MOM as an underlying transport while adding management and control facilities. On the other hand Relational Database vendors have been breaking the relational model of a strict separation of data and code by including RPC-like stored procedures [Bakken, 2003]. To complicate the matter further Java is being used to program these stored procedures.

This paper combines three classifications by several authors [Bakken (2003), Bye (2003), Campbell et al (1999)]. The classification moves away from classifying the technologies by their usage but rather does it by type:

- Distributed Tuples
- Remote Procedure Call (RPC)
- Message Oriented Middleware (MOM)
- Distributed Object Middleware (DOM)

4.1 Distributed Tuples

Distributed relational databases are most widely deployed kind of middleware currently. They offer an abstraction of distributed tuples (database record) using the Structured Query Language (SQL). They also offer an abstraction of transactions which provides heterogeneity across programming languages but very little or no heterogeneity across vendor platforms. The Open Group Distributed Transaction Processing (DTP) is a standard that is very common in

this class of middleware type. It is derived from Tuxedo which is available in many operating systems platforms and supports applications written in many different languages.

Linda is another common framework in this class of middleware types that offer distributed tuple abstraction called Tuple Space (TS). It offers both spatial decoupling (e.g. allows the deposit and withdraw actions to be unaware of each others identities) and temporal decoupling (e.g. allows the deposit and withdraw actions to have non-overlapping timelines).

JINI is another type of distributed tuple middleware built on JavaSpaces (a powerful JINI service from Sun Microsystems that facilitates building distributed applications for the Internet and Intranets [Sun1,2004], it is closely related to Linda's TS) that provides an open architecture that enables you to create network-centric services whether implemented in hardware or software--that are highly adaptive to change. JINI technology can be used to build adaptive networks that are scalable, evolvable, and flexible, as typically required in dynamic computing environments [Sun2, 2004].

4.2 Remote Procedure Call

This was developed by Sun as an extension to earlier distributed system technologies like Open Group's Distributed Computing Environment (DCE). A remote procedure call is simply a call sent from one machine or process to another machine or process for some service. An RPC is synchronous, beginning with a request from a local calling program to use a remote procedure and ending when the calling program receives the results from the procedure. RPCs offer no potential for parallelism without using multiple threads and have limited exception handling facilities.

4.3 Message Oriented Middleware

MOM is comprised of message-passing and message-queuing middleware in which information is passed in the form of a message from one program to one or more other programs. MOM provides the abstraction of a message queue that can be accessed across a network. It is very flexible in how it can be configured with the topology of programs that deposit and withdraw messages from a given queue. Many MOM products offer queues with persistence, replication, or real-time performance. MOM offers the same kind of spatial and temporal decoupling that Linda does.

4.4 Distributed Object Middleware

Distributed object middleware provides the abstraction of an object that is remote but whose methods can be invoked just like those of an object in the same address space as the caller. Distributed objects make all the software engineering benefits of object-oriented techniques encapsulation, inheritance, polymorphism available to the distributed application developer. The prime examples of platforms under this category include:

- **OMG's CORBA**

This open standard originates from the OMG. CORBA has evolved over a number of years. CORBA is independent of the operating system and the programming language used. The CORBA Component Model (CCM) is the version aimed specifically at distributed transaction processing and is part of the CORBA 3 definition.

CORBA is the most comprehensive in scope of all middleware technologies, largely because OMG explicitly addresses both the general- and vertical-market aspects of middleware's potential uses. In simple terms, CORBA allows applications to communicate with one another irrespective of location or origin.

The Object Request Broker (ORB) is the CORBA middleware that establishes the client-server relationships. Using an ORB, a client can transparently invoke a method on a server object, which can be on the same machine or across a network. The ORB intercepts the call and is responsible for finding an object that can implement the request. The client does not have to be aware of where the object is located, what language it's written in, or what OS it uses. By intercepting these requests, ORBs can provide interoperability between applications on different machines in heterogeneous distributed environments and can seamlessly connect multiple object systems.

However CORBA is currently seen to be fading out as it is being eclipsed by J2EE and .NET technologies.

- **Microsoft's DCOM and .Net**

Distributed COM is Microsoft's proprietary distributed-object technology. DCOM builds on the earlier Component Object Model (COM) architecture that provides a framework for application interoperation within a Windows environment. A client that needs to communicate with a component in another process cannot call the component directly, but has to use some form of inter-process communication provided by the OS. Functionally and technologically, DCOM is similar to CORBA. For example, both define an IDL and both provide support services such as persistence, security, and transactions. But there are significant differences, the foremost of which is that CORBA is an open specification, while DCOM is not.

Microsoft's latest development is Microsoft .NET, which includes supporting cooperation among widely distributed applications across the Internet and other networks. These technologies work only in Microsoft Windows environments, but are not tied to any programming language.

- **Sun's J2EE**

Sun's Java 2 Enterprise Edition standard was designed by Sun Microsystems. It has since been recognized by Object Management Group (OMG). It has a number of implementations including BEA WebLogic Server, IBM WebSphere application server and, more recently, the JBoss open source application server. Products implementing J2EE are independent of operating systems, but applications must be written in the Java language. Although it is theoretically possible to use other languages, there are no practical implementations.

- **Sun's RMI**

Distributed systems require that computations running in different address spaces, potentially on different hosts, be able to communicate. Sun's Java RMI offers a solution that enables communication between different program-level objects residing in different address spaces. Sun designed RMI to operate in Java. While other RMI systems can be adapted to handle Java objects, these systems fall short of seamless integration because of their interoperability requirements with other languages.

For example, CORBA presumes a heterogeneous, multi-language environment and thus must have a language-neutral object model. In contrast, the Java RMI system assumes the homogeneous environment of the Java virtual machine, which means the system can take advantage of the Java object model whenever possible, but also means the programmer is confined to Java.

- **Web services**

Web services are increasingly a great deal of attention. They provide a way of simplifying earlier electronic data inter-change (EDI) approaches. Web services are built on already existing and widely used standards including TCP/IP and

HTTP. A key standard in this category is SOAP (Simple Object Access Protocol). SOAP does not make any assumptions about the way applications are written.

Another key standard is Extensible Markup Language (XML), which SOAP uses for both its protocol and message encoding. SOAP can work effectively across the internet since it can use HTTP to carry the message. This is an advantage since HTTP does not cause problems with firewalls.

5. CONCLUSION

Middleware is undoubtedly now firmly established as an architectural concept for distributed computing. The dynamic nature of this sphere of computing suggests that the concept itself will change with time. The paper has outlined the origins of the concept and also presented the IT problem that that middleware tries to solve. Furthermore a discussion of exhaustive selection of the most common middleware technologies was undertaken thus providing an impartial presentation of issues involved. There is a growing degree of convergence of these middleware technologies. This will ultimately eliminate the myriad of confusing choices which will be a benefit in today's distributed system environments.

6. REFERENCES

[Bakken, 2002]. D.E. Bakken, "Middleware". Washington State University. School of Electrical Engineering and Computer Science.2002

[Bernstein, 1996]. P. Bernstein. "Middleware: A Model for Distributed System Services." Communications of the ACM, 39:2, February 1996, 86−98.

[Bye, 2003]. P. Bye. "What is middleware and where is it going?" .Unisys Technical Consulting Services. 2003.

[Campbell et al, 1999]. A. Campbell, G. Coulson, and M. Kounavis. "Managing Complexity: Middleware Explained." IT Professional, IEEE Computer Society, 1:5, September/October 1999, 22−28.

[IDC, 1999]. International Data Corporation. "Middleware and Businessware: 1999 Worldwide Markets and Trends". 1999.

[Schmidt et al, 1998]. D. Schmidt, D. Levine, and S. Mungee. "The Design of the TAO Real-Time Object Request Broker." Computer Communications, Elsevier Science, 21:4, April, 1998.

[Sun1, 2004]. Sun Micro-System. "JavaSpaces Principles, Patterns, and Practice". 2004. Available Online: http://java.sun.com/docs/books/jini/javaspaces/. Last Accessesed : 05 Sept 2004.

[Sun2, 2004]. Sun Micr-Systems."JINI Network Technology".Available Online: http://java.sun.com/developer/products/jini/index.jsp/. Last Accessed: 05 Sept 2004

[Zincky et al, 1997]. J. Zinky, D. Bakken, and R. Schantz. "Architectural Support for Quality of Service for CORBA Objects", Theory and Practice of Object Systems, 3:1, April 1997.