

Classification: Public



OSMOS Base Technology Selection

Author(s): **Simon HARVEY** **(USAL)**
 Alain ZARLI **(CSTB)**
 Yacine REZGUI **(USAL)**
 Grahame COOPER **(USAL)**
 Abdul Samad KAZI **(VTT)**

Issue Date	30 March 2001
Version	2.1
Deliverable Number	D2.1
Task Number	T2100
Status	<i>New Release</i>

Summary

This document details the results from a period of investigation into the underlying technologies that could be used to provide the infrastructure on which the OSMOS Solution will be specified and built. It complements, and should be read in conjunction with, especially, D2.2 (OSMOS Architecture Definition and Specification).

This deliverable, which is aimed at the level of a reader who has some expertise in Information Technology architectures, will introduce the underlying technologies on which the OSMOS Solution could be built upon. It will introduce the reader to networks and the Internet, introduce two and three tier architectures, introduce and analyse the various middleware technologies (such as RPC, Message-Oriented Middleware, Transactional Middleware and the more recent Object Based Middleware such as CORBA, (D)COM(+) and Java), highlight some of the security issues that need to be considered, introduce some technologies used for exchanging data and finally highlight some technologies that can be used to support teamworking in the industry.

This is the second revision of D2.1 - Final Release for Iteration 2. A new release will be produced for iteration 3.

PLEASE NOTE

The Consortium should like to emphasise that this deliverable, D2.1 Review and Selection of OSMOS Base Technology, is by its very nature an evolving document. The pace of technological change in the IT Sector is rapidly increasing and is changing on a monthly basis. Therefore, although the contents of this document reflect the state of the art as of 15 March 2001, the reader should be aware that advances may have been made which render some of this information dated.

Consequently, this deliverable should be thought of as a 'living document' and will be revised, in detail, on no less than one occasion per iteration of the OSMOS Project.

Contents

SUMMARY	2
ABBREVIATIONS	6
1. INTRODUCTION	7
2. REVIEW THE PARTNERS' BUSINESS PROCESSES AND REQUIREMENTS DEFINITION	8
2.1 ARCHITECTURAL REQUIREMENTS	9
2.2 TECHNICAL REQUIREMENTS.....	10
3. OVERVIEW OF THE POTENTIAL EMERGING INFORMATION & COMMUNICATION TECHNOLOGIES THAT SUPPORT THE OSMOS OBJECTIVES	12
3.1 COMMUNICATION TECHNOLOGIES: THE UNDERLYING INFRASTRUCTURE.....	12
3.1.1 <i>Evolution of Computer Networks</i>	12
3.1.1.1 Local Area Networks	12
3.1.1.2 Wide Area Networks.....	13
3.1.1.3 Internet	14
3.1.1.4 Intranets and Extranets.....	16
3.1.2 <i>Client/Server Architectures</i>	16
3.1.2.1 Two Tier Architecture	16
3.1.2.2 Three Tier Architecture.....	18
3.1.3 <i>Middleware Services</i>	20
3.1.3.1 Transactional Middleware.....	25
3.1.3.2 Message-Oriented Middleware	26
3.1.3.2a <i>Message Queuing MOM</i>	27
3.1.3.2b <i>Publish-and-Subscribe MOM</i>	28
3.1.3.3 Procedural Middleware	29
3.1.3.4 Object and Component Middleware.....	30
3.1.3.4a <i>Common Object Request Broker Architecture</i>	31
3.1.3.4b <i>Microsoft's Component Object Model Plus</i>	37
3.1.3.4c <i>Java-based Technologies</i>	40
3.1.3.4d <i>Comparison between CORBA, DCOM, JAVA / RMI</i>	41
3.1.4 <i>Comparison of the Potential IT Solutions That Support the Technical Requirements of the Virtual Enterprise</i>	42
3.1.4.1 Network Architecture.....	42
3.1.4.2 System Architecture	42
3.1.4.3 Middleware Technologies	42
3.2 SECURITY CONSIDERATIONS.....	44
3.2.1.1a <i>Data Confidentiality</i>	44
3.2.1.1b <i>Authentication</i>	46
3.2.1.1c <i>Authorisation</i>	48
3.2.1.1d <i>Non-Repudiation</i>	48
3.2.1.1e <i>Data Integrity</i>	48
3.2.1.1f <i>Firewalls</i>	49
3.2.1.1g <i>Audit Trails</i>	50
3.3 INFORMATION EXCHANGE AND DATA SHARING TECHNOLOGIES.....	50
3.3.1 <i>State of the Art for Semantic Product Data Modelling</i>	52

3.3.2 State of the Art for Information Interchange.....	53
3.3.2.1 XML for cataloguing and further distributing/recovering information.....	54
3.3.2.2 XML for enterprise-level services.....	55
3.3.2.3 The Simple Object Access Protocol (SOAP).....	56
3.3.2.3a Introduction.....	56
3.3.2.3b SOAP Key features.....	56
3.3.2.3c General structure of the SOAP protocol.....	57
3.3.3 Comparison of the Potential IT Solutions That Support the Technical Requirements of the Virtual Enterprise.....	61
3.4 CO-OPERATION TECHNOLOGIES (GROUPWARE)	62
3.4.1 Workflow.....	62
3.4.2 Document Management Systems.....	63
3.4.3 Decision Support Systems.....	64
3.4.4 Electronic Mail.....	65
3.4.5 LDAP - Lightweight Directory Access Protocol.....	66
3.4.5.1 What is LDAP?.....	66
3.4.5.2 LDAP Basics.....	66
3.4.5.3 More than a protocol.....	67
3.5 WIRELESS APPLICATIONS AND PROTOCOLS.....	68
3.5.1 Introduction: the Wireless Application Protocol (WAP).....	68
3.5.2 WAP Specification.....	68
3.5.3 Wireless Application Environment (WAE).....	70
3.5.3.1 Wireless Markup Language (WML).....	70
3.5.3.2 WMLScript.....	71
3.5.3.3 Wireless Telephony Application (WTA).....	71
3.5.4 Wireless Protocols.....	72
3.5.4.1 Wireless Session Protocol (WSP).....	72
3.5.4.2 Wireless Transport Layer Security (WTLS).....	72
3.5.4.3 Wireless Transaction Protocol (WTP).....	72
3.5.4.4 Wireless Datagram Protocol (WDP).....	72
3.5.5 Serving Content to WAP Devices.....	72
3.5.5.1 WAP Gateway.....	72
3.5.5.2 WAP Application Server.....	73
3.5.6 Examples.....	73
3.5.6.1 Plain WML Example.....	74
3.5.6.2 Servlet Example.....	74
3.5.6.3 ASP Example.....	75
3.5.7 WAP Services.....	75
3.5.8 Known Limitations.....	76
4. CONCLUSION: SELECTION OF THE OSMOS BASE TECHNOLOGY	77
ACKNOWLEDGEMENTS.....	79
REFERENCES.....	79

Figures

FIGURE 1: THE TWO TIER SOFTWARE ARCHITECTURE.....	17
FIGURE 2: THE THREE TIER SOFTWARE ARCHITECTURE.....	18
FIGURE 3: MIDDLEWARE (BERNSTEIN, 1996)	21
FIGURE 4: MIDDLEWARE TYPES AND VENDORS (HURWITZ, 1998).....	22
FIGURE 5: THE OMG OBJECT MANAGEMENT ARCHITECTURE (ORFALI ET AL. 1999:P478)	33
FIGURE 6: METHODS FOR SHARING/EXCHANGING CONSTRUCTION-RELATED DATA (COOPER & REZGUI, 2000) .	51
FIGURE 7: WAP PROGRAMMING MODEL (WAP FORUM, 2000).....	69
FIGURE 8: WAP PROTOCOL STACK (WITH MODIFICATIONS, WAP FORUM, 2000)	70
FIGURE 9: LAYOUT OF A WML DECK.....	71
FIGURE 10: SERVING WAP CONTENT THROUGH A WAP GATEWAY	73
FIGURE 11: SERVING WAP CONTENT THROUGH A WAP APPLICATION SERVER.....	73

Tables

TABLE 1 A COMPARISON OF MIDDLEWARE TECHNOLOGIES.....	44
TABLE 2 A COMPARISON BETWEEN XML AND STEP/IFC.	62

Abbreviations

Please note at a further document will be made available that will act as a Glossary for all the deliverables in Workpackage 2.

API	Application Programming Interface
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
ISP	Internet Service Provider
HTTP	HyperText Transfer Protocol
LAN	Local Area Network
JVM	Java Virtual Machine
MOM	Message-Orientated Middleware
ORB	Object Request Broker
OMG	Object Management Group
OSMOS	Open System for Inter-enterprise Information Management in Dynamic Virtual EnvirOnmentS
PC	Personal Computer
RPC	Remote Procedure Call
RMI	Java Remote Method Invocation
SQL	Structured Query Language
TCP/IP	Transport Control Protocol / Internet Protocol
WAN	Wide Area Network

1. Introduction

The work presented in this document (Deliverable D2.1) is conducted within Work Package 2 of the OSMOS Project. The aim of this workpackage is to review and select the OSMOS base technology and specify the basic system architecture underlying the OSMOS system. The main components of this architecture will include a set of semantic models, along with construction specific services, packaged in the form of an OSMOS API, supporting teamwork in a Virtual Enterprise (VE). This architecture will provide the framework for a smooth integration of existing proprietary and commercial applications and for the conceptual development of OSMOS ideas. Finally, the workpackage will define the business and technical requirements for setting up the OSMOS team work service providers: the latter will have the fundamental mission of providing the IT infrastructure of the VE, including toolkits, information / document servers, and OSMOS plug-ins that would enable non co-located teams through their proprietary and commercial applications to join, and take an effective part to, the VE.

Furthermore, the aim of the present deliverable is to review emerging information and communication technologies, and teamwork services, such as services supporting communication (including middleware), co-operation (including standardised shared information repositories), co-ordination (including task synchronisation, and access control), and documentation management (including document routing, and version control).

The deliverable will also lead to the selection of the software and hardware technology and services for the OSMOS project.

2. Review the Partners' Business Processes and Requirements Definition

The analysis of the end-users business processes and information management practices reveals the following:

- *Homogeneity*: despite recent evolutions, mainly due to the impact of the internet, existing solutions are still often fixed and not open, with a lack of support for legacy, as well as new, upcoming systems in terms of hardware, software, databases, and networks.
- *High Entry Level*: IT solutions are still often expensive to buy for SMEs, as reported by the OSMOS end-users. More entry levels should be provided, e.g. from personal (low cost) to enterprise (high cost) editions.
- *Lack of Scalability*: most available proprietary and commercial solutions offer limited growth path in terms of hardware and software.
- *Application Centric and lack of support for business processes*: there is often a requirement to organise the enterprise around the adopted IT solution.

Given the limitations described above, a suitable teamwork solution for OSMOS would require the following:

- Providing construction specific, and scalable solutions, that take into account the particular organisational settings of each construction enterprise participating in the VE, including SMEs.
- Providing IT and organisational solutions that promote trust and social cohesion among the partners of a construction VE.
- Providing effective, model-based solutions, to support Communication, Co-operation, and Co-ordination between individuals and groups collaborating in a construction VE, based on the specificity and information / process requirements of the Construction domain.
- Providing models for business processes, working methods, organisation, contracts, and legal responsibilities related to CSCW in a VE.
- Providing services, packaged in the form of an OSMOS API, that enable, on the one hand, construction industry software to be integrated with traditional Groupware software components, and, on the other, accommodate intra-company and inter-company communication.

From these, we can deduce two sets of requirements that need to be satisfied by the OSMOS solution. The two major sets, Architectural Requirements and Technical Requirements, may also be divided into four requirement categories, although some of them fall into more than one category:

- **[COM]** Requirements relating specifically to communication and communication methods.
- **[COP]** Requirements relating specifically to enhance co-operation.
- **[COD]** Requirements relating specifically to help manage co-ordination.
- **[GEN]** All other requirements.

The technologies presented in Chapter 3 of this document will be analysed with respect to the level of which they satisfy these requirements.

2.1 Architectural requirements

These requirements are related to needs as expressed by the end users in terms of architectural solutions that must answer the desired functionality.

[GEN] Platform independence (or interoperability): this is the ability for the client and/or the server to run on almost any OS, and hardware platform.

[COM / COP] Language independence (or interoperability): this is the ability for the client and/or the server to be implemented using any almost language, and to allow requests and/or method invocations no matter the requesting language and the implementation language.

[GEN] Management of huge set of data: this is the ability to manage data of (nearly) any size, e.g. the data size has not to be taken into account (from a user application point of view, and independently of any performance concern) for application interoperability/inter-change. A specific point here is the fact that it is required or not that all the data that are accessed/exchanged should be in the application memory (or accessible through any persistent mechanisms)

[COM / COP / COD] Distribution support: this the ability of the system to “natively” provide with a set of (at least basic) services required in concurrent distributed systems, e.g. lookup facilities (naming and/or trader), transactions, control of concurrency, remote method invocation, etc. (*Note: security is considered as another full requirement, see below*).

[COM / COP] Communication (or messaging) flexibility: this is the ability of the system to support new types of messages and data flows, which require an easy updating of generic repositories used for administration, control and transformation of messages (at level of the global VE).

[COD] Schema versioning (in the sense of interface evolutions): this is the ability of the system to handle data streams that correspond to different versions of a schema definition.

[COM] Communication protocols: this is the ability of the architecture to support various communication protocols, and thus not to be dependent on only one single protocol (or not to provide with a model of communication that is strongly influenced by the protocol, even if the model can be implemented through other protocols, probably with far less efficiency).

[COM] Communication types: this is related to the fundamental types of communication, i.e. the ability of the system to support synchronous and/or asynchronous communication¹.

[GEN] Code Portability: this is the ability of the system, implemented in a given OS-based environment, to be recompiled and run on top of another environment *without any change in the code*.

[COD / COP] Security levels: this refers to the various levels of security that exist within the system. The mandatory expected services are identification/authentication and authorisation/access control. Strongly desirable ones can be confidentiality and message integrity. Others can be audit and non-repudiation (those are indeed related to integrity, through dedicated form of functionality).

[GEN] Standardisation: this is a characterisation of the capacity of the framework to integrate standards (standardised data, standardised messages, standardised protocols, etc.), Additionally, it could be related to the process of standardising the framework itself.

[GEN] End user enterprise applications integration: this point is to realise the integration of (legacy or commercial packaged) enterprise application with the framework with invasive or non-invasive application adapters.

[GEN] Quality of service: this is linked to the notions of guaranteed delivery of information, delivery in real time (or “due time”), etc. Another consideration is to know if checking/validating the type of information is required (and if so, at the source or the target application), and/or to manage or not a unique point of conservation of valid information (typically on the server), consequently leading to what can be called a *permanent data coherency*. Additionally, transactional mechanisms can here be considered as one of the fundamental devices to ensure integrity of data and thus being a full component for quality of service.

2.2 Technical requirements

These requirements should constitute criteria for further selection of the underlying technology, provided that the technology fits with the previous architectural requirements.

¹ A communication is said *synchronous* when the server responds immediately or almost immediately (depending on the network latency) to a client request. This implies on one hand an existing server and on the other hand that this one is active at the moment the request is sent. Generally this reaction is realised by the execution of the requested operation. A communication is said *asynchronous* when a request does not involve or ask for an immediate effect on the server, and that the operation execution could be postponed. In this case, the request processing will be done when it suits the server.

[COP] Handling of any data types or specific types: this is the ability to access/exchange information that is of any “strong” type (integers, floats, boolean, arrays, unions, etc.), or the ability to manipulate only data flows with strings that have then to be mapped by the application itself.

[COP] Object oriented model handling: this is the ability to allow the integration and management of the application object model (potentially at any level of object granularity) without any specific adapter layer. Thus, this is related to the adaptability (or not) of the system to deal with various applications interfaces or invocation semantics.

[COM / COP] Model for the description of components interaction: this is the identification of the complexity of the model of communication between applications/objects (in terms of implementation and integration: this has to be related to tight coupling (versus low coupling) between applications in the framework), along with a required strong or weak adherence from any application of the framework to the semantics of messages (e.g. CORBA messages through IDL messages *versus* XML messages).

[GEN] Code mobility: this is the ability to provide mechanisms to ship code in order to make some code execute locally, e.g. on the client side.

[COD / COP] Security: security is here considered from a technical/operational point of view, i.e. the capacity of dealing with firewalls (e.g. IIOP proxy if an IIOP stream has to pass a firewall), of providing mechanisms to protect information (e.g. the Java sandbox), etc..

[GEN] Object memory management: this is related to some capability to manage application memory and data caching following an optimal way (this is related to performance requirements).

[GEN] Performance: this criteria characterises a process execution time, and has to be associated to the identification of a set of basic services or operations allowing the comparison of performances of different technologies in a similar *functional* architecture.

3. Overview of the Potential Emerging Information & Communication Technologies That Support The OSMOS Objectives

There are a number of information and communication technologies that could be implemented to achieve the OSMOS overall objective:

The overall aim of the OSMOS project is to enhance the capabilities of construction enterprises, including SMEs, to act and collaborate effectively on projects by setting up and promoting value-added Internet-based flexible services that support team work in the dynamic networks of the European construction industry

(OSMOS, 1999)

The purpose of this chapter is to present an overview of technologies that could potentially support the future OSMOS infrastructure and services, with particular attention being focused on those technologies that can be used as part of an internet-based infrastructure.

3.1 Communication Technologies: The Underlying Infrastructure

This section will introduce the multitude of information and communication technologies that could be used to provide the underlying infrastructure for the OSMOS project. This section will be continually referred to from Chapter 4 of this deliverable, as well as the D2.2, D2.3 and D2.4, when discussing which technologies should be implemented for the OSMOS solution.

3.1.1 Evolution of Computer Networks

This section gives a brief overview of the current networking topologies that are in widespread use today, emphasising their relative advantages and disadvantages at a generic level.

3.1.1.1 Local Area Networks

A Local Area Network (LAN) is a group of computers, workstations and peripherals that are connected to each other over a relatively small geographical area, such as within a building or small group of buildings (Webopedia, 1998). LANs tend to be built upon a proprietary communications technology especially if the network consists of a single operating system. In addition the network protocol used may be based on communications standards such as TCP/IP (see Section 3.1.1.3).

LANs can give everyone shared access to shared resources such as file servers and to expensive peripherals such as colour laser printers. They also allow users to share files and documents across the network by setting appropriate permissions on their systems. Some LANs also allow users to dial into the network remotely via the telephone network.

Groupware services such as workflow, internal electronic mail and directory services can be used on a LAN although the use of such services is totally constrained by the small size of the network and its users.

Although the LAN gives relatively fast and reliable performance to its users, it can only be used within a small geographical boundary and most certainly within an organisation or organisational department. If a user wishes to share a file or document with somebody outside of the LAN, then the file must be transferred by some removable medium such as via floppy disk, ZIP disk, Compact Disc or printed out onto paper.

The *de facto* standard low-level protocol used within LANs is Ethernet (devised by Xerox, DEC and Intel in 1976). It defines the physical cabling and network communication layers and transfers data at speeds up to 10 megabits per second (Mbps). Other implementations (e.g. Fast Ethernet, Token Ring, Asynchronous Transfer Mode and Gigabit Ethernet) transfer data at much faster rates, such as 100 Mbps for Fast Ethernet, however the hardware costs increase proportionally. Communication across the LAN also involves the use of a *protocol*, a 'standard' language used for computers and peripherals to communicate with each other. Examples of protocols in widespread use today include:

- *TCP/IP* for Unix and Windows 9x/NT-based networks
- *IPX* for use within Novell's NetWare-based networks
- *DECnet* for Digital Equipment Corporation's networks
- *AppleTalk* for networking Apple Macintoshes
- *NetBIOS and NetBEUI* for LAN Manager and Windows NT-based networks

3.1.1.2 Wide Area Networks

Two or more LANs that are connected together in some way (with the required hubs and bridges to convert between the protocols used in each LAN) is called a Wide Area Network (WAN). WANs can often span large geographical areas such as cities, regions and countries. WANs can be connected by a variety of means, such as through satellite, radio and telephone (leased lines) links. These are all expensive forms of electronic communication, when compared to the Internet (see Section 3.1.1.3).

Depending on what is required by the organisation, a WAN could encompass parent and sibling companies, link together industries (such as the University and Military networks) or even multi-industry groups.

However as the size and complexity of the WAN increases, the management of the network does increase exponentially as does the whole host of security issues (see Section 3.2) that

come to the fore, both within the component LANs and the interconnecting WAN communication links.

WANs basically use the same underlying technologies as LANs (see Section 3.1.1.1) although the physical cabling used conforms to industry-wide standards². The protocols used tend to be Asynchronous Transfer Mode and Frame Relay (Orfali *et al*, 1999:61). The largest WAN that currently exists is the Internet.

3.1.1.3 Internet

The Internet can be thought of as thousands of interconnected LANs and WANs that spans the globe.

The emergence of the Internet (Whatis.com, 1999) offers the unique opportunity for actors to connect to a server of their choice from their PC regardless of their physical location, anywhere in the world, as long as they have the appropriate rights and permissions to do so. Many organisations are now in the process of giving their actors access to the Internet via their LANs and WANs to aid them in the course of their day-to-day duties. They are also connecting servers to the Internet to make information available to others, around the world 24 hours a day, 7 days a week. This information can be made public for anonymous access, or can be made private to only allow access by a specific group of users. In the latter case, this can be achieved using a number of security related technologies (see Section 3.2).

The Internet can usually be accessed by using an Internet Service Provider (ISP) via the local telephone networks. Different ISPs serve different types of customer depending on their requirements. Medium and large organisations tend to have a permanent connection to the Internet whilst individuals and smaller organisations connect on an ad-hoc basis, as and when required. The cost of such connections varies with the customer's requirements, such as bandwidth, availability and associated services. ISPs charge organisations and individuals to connect to the Internet, usually on a fixed fee in the former case and on a per-minute basis in the latter, although in a growing number of countries (especially in the UK and gradually throughout Europe) certain ISPs have waived their charges for individuals. In these cases the only financial cost is the cost of the telephone call, usually at local rates.

For those that are not permanently connected to the Internet via their LANs and WANs, a modem is required to convert digital computer signals into an analogue format that can be sent through the telephone network to the ISP. Modems are usually included in most computer packages and have recently been included in laptop computers. In recent years, the explosive growth in use of laptop computers has allowed actors to connect to the Internet using a mobile telephone. (The actual connection between the PC and the telephone is by using a serial cable, or by using infrared ports which are commonly being included on telephones and laptops.) This enables actors to work whilst on the move or in an area which has no permanent telephone lines installed.

² Discussion of which is outside the scope of this document. However if the reader requires further information we refer to Chapter 4 of Orfali *et al*. (1999) where technical issues regarding cabling are discussed in detail.

All the applications that can be used via the Internet are built upon the Transmission Control Protocol/Internet Protocol (TCP/IP) (Socolofsky & Kale, 1991). This protocol is supported at the network layer by all the operating systems in common use worldwide. The major application protocols in use on the Internet today all make use of TCP/IP. These include:

- HyperText Transfer Protocol (HTTP),
- File Transfer Protocol (FTP),
- Simple Mail Transfer Protocol (SMTP) for E-mail,
- Telnet,
- Network News Transfer Protocol (NNTP)
- and Gopher.

The World Wide Web is a user-friendly front-end to the Internet which subsumes most of the aforementioned protocols (Orfali *et al.*, 1996:p466).

The Internet is a self-sustaining mechanism that allows actors to communicate with each other and conduct research regardless of physical location. It can also be accessed by the vast majority of personal computer systems worldwide, 24 hours a day, 7 days a week. Accessing the Internet is usually far cheaper than building WANs, especially for those involved in global or virtual organisations. The dynamic TCP/IP protocol does not place reliance on a single server - if a server on the Internet goes down, the protocol tries to find another route for a packet of data to reach its destination.

Internet performance can vary due to a multitude of factors, including the type of connection the user has to the Internet, how busy the Internet actually is and the software used on the client machine. In addition, the Internet is an uncontrolled global network and is inherently insecure.

For example, few realise that electronic mail sent across the Internet is transferred in "plain text" - analogous to sending documents through the postal service without being covered with an envelope. In addition to breaking confidentiality, this allows malicious users to be able to intercept electronic communications through the use of computer programs called *sniffers* without being detected. Also the constant, and indeed worldwide, availability of the Internet makes it easy for hackers and crackers to target organisational servers, with the intention of breaking in to modify and/or steal confidential data. Again, this usually occurs without detection. Attempts are made to combat this problem through the use of security technologies (see Section 3.2), however it is usually difficult to bring these people to justice as the laws of one country do not usually apply in another, i.e. there is *no universal legal or governmental system* that controls the Internet³.

³ Although outside the technically-focused scope of this document, the Consortium feels that this point needs to be considered within the OSMOS project as a whole.

3.1.1.4 Intranets and Extranets

An Intranet can be thought of as an 'organisational mini-Internet' that serves, and can only be accessed by, those actors within a specific organisation (TechWeb, 1999). Intranets are usually set up within company LANs and WANs. Usually characterised by the use of web-based technologies (such as HTML pages, Java, JavaScript, discussion groups and web-based groupware services), the Intranet acts as an 'internal Internet' that is accessible by those actors within the organisation. Like with the Internet, the networking protocol employed is TCP/IP and is supported on all major computer configurations.

As an Intranet is self-contained within an organisation, Intranets tend not to suffer the major security disadvantages of the external Internet as they would fall within the remit of the organisation's existing security policy. However if company Intranets are linked via the Internet, the previously mentioned security considerations do need to be taken into account. Intranets linked in this way are often termed *Extranets*.

3.1.2 Client/Server Architectures

Client/Server architectures have been around since the mid-to-late 1980s and were designed to overcome the problems of scalability and performance with a large number of users that occurred with the preceding Mainframe and File Sharing architectures. The growth in popularity and processing power of desktop personal computers has also introduced the possibility of implementing user-friendly Graphical User Interfaces (GUIs) on the client side, as opposed to text-based terminals as used in the preceding era. The aim of the architecture is to improve the usability, flexibility, interoperability and scalability of computer systems by distributing certain tasks to where the processing is most needed, as opposed to centralised, mainframe, timesharing computing. (Sadoski, 1997).

3.1.2.1 Two Tier Architecture

The two-tier architecture consists of three components distributed into two layers: the client (which requests services) and the server (provider of services). Sadoski (1997b) describes how the three components are:

1. *User System Interface* (including session, text input, dialog and display management services)
2. *Processing Management* (such as process development, enactment, monitoring and resource services)
3. *Database Management* (including file and data services).

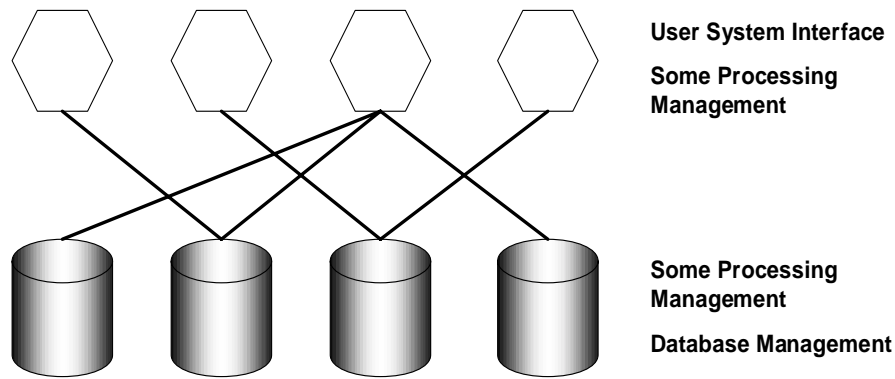


Figure 1: The two tier software architecture

Figure 1 shows how these components are distributed among the two tier architecture, with the processing management being shared on both the client and server sides. Here the PC client would assume responsibility for the application/functionality logic whilst the database server (located on specialised hardware) handles the data intensive tasks. Communication would take place between the tiers in the form of SQL statements.

The two tier model works extremely well in environments which are relatively homogeneous with fairly static business rules. It is possible to change dynamically the connectivity between tiers depending on the user's request for data and services (Sadoski, 1997b). The time required for application development, when compared to designing and implementing legacy systems, is also considerably reduced (Gallaughner & Ramanathan, 1996). This can be achieved by using Rapid Application Development techniques and by using an iterative approach to development by installing the client and server onto one machine.

However, Schussel (1996), Sadoski (1997b) and Gallaughner & Ramanathan (1996) point out a number of limitations of the two tier approach. They can be summarised as follows:

- *Scalability.* After the network architecture exceeds approximately 100 concurrent users, the performance of the architecture seems to deteriorate. This is due to the high number of 'keep alive' messages passed between clients and servers saturating the network.
- *Administration.* As some of the processing is done on the client side, there could be problems with version control and distribution of client-side upgrades. This problem is worsened in cases where there are frequent changes of business rules.
- *Interoperability.* The SQL middleware and client tools available within the market tend to be of a proprietary nature, with relatively poor client-side development tools compounding the problem. In addition proprietary stored database procedures which are implemented as part of vendor solutions often make it difficult to transfer from one vendor to another.

3.1.2.2 Three Tier Architecture

The three tier architecture tries to overcome these problems by inserting a tier between the client and server, separating out the business and process logic from these tiers, as depicted in Figure 2. The purpose behind the middle tier is to remove the burden of process management from the client and server tiers, by providing functions such as queuing, scheduling, prioritisation, application execution and database staging. (This allows the client to deliver a request to the required processes in the middle tier and then disconnecting, freeing itself up to do other work. The required process then independently handles the request and returns the answer to the client when completed.) The middle tier provides a location where business rules and logic can be executed whilst concurrently hiding the complexity of the architecture from the user (Sadoski, 1997c).

Like with the two tier architecture, different hardware, software and operating system configurations can be implemented on each server within each tier with special consideration given to the needs of each server's process, as long as each server can communicate with the others using standard protocols (e.g. TCP/IP) and query languages. This allows, for example, specialised database servers with multiple processors to be used in the middle and server tiers, whilst the user interface clients may be thin clients running on desktop or laptop PCs.

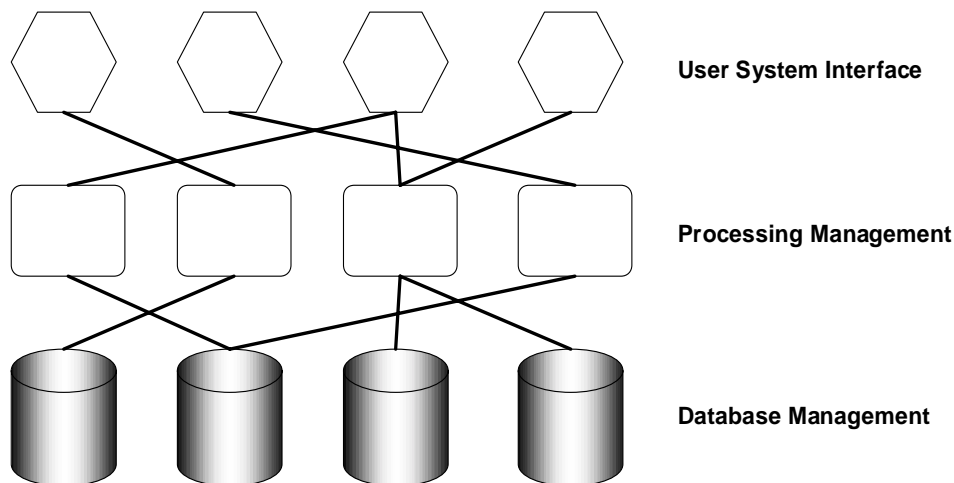


Figure 2: The three tier software architecture

There are a large number of functions and services that can be performed in the middle tier. Schussel (1996) and Sadoski (1997) discuss these in depth, however these functions can be summarised under the following generic headings:

- *Transaction Processing (TP)*. Transaction Processing technology offers services such as message queuing, transaction scheduling and prioritisation services. Instead of clients connecting to the database servers, they connect to TP monitors that pass the request on to the relevant database servers, often multiple servers in one transaction.

TP monitors tend to support multiple data source types and include robust security services.

- *Messaging Servers.* These provide the same services as TP Monitors but also can connect to data sources other than DBMS servers to obtain information. The difference is within the requesting messages - here the system intelligence is built into the message itself whereas with TP monitors the intelligence is built into the TP server. Messages are seen as discrete objects that know their source and destination addresses, and can even invoke stored procedures or business logic on the messaging server if applicable.
- *Application Servers.* This approach moves the main body of an application (sharing and including business logic, computation and data retrieval) onto the middle tier, as opposed to bundling it with the presentation logic on the client side. The client side then concentrates on the processing logic, driving a GUI for example - very similar to the role that a terminal plays within a mainframe architecture. With the main application being hosted on its own server (within a controlled environment) and shared by many clients, issues such as version control and security can be more rigorously controlled as version updates are not usually required on each client machine. The major advantages of this approach are increased scalability, flexibility, security and control over maintenance costs. However the initial high set-up and required application development costs tends to act as a disadvantage to using this approach.
- *Distributed Components.* This approach takes the application server school of thought and splits the application into self-contained components throughout the middle tier, that share a common communications (and associated services) backbone. As these components are self-contained business objects (including data and procedures), they can be shared across a network tier irrespective of required hardware and operating systems as long as they conform to the backbone. This opportunity for improved interoperability shows great promise for developers and gives administrators the ability to move objects between servers to improve performance by using under-utilised processing time on an ad-hoc basis. The backbone that is required for this is provided by Object Request Broker (ORB) technologies (see Section 3.1.3.4).

As can be inferred from the above descriptions, the splitting up of applications into tiers using any of the above approaches can free the developer from worrying about issues such as remote data source types, networking and security, when writing an application. The degree to which this can be done varies with each type of three tier architecture, but is most apparent in the Distributed Components approach. For example a development team may concentrate on the presentation layer of an application without having to know about the type of data sources being used.

Studies have shown that the three tier architecture improves system performance and scalability over two tier architectures by supporting thousands of users at any one time. In addition flexibility is increased as it allows system administrators to move processes between servers dynamically in response to network usage conditions. Middle tier processes are also usually coded in non-proprietary languages such as C, C++, SmallTalk, Java and Ada, which

aids interoperability and the ease of porting applications across heterogeneous platforms. The use of middle tier applications also helps to remove dependency on vendor specific client and server applications. Legacy systems can also be kept in useful service using this approach by maintaining the old database and process management rules in the middle tier until such time that each application and data source is moved over onto the new design (e.g. from RDBMS to OODBMS). (Gallaugher & Ramanathan (1996), Sadoski (1997c)).

However, the major disadvantage of the three tier approach is the complexity of building the architecture. Because of the sheer number of permutations such an architecture can be implemented, the development environments that are currently available (such as those provided by BEA, IBM and Oracle) do not fully cover every available client and server platform, operating system and data structure. Therefore, there is some effort required in 'filling in the holes' left by these products. However, Gallaugher & Ramanathan (1996) show in their paper that although there is increased initial cost and effort in designing and developing a three tier architecture as compared to a two tier architecture, considerable savings are made in subsequent development efforts and migrating between client development tools. Additionally, Sadoski (1997c) also argues that the separation of logic into the three tiers is not always an obvious one.

3.1.3 Middleware Services

Orfali *et al.* (1996:16) defines middleware as 'a vague term that covers all the distributed software needed to support interactions between clients and servers'. Bray (1997) also describes middleware as 'connectivity software that consists of a set of enabling services that allow multiple processes running on one or more machines to interact across a network'. From these definitions we can infer that middleware can be found in the middle tier of the three tier architecture, whilst it can also be found on both the client and server sides of a two tier architecture.

As described in Section 3.1.2.2, the use of middleware allows clients and server applications to communicate across heterogeneous platforms. Figure 3 shows how the applications communicate with the middleware services by using Application Programming Interfaces (APIs), and how a further set of interfaces allow the middleware to communicate with the platform-specific servers. The use of interfaces allows the application designer to concentrate on, for example, designing and implementing the client without having to worry about having to code the underlying network services provided by the middleware and the actual data sources, as long as the required parameters are passed via the appropriate API.

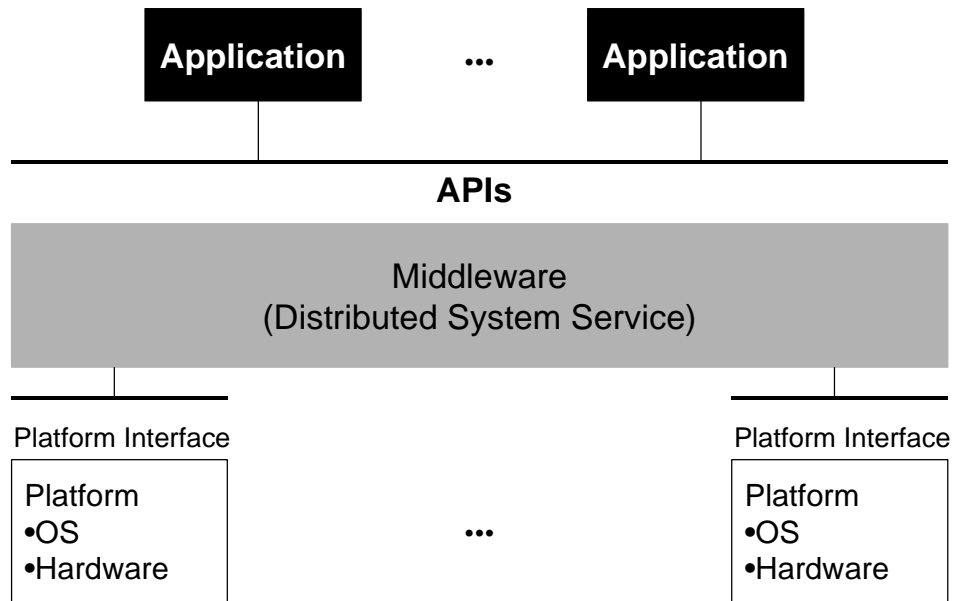


Figure 3: Middleware (Bernstein, 1996)

It is difficult to define middleware in a technically precise way, as it is such a high-level concept. Indeed, Bernstein (1996) acknowledges there is no standard terminology for many of the concepts used and that the terms used by vendors and consortia tend to be advocated in conflicting ways. Bernstein (1996) argues that middleware has several properties that, when taken together, usually make clear that the component is not an application or platform specific service:

- Middleware is generic enough to meet the needs of many industries.
- Middleware Services have implementations that run on multiple platforms. (This is usually why the implementations are coded in non-proprietary languages such as C, C++ and Java.)
- Middleware offers distributed services to enhance interoperability across different platforms.
- Middleware itself can be accessed remotely, and enables other services and applications to be accessed remotely.
- Middleware supports *de jure* standard protocols such as the ISO OSI protocol. However, *de facto* and other published non-proprietary standards are also usually supported, such as TCP/IP and SQL.
- Middleware services also should support a standard API, with their services being transparent with respect to that API without requiring modification to the API.

The main advantages of middleware (i.e. increases in network and application flexibility, scalability, reliability and robustness, ability to hide complex distributed systems services

from the developer, transparency of applications on the network, decreased maintenance and integration with legacy systems, etc.) have been previously outlined in the section on three tier architectures. However Bernstein (1996) acknowledges there seems to be a gap between theory and practice - a number of vendors have introduced proprietary implementations into their products, making interoperability between some middleware products difficult. There are also some hard design choices to be made - the designer must decide which functions are part of the client, server or middleware when designing an application. Indeed, Bernstein (1996) notes that certain middleware services such as communication services may migrate into, and become part of, operating systems and vice versa!

Because of the high level abstraction used in the literature when discussing middleware, it is often difficult to classify the various types into generic headings. Hurwitz (1998) proposes that we can classify middleware based on its such as recoverability and scalability. These characteristics have an inverse relationship, so Hurwitz (1998) describes how they can be displayed as a matrix (Figure 4) which also gives examples of products available in the marketplace.

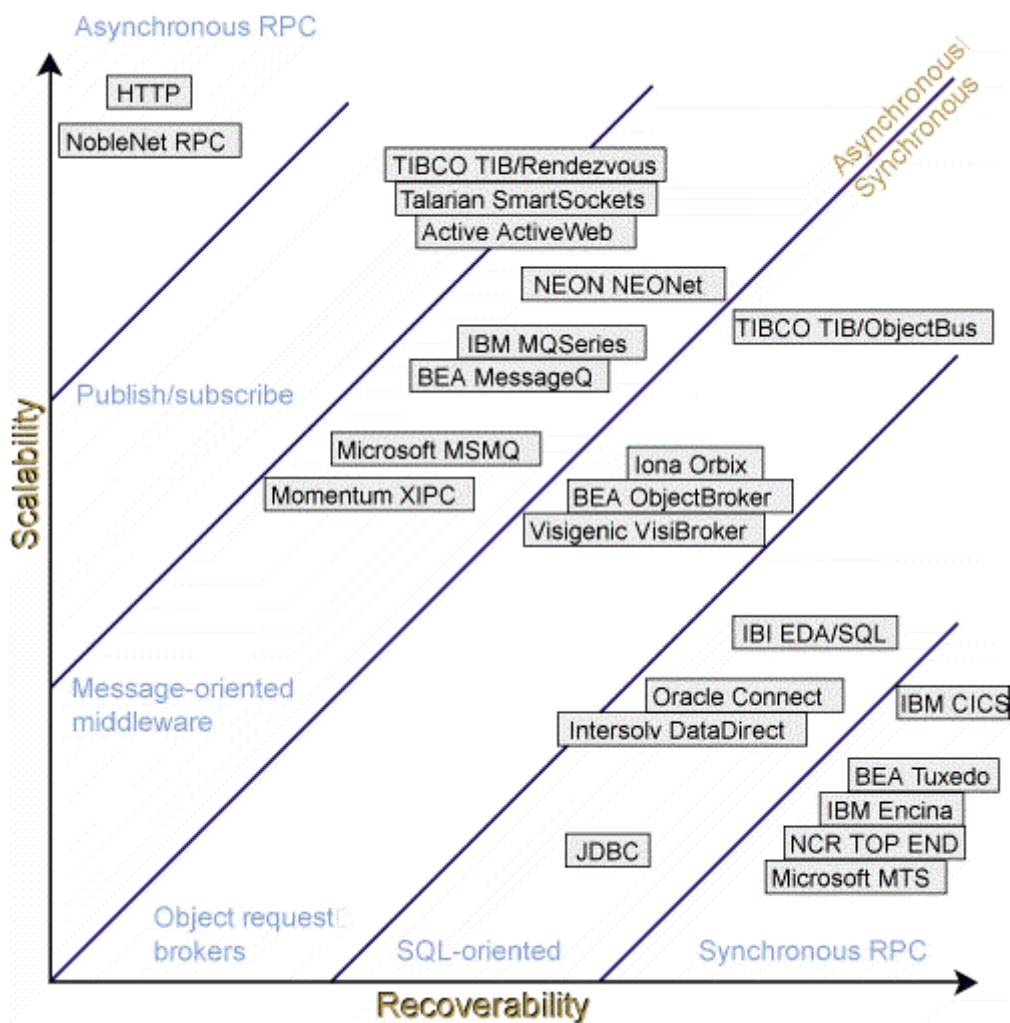


Figure 4: Middleware Types and Vendors (Hurwitz, 1998)

As Hurwitz (1998) acknowledged in her article, the actual marketplace isn't as clear cut as the matrix suggests. This is because the fine lines separating these classifications have recently become blurred, as the vendors are starting to provide services within their products that encompass more than one classification.

However, Emmerich (2000) provides a powerful framework within which he compares middleware technologies based upon the methods of interaction between distributed objects. The paper compares the technologies based on the following areas:

- *Network Communication.* How well does the middleware effectively hide the complex networking infrastructure, protocols and operating systems from the developer and objects? This can be thought of as hiding the transport layer of the ISO/OSI reference model. However sophisticated middleware also hides layers above this level, specifically the session and presentation layers, thereby allowing the developer to provide parameterised requests between objects without needing to worry about implementing code to maintain sessions or worry about converting complex data structures to (*marshalling*) and from (*demarshalling*) a format that can be transferred over heterogeneous networks respectively.
- *Co-ordination.* Objects on the same host are executed concurrently. However, when these objects are distributed with multiple points of control, how can these objects be synchronised so communication between themselves is conducted in the correct manner? *Synchronous* communication occurs when a client component waits for a server component to complete its service before continuing. In cases where a request is issued (and immediately continues with its work) with the intent of synchronising later, such synchronisation is termed *deferred synchronous* when the client initiates the synchronisation by methods such as polling. In cases where the server initiates the synchronisation process, the term *asynchronous* is often used. In addition, how does the middleware support *group requests* where more than two components are interested or involved in a service request?

In addition, Emmerich also discusses how the middleware should independently manage the *activation* and *termination* of (potentially a large number of) components from the application, to co-ordinate component availability on hosts with limited resources such as (virtual) memory. The same principle applies to servicing concurrent requests by the implementation of *threading policies*.

- *Reliability.* Networking protocols vary in their ability to guarantee that every single packet reaches its destination. The middleware therefore needs to provide services such as *error detection and correction*. As reliability is often associated with a trade-off in system performance, the question of provision of varying degrees of reliability/performance by the middleware comes to the fore. For example, reliability of a request being received at its destination can be classified as *best effort* (no guarantee is made), *at-most-once* (guaranteed, but notified of a failure if applicable), *at-least-once* (guaranteed, but may be executed more than once), or the ideal *exactly-once* (guaranteed to be executed only once). For group communications, *k-reliability* guarantees that at least *k* components receive the

packet, *time-outs* specify a time period after which attempts to resend the packet must cease, whilst *totally-ordered* communication ensures that packets are always received in the correct order.

Grouping individual requests into transactions, the middleware should be able to observe the ACID⁴ principle ensuring that a transaction is successfully completed or not at all, leaving the objects in a coherent state. In addition, reliability of the system as a whole may be handled by the middleware, by supporting the replication of objects for fault-tolerant computing, whilst keeping these replicas synchronised (including the *state* of a component) to provide services at any time.

- *Scalability* is where the architecture can accommodate any future growth in demand and system load. Emmerich (2000) notes that this can be achieved by distributing the load across several hosts, although the difficulty is to support such *load balancing* without changing the architecture of the system and without modifying the design and implementation of any component. This can be achieved by middleware implementing the concept of *transparency*, where it allows the architecture to provide services without the client necessarily knowing where, or how, the server object is implemented. Examples include *Access transparency* is where the client object accesses the services of another object in exactly the same way, regardless of whether the serving object is local or remote; *remote transparency* is where the client is ignorant of the serving object's precise location; *migration transparency* occurs when the client does not know that the serving object has been moved onto another host to optimise resource utilisation (this is known as *load balancing*); and *replication transparency* is where the client does not know whether it is requesting the services of a replica of, or the actual, master object. These goals are quite complex to achieve especially in heterogeneous environments.
- *Heterogeneity* is where the middleware ensures that communication is achieved throughout the architecture, regardless of the different types of hardware and operating systems used, what data sources are accessed (and whether they are legacy or not) and what programming languages used (and the differences in programming models – e.g. inheritance is handled differently in different object-oriented programming languages); without the knowledge of the client. Indeed, in cases where different types of middleware are implemented, there is a need for these to interoperate.

The OSMOS Consortium feel that Emmerich's approach is most suitable in the context of OSMOS to be used as a platform for the following discussion of middleware technologies. It is interesting to note that Emmerich advocates that a combination of, not a single, middleware solution because a single solution rarely produces the desired architecture. Emmerich (2000)

⁴ See Section 3.1.3.1

classifies technologies based on the primitives used to communicate between distributed components: *transactional*, *message-oriented*, *procedural* and *object or component middleware*.

3.1.3.1 Transactional Middleware

Orfali *et al.* (1999:320) defines a transaction in business-language as ‘an action that changes the state of an enterprise.’ They give an example of a customer depositing a cheque into their bank account, as this is a banking transaction. However, technically speaking, they also define a transaction as ‘a collection of actions imbued with ACID properties’:

- **Atomicity** is where a transaction is an indivisible unit of work: it either completely succeeds or it fails.
- **Consistency** requires that after the execution of a transaction, the system must be left in a correct but stable state. If this is not the case, the transaction must abort.
- **Isolation** requires that a transaction’s behaviour is totally unaffected by any other transaction that is executed concurrently, for example the transaction’s changes to any shared environment remains hidden until the transaction successfully completes (*commits*).
- **Durability** (a synonym for *persistence*) is where the effects of the transaction are made permanent after the transaction commits.

(Orfali *et al.*, 320-321)

Transactional middleware ensures that these principles are adhered to by managing the communication between client and service-providing components as transactions, and by implementing protocols such as the two-phase commit protocol (Bernstein *et al.*, 1987). Transaction Processing (TP) Monitors provide services for managing processes, transactions and client/server communications. Because they ensure that transactions satisfy the basic ACID requirements, they automatically ensure data consistency, recoverability and reliability throughout the architecture. Orfali *et al.* (1999:321) even propose that in an ideal client/server environment, all client/server communication should be in the form of transactions.

As clients access services provided by a component as part of a transaction, transactional middleware hides the complexity of the underlying network from both client and server. Emmerich (2000) also notes that communication can occur in both *asynchronous* and *synchronous* modes, and implementations supports a number of activation policies to dynamically manage the (de)activation of services from memory. In addition, TP monitors can be used to provide fault-tolerant, highly available architectures by dynamically performing *replication of processes* and *load balancing* to spread load in times of high demand (Orfali *et al.*, 1999:350-351).

Although the all-or-nothing approach advocated by transactions can be viewed as a feature, Orfali *et al.* (1999:328) also discuss how it can be a hindrance in those transactions where it is acceptable for a partial-rollback to occur where transactions fail to complete or where

transactions can span long periods of time, across companies or the Internet. In addition, transaction middleware does not provide a mechanism to (de)marshal data, so this is left as a task for the developer; whilst transactional middleware also produces a high overhead in cases where ACID principles do not need to be enforced (Emmerich, 2000).

Examples of TP Monitors in common use today include IBM's CICS, IBM/Transarc's Encina and BEA's Tuxedo.

Transaction processing middleware implemented on is rarely suitable for most systems due to the reasons outlined above. Notwithstanding the above, the use of TP Monitors as co-ordinating services in conjunction with other forms of middleware, especially MOMs (see Section 3.1.3.2) and as Object Transaction Monitors (OTMs) within Object Based Middleware (see Section 3.1.3.4) is extremely advantageous. Services TP Monitors may provide include dynamically managing processes such as replica synchronisation, prioritisation, component replication and load balancing between servers and services, as well as ensuring that ACID principles are enforced. In the OTM arena, examples include Microsoft's Transaction Server (integrating with COM), BEA's M3 and IBM's Component Broker Technologies (integrating with CORBA).

3.1.3.2 Message-Oriented Middleware

Message-Oriented Middleware (MOM) supports communication by the passing of messages (packets)(such as events, service requests or responses, etc.) between a networked client and server, via the middleware instead of a dedicated communication channel. A MOM can be conceptualised as a centralised Post Office. Indeed, in this case the Post Office always knows where the individual servers and clients are, so can forward the packets accordingly if they have moved location (Erzberger & Altherr, 1999). This helps the client and server components to be decoupled from each other, providing a degree of location transparency for components. MOMs lend themselves very naturally to architectures where *events* are important – the notification of an *event* is basically the same as distributing a *packet*.

The major advantage of MOM is that the both client and server do not need to be online at the same time (*i.e. time independence*). This is because the MOM accepts the packet and puts it into a queue when the destination component is unavailable. This leaves the original component to be freed to continue with other work. In addition *location independence* is achieved as the client and server are decoupled from each other – they communicate via the MOM – therefore the serving component may be migrated to another part of the architecture whilst the MOM re-routes packets as necessary, all without the knowledge of the client. MOMs very naturally support *asynchronous* message delivery, which leads to systems that are highly scalable. In addition, a high level of reliability can be achieved by implementing temporary but persistent storage within the MOM, especially in cases where the MOM stores the packet where the recipient is offline, until it can be delivered in due course.

This also gives the opportunity for various tasks to be invoked *within* the message queues themselves (such as prioritising certain messages, or adopting a first-in-first-out policy) as the network administrator desires. Depending on reporting requirements, the message queues

could themselves be logged to provide a measure of disaster recovery. (Orfali *et al.*, 1996; Vondrak, 1997).

However, Emmerich (2000) notes that those implementing *synchronous* messaging suffer from the requirement that the synchronisation needs to be manually implemented into the client itself. In addition, although MOMs are scalable in the sense of extensibility, they do suffer from complexities in cases of local communication, where local message queues aren't usually implemented because it simply doesn't make sense (thereby violating *access transparency*, complicating matters for *migration* and *replication* at a later date). MOMs also tend to support *at-least-once* messaging, so the same request could be delivered to a component a number of times. Finally, because of the fundamental messaging nature of MOMs, the ACID properties of *transactions* tend not to be supported especially where there are a large number of distributed recipient components. An example of this could be that there isn't a built-in way to ensure that a particular atomic transaction is executed on either *all* or *none* of the recipients. Orfali *et al* (1999:174) also cite where the lack of support for transactions can result in a number of performance issues. They give an example where the output of one application can be used to veto the entire process taking place (such as someone's credit rating is too low when trying to open a credit card account). This could result in a high number of messages compensating for the cancellation of the original, presenting a high load on the application and network.

The asynchronous approach taken within MOM does degrade performance in situations where synchronous communication could possibly take place (such as in local communication), as there is a further step within the process. Unlike in synchronous messaging, there is no control over the network burden imposed by the middleware solution. However most implementations do default to synchronous message transfer using RPC (see section 3.1.3.3) and fall back to asynchronous mode using MOM if the server or client is unavailable. Unfortunately the majority of MOM available today tends to be implemented as a proprietary product, thereby impeding interoperability and leaving a dependence on the vendor for support and maintenance. (Orfali *et al.*, 1996; Vondrak, 1997).

The literature generally categorises MOM into two types: *message queuing* and *publish-and-subscribe*. The distinction between the two can be thought of as how the producing component knows which components are destined to receive the packet. There are a number of MOM products in the marketplace, they tend to support one or the other type of MOM as depicted in Figure 4. However, Java's Messaging Service⁵ supports both types of MOM and this is being implemented into a number of products such as Softwired's iBUS (Erzberger & Altherr, 1999).

3.1.3.2a Message Queuing MOM

A Message Queuing MOM (Erzberger & Altherr, 1999) is analogous to electronic mail. The producer knows who is/are the intended recipient(s), and delivers the packet to the

⁵ The specification for JMS can be freely downloaded from <http://java.sun.com/products/jms/>.

MOM with the list of recipients. The MOM then delivers a copy of the packet into each recipient's message queue for collection as and when required.

This method of packet delivery is suitable in cases where messages are targeted at specific components, such as in a well defined process where a flow of a message should travel from component to component in a pre-determined sequence. If there are a number of replicas of a component, such as a StartLoanApplication component, then once the message has been delivered to an instance of StartLoanApplication, then it has been 'consumed' – i.e. it is deleted from the MOM to avoid replication of messages.

However, the disadvantage of Message Queue MOMs is highlighted by Emmerich (2000) where he states that these queues are defined by administrators, and their use is often hard-coded into client and server components leading to inflexible and poorly adaptable architectures.

An example of where such a MOM could be used is through a loan processing application where references are required from credit bureaux and data required validation from external sources. (Hurwitz, 1998).

3.1.3.2b Publish-and-Subscribe MOM

Publish-and-Subscribe MOM can be thought of as analogous to Usenet Newsgroups and similar discussion forums found on the internet. Instead of the server component 'pushing' packets to pre-determined clients, this model depends on client components *subscribing* to particular event messages. Once such an *event* takes place, the server passes a packet notifying this to the MOM, which *publishes* this information. The MOM then ensures that all components which previously registered an interest in such an event by *subscribing* to it is notified when they are next available and distributes the packets accordingly.

Publish-and-Subscribe MOM is therefore extremely scalable as a single event is distributed by the MOM to those components that have previously registered their interest. Erzberger & Altherr (1999) discuss how this is more efficient than RPC-based communication, which takes place between with one-to-many and many-to-many components.

It is therefore an event-driven model, as once an event has occurred a message is published to this effect. Subscribing clients who have previously registered an interest in this can therefore download the data for the purposes it desires. Examples of an event include an inventory count falling below the re-order level, the approval of a loan or even the expiry of a digital certificate (Orfali *et al*, 1999:175). Indeed, there is also an example of how such a MOM can be used within a stock exchange, where certain components may register an interest in the change of share prices for a particular organisation (Erzberger & Altherr, 1999).

There are a number of vendors in this market, each with products that satisfy different niches. Orfali *et al* (1999:177) describe some examples, such as Open Horizon's Ambrozia (totally Java based, implements high levels of security and provides

interfaces to C and CORBA), TibCo's Velociti (an internet-based broker which is CORBA compliant and uses IIOP) and Active Software's ActiveWeb (providing adapters for Web Browsers, DBMSs and ERP packages).

3.1.3.3 Procedural Middleware

Procedural Middleware is generally based on Remote Procedure Calls (RPCs). Emmerich (2000) recalls that they were initially devised by Sun in the early 1980s and how they have been subsequently standardised by the X/Open Group⁶ as part of the Distributed Computing Environment. RPCs allow a client component to invoke a server component remotely and have the results returned, both via the middleware. The added value for the programmer is that these invocations are coded in exactly the same way as simply passing a parameter to a procedure call. The programmer no longer has to worry about collecting values for that parameter, marshalling and unmarshalling the data into a message, detailing the required networking functionality, sending the message or calling the procedure at the remote end and handling the reply, as this is all handled transparently by the middleware (Orfali *et al*, 1999:166). RPCs are built into the vast majority of network operating systems available today, including Unix and Microsoft Windows.

The vast majority of RPC implementations are synchronous in nature (even though Hurwitz (1998) quotes some examples of asynchronous RPC in her matrix, Emmerich (2000) notes that asynchronous communication is actually not directly supported by procedural middleware) and support interactions between exactly one client and one server. They are analogous to a telephone call being made by a customer (client) to a bank (server): the client delivers its request and then waits for a response from the server. This requires that both the client and server are online and available at the same time the client makes the request. The client and servers communicate by having 'stubs' included into the client and server tiers at compilation.

The use of RPCs in procedural middleware offer a number of advantages. They offer the highest level of recoverability by the use of logging transactions. As with all the middleware groupings described in this chapter, it offers the opportunity for components to communicate regardless of platform, programming language and location (although the client stub needs to know explicitly the network address of the server's stub. This is usually done using configuration files, an online directory or by hand-coding the address into the application). In addition, the synchronous nature of RPC ensures that the network does not become overloaded, because as RPCs are executed with *at-most-once* semantics, the procedural middleware returns an exception in the event of a failed RPC.

Emmerich (2000) however notes that transactions are not supported by procedural middleware. In addition, the scalability of RPC is limited as replication is not handled by the procedural middleware: this is left as an exercise to the application designer. The other major disadvantage of RPCs is that their implementations are nominally incompatible with other implementations and that there is no single standard for implementing a RPC (Vondrak,

⁶ This specification is available for download from <http://www.opengroup.org/publications/catalog/c706.htm>.

1997b). This also makes RPC a poor proposition in the use of distributed or object-orientated systems.

Returning to our telephone call analogy, the use of Synchronous RPC requires that both the client and server are online and available to process data at the same time. Although this provides a relatively fast response time, it does present problems when the connecting network is unreliable (such as the Internet) as the client is waiting for a response. Failures need to be handled accordingly. If no response is received from the server in a specified period of time, the client will usually block, time out and retry the request. The system must provide a mechanism to avoid executing duplicate requests on the server, whilst providing recovery services if the client becomes unavailable after sending a request. (Orfali *et al*, 1999:167).

Because of these disadvantages of RPC, the majority of RPCs are either implemented by a programmer using a proprietary tool, or are supplied as part of a broader, more encompassing middleware product (Vondrak, 1997b).

3.1.3.4 Object and Component Middleware

Emmerich (2000) describes how the fundamental ideas behind Object and Component Middleware have evolved from RPC (see Section 3.1.3.3) and the evolution of object-oriented programming languages such as Java and C++. Indeed, Orfali *et al*. (1999) note that it is characterised by the extension of object-oriented principles (such as identification of objects through references, inheritance, encapsulation, component-based development, the separation of interfaces from implementation, etc). These extremely powerful concepts should allow the creation of highly flexible, reliable, platform-, vendor-, location-, application-, language-, operating system- and bytecode-independent systems assembled from 'plug-and-play' components.

At the heart of Object and Component Middleware, an Object Request Broker (ORB) manages the communication and data exchange between objects located locally or remotely. The ORB acts as an 'object bus' so the client does not need to know the mechanisms used to communicate with, store or activate the server objects. The ORB itself keeps track of information about each object, such as the interfaces and actual network addresses of all the objects within the system. It can be thought analogous to a telephone exchange - by providing a directory of services and helping to establish connections between clients and these services (Wallnau and Foreman, 1997; Orfali *et al*, 1996; Hurwitz, 1998).

Object and Component Middleware allow objects to hide their implementation details (such as location, host platform, programming language, operating system, etc) from the client, so therefore developers only need to know the interface details of the object. This enables a high level of transparency and interoperability within architectures, whilst maintainability is enhanced as object communication is encapsulated within the ORB, hidden from the developers. Indeed, legacy systems (including DBMS servers) can be easily integrated into the architecture by building a *wrapper* around the legacy component and providing an

interface to advertise the services of that component. Objects communicate by simply providing a call to a service – the ORB is responsible for passing that call to the destination (serving) object, whether local or remote – the location of which does not need to be known by the requesting object. Although there are a number of Object and Component Middleware technologies available, there are two major industry specifications and standards that are in common use today, Microsoft's COM+ and the Object Management Group's CORBA specification. A third possible candidate, Remote Method Invocation in Java, is discussed in Section 3.1.3.4c.

Emmerich (2000) notes that the default method of co-ordination between components is *synchronous*, although support is being included for other methods such as for *deferred synchronous* and *asynchronous* object requests (in CORBA 3.0). In a similar vein to publish-and-subscribe MOMs, broadcast-style group communication is possible through the Event and Notification services provided within CORBA. The middleware also has a number of activation and threading policies that can be implemented as required, such as enforcing sequential or concurrent operations as requested by clients.

In terms of reliability, Emmerich (2000) classifies the default reliability to be *at-most-once* as Component and Object Middleware supports exceptions. (CORBA's messaging and notification services can be used to achieve *exactly-once* reliability). However, interestingly, the middleware embraces and supports the concepts of *transactions* – CORBA provides a transaction service, Java also provides its own transaction services, whilst COM+ is integrated within Microsoft's Transaction Server. Emmerich also notes that, because of the object-orientated nature of the middleware, heterogeneity is supported by the use of a standard IDL-based interface to objects, which itself can be bound to components written in the majority of programming languages. The use of the interface also resolves data heterogeneity can be transferred between multiple platforms⁷. In addition, efforts are underway to enable different ORBs to interoperate – especially in the area of interworking between COM+ and CORBA implementations.

However, Emmerich (2000) cites that some issues still remain with the scalability of Component and Object Middleware in large-scale implementations, although these are being addressed within the newer implementations. For example, CORBA attempts a measure of load balancing by returning object references based on the least-loaded host.

As can be inferred from the above paragraphs, Component and Object Middleware now provides the vast majority of the services, and combines many of the strengths, of the previously discussed types of middleware especially with regards to transactions and messaging. Because of this all-encompassing nature, we discuss the approach undertaken by the currently available specifications of this type of middleware in detail.

3.1.3.4a Common Object Request Broker Architecture

⁷ Java takes a slightly different approach as the client and server objects reside in a *Java Virtual Machine* – freely available implementations of which can be downloaded for free for all platforms from <http://java.sun.com>.

The Common Object Request Broker Architecture (CORBA) has been devised by an 800-strong consortium of academic institutions and software companies representing the entire spectrum of the computer industry. This consortium is called the Object Management Group (OMG) and provides an in-depth information at <http://www.omg.org>.

CORBA is actually a specification for, and not an implementation of, an Object Management Architecture based on an ORB. Orfali *et al.* (1999) propose that it has the potential for subsuming every other form of middleware: 'CORBA uses objects as a unifying metaphor for bringing existing applications to the bus. At the same time, it provides a solid foundation for a component-based future.'

The current release of CORBA⁸ provides a platform for interoperability between components and across ORBs. The interfaces of the objects are written in a standardised programming language and platform-independent language, the Interface Definition Language (IDL). Orfali *et al.* (1999:475) note that CORBA's specification provides mappings for languages such as C, C++, Ada, Smalltalk and Java, with others (Perl, Objective C and JavaScript) currently in the works. This enables access to (e.g. communication, creation, storage, etc) components defined using IDL via the CORBA ORB without needing to worry about the physical location, programming language, platform or operating system used by each component. This is an extremely powerful feature, as it enables the implementation of the object to be completely encapsulated and only be accessed via the methods specified within the IDL-ized interface. The IDL-ized interface also defines the structure of the object and is also used by CORBA's *Interface Repository* to allow objects to discover services provided by other objects and where those objects are located.

CORBA makes use of IDL-defined *stubs* on the client to define the client's *static* interfaces to object services. These stubs are compiled using an IDL compiler at the same time as compilation of the object. This client stub corresponds to an IDL-defined *skeleton* on the server. This allows client objects to simply make a local call to the remote service that it requires, as the client stub acts as a *proxy* for the server object regardless of its location, with the communication and associated services handled by the ORB. The client stub itself also performs a number of tasks such as *marshaling and unmarshaling* complex data structures into a flattened message format to be transferred across the network(s).

CORBA also enables mechanisms such as the *Dynamic Invocation Interface (DII)* for object methods to be *dynamically* invoked at run-time, even if the client object does not have a corresponding stub, or even if the server object does not have IDL-defined skeletons. Although this interface can be very complex to program, it provides a very high level of flexibility by allowing objects to explore and discover what methods are available them during runtime (Orfali *et al.* 1999:484).

⁸ The CORBA specification is freely available and the current version (2.31) is available for download from the OMG (OMG, 1999).

In addition, it is entirely possible, and strongly recommended, to encapsulate existing legacy code (such as COBOL, stored procedures, CICS, etc) with an IDL-based wrapper, making the legacy code look, and act, like an object within the system. In addition, the object-orientated approach taken by CORBA is highly visible with the advent of polymorphic messaging. Unlike static Remote Procedure Calls, this allows a method invocation (such as *configure_yourself*) to behave differently when applied to different objects (such as a database and a printer object). (Orfali *et al.* (1999:p480).

Communication between ORBs has been enabled by the specification of the mandatory General Inter-ORB Protocol (GIOP). Defined in CORBA 2.0, it overcame the initial interoperability problems that surfaced between CORBA 1.0 ORBs implemented by different vendors. GIOP provides a vendor-neutral standard for which communication can be achieved between these ORBs. The most common GIOP is the Internet Inter-ORB Protocol (IIOP) which provides these services over the TCP/IP protocol used on the Internet. Efforts are currently underway, and indeed some implementations provide, interoperability between CORBA's and Microsoft's COM+ ORBs.

However, CORBA is more than just an object bus. It is a very complex architecture, the *Object Management Architecture*, as displayed in Figure 5 so this document will attempt to summarise the additional services provided based on the detailed discussion presented in Orfali *et al.* (1999:489-498).

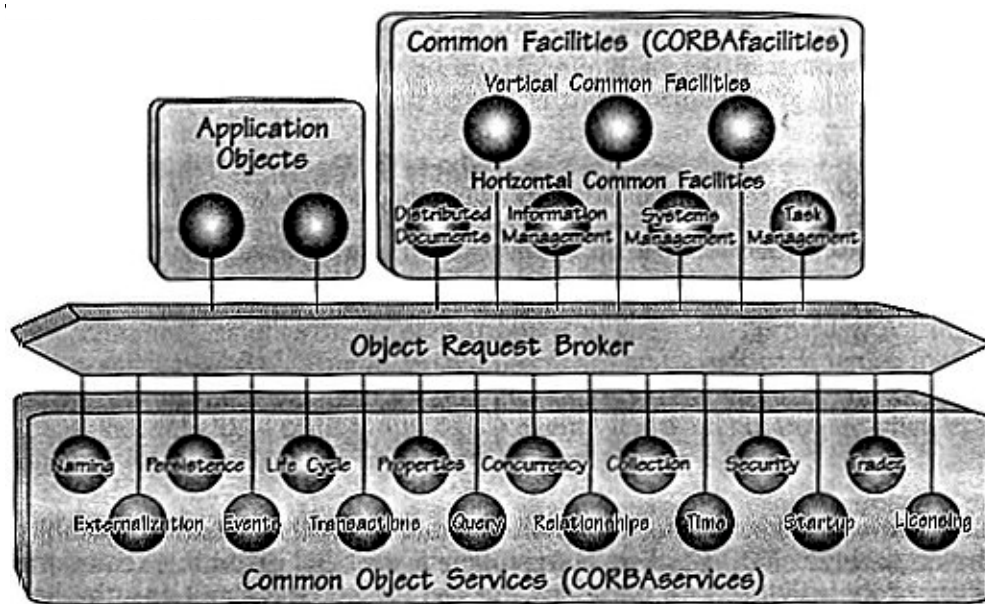


Figure 5: The OMG Object Management Architecture (Orfali *et al.* 1999:p478)

CORBA defines fifteen system-level common object services that complement and enhance the functionality of the ORB. Defined with IDL interfaces, this gives the developer the ability to create, name and introduce such a component into the system environment to utilise the services provided. They are:

- The *Life Cycle Services* handles the component's life cycle on the ORB, including creating, deleting, copying and moving them.
- The *Persistence Service* allows storage of components on a variety of types of server, including relational and object databases, via a single interface.
- *Naming Services* allows objects to locate each other by name. They also provide bindings to existing directories and contexts, such as X.500, LDAP, Novell's NDS, etc.
- The *Event Service* (Event Broker) allows objects to (de)register their interest in specific events dynamically. Distribution to multiple objects is achieved through a well-known *event channel* object.
- *Concurrency Control Services* provides locking mechanisms on behalf of transactions or threads.
- The *Transaction Service* provides two-phase commit co-ordination services for flat and nested transactions.
- The *Relationship Service* allows dynamic relationship association, tracking and integrity checking to take place between components that otherwise know nothing about each other.
- The *Externalisation Service* provides a standard stream-like mechanism for getting data into and out of a component.
- The *Query Service* provides standard SQL-based query operations for objects.
- The *Licensing Service* allows monitoring of usage of components and provides operations for charging for usage at any point within the life cycle.
- The *Properties Service* provides operations for dynamic allocation of properties (named values) to a component.
- The *Time Service* allows synchronisation of time between objects in a distributed environment. It also provides services for defining and managing time-triggered events.
- The *Security Service* provides a framework for authentication, confidentiality, non-repudiation and access control for objects.
- The *Trader Service* provides a directory to allow objects to publicise their services and promote themselves for jobs.
- The *Collection Service* provides interfaces to generically create and manipulate the most common collections.

These services provide a robust environment in which a distributed component can operate.

At the next level of the hierarchy lie the *horizontal common facilities*. These are collections of configurable IDL-defined frameworks that provide services to be used by application objects directly. The aim is to provide a standard set of services for use in system architectures regardless of the application or application domain, to enhance interoperability and to save programmers in different industries from 'reinventing the wheel'. However it can be difficult to implement a specification for such an object that will be accepted at a generic level worldwide. Examples include object categories that deal with User Interface, Information Management, Task Management and System Management. Special interest groups within the OMG are currently working in these areas.

In addition, there are also *vertical common facilities* that provide standards for interoperability in specific vertical markets or industry segments, such as within Oil and Gas Exploration and Production, Imagery, Accounting and Application Development. In a similar vein to the *horizontal facilities*, the aim is to produce a set of base objects to promote interoperability of data and objects within a specific domain, to help avoid companies within an industry 'reinventing the wheel'. Special interest groups within the OMG are also currently working upon these facilities.

The *application objects* refer to those objects that are created on a project or application-specific basis by the programmer.

CORBA seems to be evolving at a very fast rate and it is difficult to keep up to speed with the evolution. Orfali *et al.* (1999:502) believe that the development of CORBA 3.0 is highly influenced by the role of the Internet and the Java Object Model (with extensions for multiple-language support). Examples of planned enhancements include the provision of Java/RMI-to-IDL mappings (to produce IDL stubs and skeletons from RMI in Java (see Section 3.1.3.4c), support for multiple interfaces and objects-by-value, and the adoption of Business Object capabilities by introducing the CORBA Component Model (a slight variation of the Enterprise JavaBeans model). It should also enhance the portability of server objects by the provision of a Portable Object Adapter to complement the existing Basic Object Adapter, whilst also providing a method for allowing IIOP traffic through corporate firewalls.

In his detailed paper describing some of the new features that will be available in the CORBA 3.0 specification, Vinoski (1998) notes that the provision of synchronous message passing only in previous versions was 'an obvious hole in the CORBA specification'. Consequently, CORBA 3.0 provides MOM-like capabilities by introducing *asynchronous messaging*, support for *time-independent invocation* and allowing developers to specifying a *quality of service* for message delivery for any request.

At the time of writing, the OMG website⁹ quotes that CORBA 3.0 should be adopted as a formal specification shortly, most likely in mid-2000, after a few technical issues have been clarified.

Orfali *et al.* (1999:543) note that CORBA ORBs are available for most client systems at practically no cost. They are usually included within a wide variety of operating systems (such as Linux, Unix, NetWare 5.x and Silicon Graphics) with the notable exception of those written by Microsoft. However, a version is also included within Netscape's freely available web browsers and a Java/CORBA ORB is available from Sun as part of the Java 2 plug-in for other web browsers. In addition there are a wide variety of free (Open Source) and commercial ORBs available.¹⁰

Products that implement some of the services that will be specified in CORBA 3.0 are starting to emerge onto the market, even though the final specification has not been adopted as yet by the OMG. Examples include IONA's Orbix 2000 product suite that is currently in the beta-testing stage¹¹.

The major advantage of CORBA is that components, modelled as Business Objects, provides a potentially excellent fit for three tier architectures distributed across intranets and the Internet. The use of IDL wrappers allows existing applications to be encapsulated as objects, ensuring an evolutionary approach to system development rather than having to start from scratch. CORBA is also a neutral specification without dependencies on particular technologies or languages, although CORBA 3.0 is taking many of its Object Model concepts from those already present in Java. In addition, the ability to statically and dynamically invoke methods on objects, and the use of IIOP to enable interoperability via ORBs connected through the Internet, makes the architecture quite powerful for distributed and evolving systems. It has been developed over a number of years by an industry consortium as an 'open' standard.

However, as previously mentioned, CORBA is extremely complex and is not an implementation. Wallnau (1997) cites this as a major disadvantage of CORBA - the 'lack of a predefined compliance test suite' and the 'variability of vendor implementation choices' makes it difficult to assess vendor implementations' conformance to the CORBA standard, or even to which version of the standard they conform to. In many cases, for example, implementations tend not to include the complete set of system-level services (Yee, 1999). Even with the introduction of the IIOP, vendor enhancements to their ORBs may impede interoperability. In addition, a CORBA client needs to be installed and configured onto every machine to be able to access the services provided by the architecture, whilst the product suites encompassing CORBA clients and servers can become quite expensive.

⁹ <http://www.omg.org/techprocess/meetings/schedule/tech2a.html> (Cited 8 February 2000)

¹⁰ Relatively up-to-date matrices comparing free and commercially available ORBs can be found at <http://www.vex.net/~ben/corba/>.

¹¹ <http://www.iona.com/products/iPortal/index.html>.

3.1.3.4b Microsoft's Component Object Model Plus

Although Microsoft¹² is a member of the OMG, its behaviour suggests that it doesn't officially support attempts to make software interoperate between various platforms (such as CORBA and Java) as this would undoubtedly loosen its grip on the operating system market. Consequently, it has specified its own Component Object Model Plus (COM+) which has evolved over the years from its roots in Object Linking and Embedding (OLE), a set of libraries and applications for storage, data exchange, and integration of elements within compound documents. The Component Object Model (COM) evolved from OLE as an object-encapsulation technology, standardising the format of executable files and thereby solving the issue of interaction between executable code written in different programming languages; which then evolved into Distributed COM (DCOM) which allowed COM objects to communicate across a network. A minimalist COM object designed for the Internet became known as an *ActiveX* object. COM and DCOM then evolved into a further specification, COM+, which vastly improved and integrated the following further into the operating system:

- Microsoft's Transaction Server (MTS) as an Object Transaction Monitor (OTM). MTS seems to have been significantly improved over previous versions and now also supports publish-and-subscribe capabilities.
- Microsoft's Message Queue (MSMQ) to provide MOM-like asynchronous messaging capabilities.

These evolutionary specifications also coincided with their implementation into Microsoft's operating systems, for example COM and DCOM was implemented as part of Windows 98 and NT 4.0, and COM+ is implemented as an integral part of their recently-launched Windows 2000 product line. Throughout this document, we refer to COM+ as an umbrella term to encompass all the technologies described above, unless otherwise stated.

It is acknowledged by Orfali *et al.* (1999:525-6) that what was originally planned for COM+ is a lot different to what was actually delivered within Windows 2000, the only implementation of COM+. However, in the absence of detailed specifications for the COM+ model, the OSMOS Consortium have found difficulty in further exploration of what COM+ actually is and the features it provides.

The literature seems to suggest that there is very little difference between the goal that the standardised CORBA and proprietary COM+ models intend to achieve, the differences between the two stem from the way they are specified and implemented. Indeed, both models facilitate the client-server model by supplying a framework for objects to interact regardless of location. Whilst CORBA facilitates communication by

¹² Further information about Microsoft can be found at <http://www.microsoft.com>.

the use of an ORB, communication between COM+ components is done by Microsoft's version of Remote Procedure Calls (MS-RPC). Both models require that an object has an implementation (in whatever language) and an interface specified in a Microsoft's IDL (incompatible with CORBA IDL). In CORBA, objects and their interfaces are registered in an interface repository accessible to the ORB, whilst in COM+ objects locate each other in a similar way as their interfaces are registered within the server (within the system registry or Active Directory, depending on which of Microsoft's operating systems is used) at installation (servers hosting COM+ objects must be registered with a library). From a practical standpoint, there is little difference in how objects are created in each model.

COM+ does have a specific advantage in that Microsoft have implemented it as part of their products, rather than leaving vendors to implement their understanding of a specification. Experience with CORBA has found that vendor implementations do suffer some interoperability problems with each other's products. However, this isn't a problem with COM+ as there is only one vendor's implementation, Microsoft. Microsoft has kept control of the specification by not offering it as an open standard to the industry, but sells licences to other companies if they wish to offer COM+-based services in their products. Consequently, Microsoft has not offered COM+ as an open standard, or allowed implementation of COM+ in other any operating system¹³. The literature seems to suggest that if you wish to use COM+, you must use Microsoft's (latest) operating systems for all your servers and the majority of your clients.

Regarding the openness and availability of the standard, the company did make attempts in this area too. Wallnau & Foreman (1997) noted that the DCOM specification were turned over to the Open Group but the outcome of this process is still unclear.

The Consortium has found difficulty when researching into the architecture of COM+, as it mostly came around through the aforementioned evolutionary approach. Documentation about the architecture as it stands today tends to be scattered across the Microsoft website in an incoherent fashion^{14,15}. Indeed, the lack of a defined and integrated architecture is cited by Emmerich & Ellmer (1998) as one of its major deficiencies. In addition, Morris and Litvak (1997) cite that the distinctions between Microsoft's technologies and platforms are often blurred. For example, the COM+ does

¹³ However, Microsoft has licensed a number of software houses to simply port (D)COM onto other platforms. Yee (1999) notes that although versions are available on on AIX, HP-UX, Linux, OpenVMS, OS/390, Solaris, and Tru64 UNIX, broad acceptance seems to be some way off. Indeed, Yee (1999) agrees with the literature currently available, hypothesising that COM+ will not be made available on other platforms, stating that interoperability 'on the wire' will still be possible using DCOM, however COM+'s additional features such as messaging will not be available on other platforms.

¹⁴ The majority of references to 'architecture' relate to Microsoft's Distributed Network Architecture (DNA), which seems to be a marketing exercise to combine sales of Microsoft's operating systems, back-office systems and development tools.

¹⁵ The specifications that are available at <http://www.microsoft.com/com/resources/specs.asp> are early draft versions with no indication of whether these are the final specifications or otherwise.

not explicitly provide many system-level services as provided by the CORBA specification, but as it is highly integrated into the Windows operating system, COM+ implicitly makes use of Windows' underlying services, for example security and naming (Yee, 1999). This blurring makes it extremely difficult to separate the model from the underlying operating system architecture.

Orfali *et al.* (1999:506) acknowledge that COM+ provides some of the same functions as CORBA, but the terminology, and the implementation, is done in a different way. By way of an example, they feel that

A COM object is not an object in the OO sense. Unlike CORBA, COM objects do not have a persistent identity. A COM object reference is simply an interface pointer to an object in memory. If the object is released from memory, you cannot use its reference to access it again at a later time. [...] In other words, COM objects do not maintain state between connections. This can be a big problem in environments where you have faulty connections - for example, the Internet.

(Orfali et al (1999:507))

Although this is strictly true, an object's state can be maintained by the use of *monikers*. A moniker is an object that acts as a 'persistent alias name for another object' (Orfali *et al.*, 1999:507). COM+ will then allow the association of a COM+ object (which is by nature transient and stateless) with a context via the use of such a moniker.

Morris and Litvak (1997) define COM+ as 'a binary compatibility specification and associated implementation that allows clients to invoke services provided by [COM+]-compliant components ([COM+] objects).' This is achieved via a set of interfaces (methods) on the object, invoked at run-time. As with CORBA, the actual language that the components and clients are coded in is irrelevant, as long as the compiler can provide the relevant language bindings at the binary level. Orfali *et al.* (1999:508) note that Microsoft have provided this in all their development environments, such as Microsoft Visual C++ and Visual Studio, with some other vendors doing likewise. COM+ and DCOM extended COM to be able to run over distributed networks as well as within one machine by using Remote Procedure Calls.

(D)COM was been implemented as an integral part of Microsoft's Windows 95, 98, NT4 and 2000 operating systems. COM+ will be implemented as part of Windows 2000, providing further integration between the existing transaction services and message queuing/publish-and-subscribe services (Orfali *et al.*, 1999:525). The vast majority of applications implemented on PCs make use (D)COM to some extent and it is on Microsoft's platforms that the model is most mature (Morris and Litvak, 1997). It is highly likely that applications developed on the Windows 2000 platform will make use of COM+ in a similar way. However, because COM+ is based on a native binary format, objects written on one platform will need to be recompiled when ported to a different platform. This brings into play the classical pros and cons of interpreted and compiled languages. Morris and Litvak (1997) feel the binary format usage can be an advantage (using the capabilities of the platform to the full and faster execution) but

also a disadvantage (ActiveX controls implemented on the Internet are not machine independent¹⁶, unlike its 'competitor', Java that provides virtual machines for all platforms).

As previously mentioned, DCOM is most mature on Microsoft's Windows range of operating systems. Although some third parties have recently attempted to implement DCOM on other platforms (such as UNIX and Macintosh) the technology has not had a long enough period of time to prove its robustness in heterogeneous environments. Morris and Litvak (1997) recommend that DCOM is simply 'best applied in environments that are primarily Windows based.' Indeed, Orfali *et al.* (1999:525) note that the services provided by COM+ will only be available within the Windows 2000 platform.

3.1.3.4c Java-based Technologies

Java¹⁷ is not a middleware technology. It is, in fact, a freely available, open, platform-independent, object-orientated programming language (and set of APIs) devised by Sun Microsystems. Java applications (*objects*) can be written on one platform and will theoretically run without modification on any other as long as a 'Java Virtual Machine' is available on that platform. Java Virtual Machines are available on most platforms and operating systems, and convert Java code into the native bytecode of the machine it is executed on (Sun, 1999).

However, it has started to provide some ORB-like features as part of the language, and because of its heavy influence onto CORBA 3.0, the OSMOS Consortium feels they are worthy of discussion here. These include features such as transaction management (Java Transaction Services), a name-to-object mapping service (as part of Remote Method Invocation), publish-and-subscribe capabilities (Distributed Events and Java's Messaging Services) and connectivity to databases via SQL (Java DataBase Connectivity). The Java Naming and Directory Interface¹⁸ also provides a standard APIs for developers, allowing them to develop Java applications which can store and access information and objects in a number of directories using widely-used protocols, such as LDAP (for X.500 directories), NDS (for Novell's directory products) and the RMI Registry (see below).

Java's Remote Method Invocation (RMI) can be thought of as a Java-based cut-down ORB. It is an integral part of the Java Virtual Machine and language, sharing the Java object model. It allows Java objects to communicate with, and execute, other Java objects remotely across a network, intranet or Internet regardless of platform (as they all

¹⁶ ActiveX controls are designed to be executed within Microsoft's Internet Explorer web browser, and are not directly supported by others (such as Netscape Communicator) without the use of third-party plug-ins. However, regardless of the browser used, they require that the operating system used is Microsoft Windows.

¹⁷ Detailed information about Java can be found at <http://java.sun.com>.

¹⁸ JNDI Specifications and further information can be found at <http://java.sun.com/products/jndi/docs.html>.

run JVMs) as if they were local, using the Java Remote Method Protocol. Wallnau & Foreman (1997) propose that this 'provides ORB-like capabilities as a native extension of Java'. This includes all the basic services provided within Java, such as security.

Raj (1998) nicely summarises how Java objects locate each other using RMI.

Each Java/RMI Server object defines an interface which can be used to access the server object outside of the current Java Virtual Machine (JVM) and on another machine's JVM. The interface exposes a set of methods which are indicative of the services offered by the server object. For a client to locate a server object for the first time, RMI depends on a naming mechanism called an RMIRegistry that runs on the Server machine and holds information about available Server Objects. A Java/RMI client acquires an object reference to a Java/RMI server object by doing a lookup for a Server Object reference and invokes methods on the Server Object as if the Java/RMI server object resided in the client's address space. Java/RMI server objects are named using URLs and for a client to acquire a server object reference, it should specify the URL of the server object as you would with the URL to a HTML page.

(Raj, 1998)

The main disadvantage of this approach is that the use of RMI doesn't completely hide the underlying networking issues from the developer. In addition, it also relies on both objects having been coded using Java. However, Orfali *et al.* (1999) note that these issues can be overcome by using Java in conjunction with a complementary technology, CORBA. They view CORBA as a technology which provides network transparency whilst Java provides implementation transparency. Indeed, efforts have been made by Sun and OMG to bring these two technologies together (Sun, 1997).

The current release of Java (version 1.2, branded as "Java 2") has implemented the ability for Java objects to communicate with other components (written in different languages) using a subset of RMI over CORBA's IIOP¹⁹. In addition, a CORBA-compliant ORB has also been included as part of the Java platform, giving the ability for developers to create CORBA interfaces using IDL from within Java. Java objects can now therefore be implemented as part of a CORBA architecture and communicate with non-Java objects, without the need for knowledge of the underlying networking issues.

3.1.3.4d Comparison between CORBA, DCOM, JAVA / RMI

It is difficult to compare the merits of the three object-orientated middleware approaches described above, especially as one (CORBA) is an open specification with a number of implementations of varying quality, whilst the remainder (DCOM and RMI) are specifications and proprietary implementations by single companies (although Java's

¹⁹ Further technical information can be found at <http://java.sun.com/products/rmi-iiop/index.html>.

RMI is part of a fairly open language). Emmerich & Ellmer (1998) attempted such a comparison and concluded that from a functional perspective, CORBA seems to be the most advanced, although this was an unfair comparison for the reason previously stated. Orfali *et al* (1999) analyses how the market for object-orientated middleware seems to have split into two camps: the industry-based CORBA/Java/RMI model versus Microsoft's proprietary COM/ActiveX model, emphasising the relative advantages and disadvantages of each approach. However, a comparison of the models with respect to the requirements of OSMOS will be made in Section 3.1.4.

3.1.4 Comparison of the Potential IT Solutions That Support the Technical Requirements of the Virtual Enterprise

This section of the document will discuss some of the issues that need to be considered at a generic level for effective Client/Server computing. Although examples will be given to illustrate how these issues have been dealt with within specific technologies, these issues should be carefully considered when discussing any proposed technological implementation within OSMOS.

3.1.4.1 Network Architecture

The OSMOS architecture is reliant on providing *Internet-based* services for the Construction industry, so therefore the Consortium feels it would not be advantageous to compare and contrast the requirements against LANs, WANs, Intranets and the Internet. However, we because we acknowledge that the OSMOS solution must be designed with a viewpoint to the solution being deployed on the Internet, it will be automatically be implementable on Intranets, LANs and WANs with the use of Internet-based technologies (as discussed in Section 3.1.1.4).

3.1.4.2 System Architecture

Similarly, the Consortium feel that it would not be particularly advantageous to analyse two tier and three tier architectures against the requirements as set out in Chapter 2, as this comparison has previously been done in considerable depth by others (such as Gallagher & Ramanathan, 1996; and within Section 3.1.2 of this document). Indeed, the inclusion of middleware technologies (see Section 3.1.3) in the proposed solutions (see Chapter 4) necessitates that a three tier architecture is chosen, as the middleware that is used is the middle tier of the three tiers.

3.1.4.3 Middleware Technologies

The following table gives a brief overview of the various middleware technologies, concentrating on how well they answer the requirements as set out in Chapter 2. However, the reader must appreciate that this is an evolutionary area and is correct only at the time of writing. The Consortium would also like the reader to refer to Section 3.1.3 for detailed discussion regarding these middleware technologies.

KEY ✓ Yes ✗ No ? Unknown ²⁰ ◆ Varies ²¹ - N/A	Distributed Object Communication Principles			Message-Oriented Middleware		Remote Procedure Calls	Object/Component-based Middleware Platforms		
	CORBA	DCOM	RMI	Message Queuing	Publish & Subscribe		CORBA Components	COM+	Enterprise JavaBeans
Transactional	✓ (using Transaction Service)	✗	✗	✓	✓	✗ (✓ if used with a TP monitor)	✓ (integrated)	✓ (integrated)	✓ (integrated)
Platform independence / interoperability	✓ (Uses IIOP)	✗ (Requires Microsoft Windows)	✓ (Uses JMRP & IIOP)	✓	✓	✓	✓ (Uses IIOP)	✗ (Requires Windows 2000)	✓ (Uses JMRP & IIOP)
Language independence/ interoperability	✓	✓	✗ (limited interoperability via IIOP)	✓	✓	✗	✓	✓	✗ (limited interoperability via IIOP)
Management of huge set of data	✓	✓	✓	?	?	✗	✓	✓	✓
Distribution support	✓	✓	✓	✓	✓	✓	✓	✓	✓
Communication flexibility	✓	?	?	?	?	✗	✓	?	?
Schema versioning (Interface evolution)	✗ (Possible, but difficult to achieve)	✗ (Possible, but difficult to achieve)	✗ (Possible, but difficult to achieve)	✓	✓	✗	✗ (Possible, but difficult to achieve)	✗ (Possible, but difficult to achieve)	✗ (Possible, but difficult to achieve)
Support For Multiple Communication protocols	✓ (IIOP)	✓	JRMP IIOP RMI	✓	✓	✓	✓ (IIOP)	✓	IIOP JRMP RMI
Communication types	Synchronous	Synchronous	Synchronous	Asynchronous	Synchronous	Synchronous	Synchronous	Synchronous	Synchronous
Code Portability	✓	✗ (Tied to MS Windows)	✗ (Java Only)	◆	◆	✗	✓	✗ (Tied to MS Windows)	✗ (Java Only)
Ability to Define Security levels	✓ (using Security Service)	✓ (using Windows Services)	✓ (using Java Policies)	◆	◆	✗	✓ (using Security Service and Configurations)	✓ (using Windows Services)	✓ (using Java Policies and Configurations)
Standardisation	✓	✓	✓	?	?	✓	✓	✓	✓
Openness of Standards & Standardisation Process	✓ (OMG)	✗ (Microsoft Proprietary)	✓ (Sun Proprietary, with input from others)	?	?	✓ (OSF)	✓ (OMG)	✗ (Microsoft Proprietary)	✓ (Sun Proprietary, with input from others)

²⁰ Unknown at this present time, although research into this is continuing.

²¹ The taxonomy presents a generic comparison between the *types* of middleware. It does not compare specific vendor implementations of the middleware. For cases where some vendors provide this service and others don't, a diamond symbol denotes that this varies between implementations.

KEY	Distributed Object Communication Principles			Message-Oriented Middleware		Remote Procedure Calls	Object/Component-based Middleware Platforms		
	CORBA	DCOM	RMI	Message Queuing	Publish & Subscribe		CORBA Components	COM+	Enterprise JavaBeans
✓ Yes									
✗ No									
? Unknown ²⁰									
◆ Varies ²¹									
- N/A									
End user enterprise applications integration	◆ (more or less strongly invasive)	◆	◆	-	-	-	◆	◆	◆
Quality of service	◆	✓	◆	◆	◆	✓	◆	✓	◆
Any data types or specific types	✓	✓	✓	✓	✓	✓	✓	✓	✓
Object oriented	✓	✗	✓	◆	◆	✗	✓	✗	✓
Description and specification	✓ (IDL)	✓ (MIDL)	✓ (Java)	✗	✗	✓ (RPC gen)	✓ (IDL / CIDL)	✓ (MIDL)	✓ (Java)
Code mobility	✓	✗	✓	◆	◆	✗	✓ (OBV)	✗	✓

Table 1 A Comparison of Middleware Technologies

3.2 Security Considerations

There are a number of issues that need to be addressed relating to security of data being transferred across networks. This is especially apparent when the data is transferred across untrusted and inherently insecure networks such as the Internet.

In this section, 'data' is defined as any form of electronic information being stored on, or transferred between, computers. This includes databases, files and ad hoc information created by the user, such as passwords, payment instructions, bank details, etc.

3.2.1.1a Data Confidentiality

The majority of data that is transferred across a network is transmitted 'in the clear'. This is where the data is transferred between machines in the same binary format that it was created and intended to be processed. This data can be easily intercepted by a malicious individual by using 'sniffer' software which monitors network connections and takes a carbon-copy of all messages (packets) being transferred. It is then a relatively trivial task to reassemble these packets into the original form, for use by the individual. In addition, because the packets have been copied, the sender and receiver are totally unaware the data has actually been intercepted!

This risk can be minimised by the use of cryptography. This is where data is scrambled (*encrypted*) into an unreadable format (*cipher text*) before being stored or transmitted.

This is performed with the use of a cryptographic algorithm (which is usually published and is scrutinised by mathematicians and computer scientists for its robustness) and with a key (a random sequence of characters that is kept secret). To unscramble (*decrypt*) the data both the algorithm and the key must be known - knowledge of one without the other will not help in the decryption of the message.

There are a wide variety of algorithms that could possibly be used. Common knowledge within the security industry suggests that those algorithms which have been published publicly, and have therefore been subject to intense scrutiny by mathematicians and hackers alike, tend to be the most secure if there have been no flaws reported in the algorithm. Examples of these algorithms include IDEA, RSA and TripleDES.

However, the key that is used in conjunction with the algorithm is kept secret and usually randomly generated by a computer. A key is usually a random sequence of characters of predetermined length. Examples of key lengths will be given in the next couple of paragraphs, where *weak* denotes encryption that can be cracked relatively easily by brute force (i.e. by trying each possible combinations of keys in turn) by organisations and governments, whereas *strong* denotes encryption strength which (allegedly) worries governments because it would take them more seconds than the world has existed, even using the most powerful of computers available today. Of course, the phenomenal advances that have occurred in computing power in recent decades means that stronger encryption methods are continually being developed and tested.

Keys are usually stored in a (relatively insecure) medium, such as on a computer's hard drive. To help protect them against hackers and computer viruses, they are usually encrypted themselves and protected by an authentication mechanism such as a passphrases (a password that includes a mixture of numbers, characters and other characters to avoid 'dictionary attacks'). Newer technologies enable the keys to be stored on swipe cards or smart cards, and even access to the key can be controlled with the use of biometric devices such as fingerprint and voice recognition technology.

In *symmetric (shared private) key encryption* the same key, which is kept secret, is used to encrypt and decrypt the message. Obviously, the both the sender and receiver must have a copy of this key. Key lengths used generally range from 40-bit (*weak*) to 256-bit (*strong* by today's standards) and above. This enables fast encryption and decryption of message, although the major problem with this method is how to distribute the key, especially across the Internet, to both parties without it being intercepted en route. This makes the symmetric key approach difficult to scale. Examples of the algorithms used here include IDEA, TripleDES and DES.

In *asymmetric (public) key encryption* two different keys work together as a pair. One key, a *public key*, can be used to encrypt a message whilst its corresponding *private key* is the only key that can decrypt that message. The keys are related so that it is mathematically impossible to create a private key from a public key. Therefore, the public key is made available (usually in an X.500-based directory) to all those who would use it, whilst the private key is kept secret. The complex relationship between the keys requires that the key lengths are very long, typically ranging from 512-bit (*weak*)

to 1536-bit (strong) and above. The advantage of this method is that both parties need not have prior knowledge of the same secret key, as is required with symmetric encryption. It is therefore scalable across insecure worldwide networks such as the Internet. The disadvantage of public key encryption is that it is over 1000 times slower than symmetric encryption to encrypt large messages. In addition, the private key must be kept private otherwise the confidentiality is broken. RSA is the most popular algorithm using this technology.

A number of software implementations, such as Pretty Good Privacy²², attempt to overcome the speed disadvantage of public key encryption and the key distribution problem of symmetric encryption by combining both technologies. Here a dynamically generated symmetric key is used to encrypt a file. The symmetric key is also encrypted using the recipient's public key. This, and the aforementioned file, can now be safely sent across the Internet, where the recipient is the only person in the world who has the corresponding private key. This key is used to decrypt the encrypted symmetric key, which is used to decrypt the message. This seems like a complex process - but one that can be automated within systems.

Whichever form or algorithm is used, there is a performance overhead involved with encrypting and decrypting messages. The general rule is that the longer the key used, the higher the performance overhead on the system. In addition some countries prohibit the use, or export, of encryption above certain key lengths for reasons of national security²³.

3.2.1.1b Authentication

Authentication is the process where a client (or person) proves to a system that it is who it purports to be. This can be generally done by the client proving itself by disclosing:

- Something it is (e.g. a human may use a fingerprint, voice recognition, etc)
- Something it knows (such as a username and password or passphrase)
- Something it has (e.g. a human may have a token (smartcard), hardware or software key, etc.)

The most secure systems usually use a combination of the above, for PGP uses a passphrase to protect access to the software private key. The key itself could also be held on a smartcard instead of the computer's disk for added security.

²² PGP is freely available from <http://www.pgpi.com>. It is commonly used to encrypt and decrypt data sent using electronic mail via the Internet.

²³ This is because *encryption* is classed as a military weapon (a *munition*) in a number of countries as part of the Wassenaar Arrangement. The situation is changing on a daily basis and a survey can be found at <http://www.epic.org/reports/crypto1999.html>.

In addition, one major problem with messages received over untrusted networks (such as the Internet) is answering the question: "How can I tell that this message really came from the sender whom it claims to originate?"

Firstly, a simple authentication method is for an object (or person) to send a message encrypted with their *private key*. Anyone with access to the corresponding *public key* can decrypt that message - but as the private key is known only to the sender, then the message must have been sent by the owner of the private key. The Kerberos system (Orfali *et al*, 1999:142) is a method used for distributing *asymmetric* keys to servers, although it doesn't scale well to large implementations.

A second method is to use a *digital signature*. Digital signatures are created by appending a date and time stamp to the message, hashing the message (compressing it to a unique fixed-length message) using an algorithm such as SHA-1 and MD-5, and then encrypting the resulting hash with the sender's private key. This digital signature is then attached to the message and sent. At the receiving end, the receiver uses the sender's public key to decrypt the signature, creating a temporary copy of the message and compares the received message with the temporary copy. If they are equal, this also proves that the message has not been altered en route. As nobody else has access to the private key that was used with the hash, the message can be deemed to have been sent by that sender at the stated date and time and has not been modified. (In addition, confidentiality can be achieved by the sender encrypting the entire message (including signature) with the public key of the receiver, thereby ensuring that only the receiver can decrypt the message). The Member States of the European Community are currently passing legislation to give digital signatures the same legal status as a hand-written signature in a court of law.

How does a receiver know that a public key is related to the private key owned by the person he thinks it is? This is where Certificate Authorities (CAs) come into the picture.

Certification Authorities (CA) are used to certify that a public/private key pair belongs to the person it says it does. A CA is usually a body that can be trusted by any number of people, such as the author of this document, the government, a Bank, the Post Office, private companies such as VeriSign and Thawte, the organisation to which one a party belongs, etc. Once a CA has verified that a public/private key pair does in fact belong to the subject it claims to, they create a (usually X.509) *digital certificate* that usually contains the following:

- An identification number of the certificate;
- The *public key* of the subject owning the key pair;
- The date and time that the verification was made;
- Information regarding the level of trust that is attested by this certificate (e.g. it only binds the key to an email address, server, object or job description);
- The date and time that the certificate expires;
- Other relevant information about the subject (e.g. name, email address, X.500 address, etc);
- etc.

The certificate is also *digitally signed* by the CA to ensure that it isn't tampered with while it is stored in a directory or transferred across the Internet.

3.2.1.1c Authorisation

Once the user or object has been authenticated, the system will also need to know what *permissions* the user or object has, i.e. what information it can access, invoke, update, delete, copy, etc; and which peripherals and network services that the actor has permission to use. The system usually finds this from an *access control list* (ACLs) that lists the user (and group) names and the types of operation and data they should have access to. The vast majority of (network) operating systems such as Novell NetWare, Unix, Linux and Microsoft's Windows NT/2000 have implement such features.

3.2.1.1d Non-Repudiation

Non-repudiation is the ability to provide irrefutable evidence that an action took place, i.e. that two (or more) parties were involved in a transaction. Orfali *et al* (1999:143) cites that system which supports non-repudiation must satisfy the following services, as defined by the ISO non-repudiation model:

- An *evidence store* (a long-term storage facility to securely store the following:)
- *Evidence of message creation* (such as a *proof-of-origin certificate*)
- *Evidence of message receipt* (such as a *proof-of-receipt certificate* created and sent by the recipient)
- An *action timestamp* (to record the date and time of the above)
- An *adjudicator* (to arbitrate disputes based on the evidence stored in the storage facility)

A time-stamped digital signature used with a public key cryptosystem (see Section 3.2.1.1b above) can be thought of as a simple non-repudiation implementation. This can be augmented by secure logs to provide evidence in case of a dispute. However in the case of electronic mail, some protocols (such as the widely used SMTP protocol) do not support non-repudiation, whilst others (including X.400) do.

3.2.1.1e Data Integrity

Data integrity is required to show that data has not been modified without authorisation during storage and transmission across networks. It is especially useful when arbitrating cases of non-repudiation. The vast majority of transport protocols, such as TCP/IP, attempt to ensure that data remains integral (at the bit level) during transmission by the use of cryptographic checksums and hashing algorithms, however there needs to be a mechanism to ensure that stored data is not modified and have its integrity verified when required.

This can be achieved with the use of digital signatures, as discussed in section 3.2.1.1b. If the message, or file, has been changed, then when verifying the signature the message

the decrypted hash recreates will not match the modified file. The same occurs if the signature itself has been modified, as without the key used when hashing the intruder will not be able to update both the file and the signature. Comparing the checksum of the decrypted hash with the message can increase the verification performance.

The 'intruder' mentioned here includes attacks by computer viruses as well as a human attacker.

3.2.1.1f Firewalls

An excellent definition of a firewall follows:

A system or group of systems that enforces an access control policy between two networks. The actual means by which this is accomplished varies widely, but in principle, the firewall can be thought of as a pair of mechanisms: one which exists to block traffic, and the other which exists to permit traffic.

(Curtin & Ranum, 1999)

Firewalls tend to be used as a line of defence when connecting trusted networks to untrusted networks, such as an organisational LAN/Intranet to the Internet. They are essential in enforcing the organisation's security and access control policies - especially as there are a large number of hackers that use the Internet to try and obtain confidential data from corporate networks for their own personal gain.

Firewalls are highly configurable. They can be set to allow or deny access to certain machines, IP addresses, network services, servers, protocols and port numbers in either direction. For example a firewall may be set up to block any messages sent to the Internet using an email protocol (such as SMTP, POP3, X.400, etc) unless it has come from a pre-configured source, such as the IP address of the company's Mail Server. A multitude of protocols and services (such as ping, telnet, ftp, rlogin, etc.) can be denied access from either, or both, directions depending on the requirements of the organisations. Firewalls also usually provide details of every message passed and blocked (such as the sender and receiver's IP address, date, time, etc.) by audit trail (see Section 3.2.1.1g below) to allow the organisation to investigate potential lapses in security and take corrective action.

Advanced firewall systems allow for a *demilitarised zone*, where systems such as Web Servers can be hosted to allow access by Internet users, but deny those users access to the corporate networks. In addition this zone can be used for complementing security services, such as automatic virus scanning, scanning packets for non-recommended keywords such as 'internal, confidential' and 'pornography', etc.

However, firewalls tend to introduce a performance bottleneck at the point of connection between the two networks, as they check every single message passed between the trusted and untrusted networks. This can result in slower download times for a user on the trusted network downloading data from the untrusted Internet.

Firewalls are also extremely complex systems to set up and maintain. A firewall isn't a 'plug-and-play' technology: it only can be used as part of an IT solution to help implement the security policy as defined by the organisation. Firewalls do not provide protection from attacks on systems from within the organisation, or from *social engineering* attacks. They are a simply technical solution that addresses only one part of an overall security policy.

Some protocols need to be adapted to work in conjunction with firewalls. Technology that allows this to happen is usually called *tunnelling technology*. For example, the Hypertext Transfer Protocol (HTTP) works with the vast majority of firewalls, leaving the organisation the choice whether to accept or deny traffic. However, protocols such as CORBA's Internet Inter-ORB Protocol are still being worked upon to offer this service. This is in fact one of the features that will be available in the upcoming CORBA 3.0 specification (see Section 3.1.3.4a).

There are a wide variety of vendors that provide firewalls, in software, hardware and complete system form. Different firewall systems provide different levels of configurability and support. A list of commercial, free and shareware firewall vendors can be found at <http://www.waterw.com/~manowar/vendor.html>.

3.2.1.1g Audit Trails

An audit trail allows system managers to monitor activities on a network. These include activities that are successful and those that are unsuccessful such as attempted (but failed) logons (Orfali *et al*, 1999:142). In a similar vein to authorisation (as introduced in Section 3.2.1.1c), the vast majority of (network) operating systems such as Novell NetWare and Microsoft Windows NT/2000 offers the ability for the Network Administrator to log all, or some of, the operations carried out on a system. The more advanced systems also allow administrators to query the trail to monitor the activities of a certain object, individual, those performed on a particular server, etc. Audit trails should be archived indefinitely in case they are required to investigate any security incidents.

3.3 Information Exchange and Data Sharing Technologies

This section will introduce some of the potential mechanisms that actors may use to distribute data among themselves. Figure 6 attempts to summarise those mechanisms that are of particular interest to OSMOS, as they have either been used within the Building and Construction Industry or propose a method that may satisfy some of the requirements of the industry.

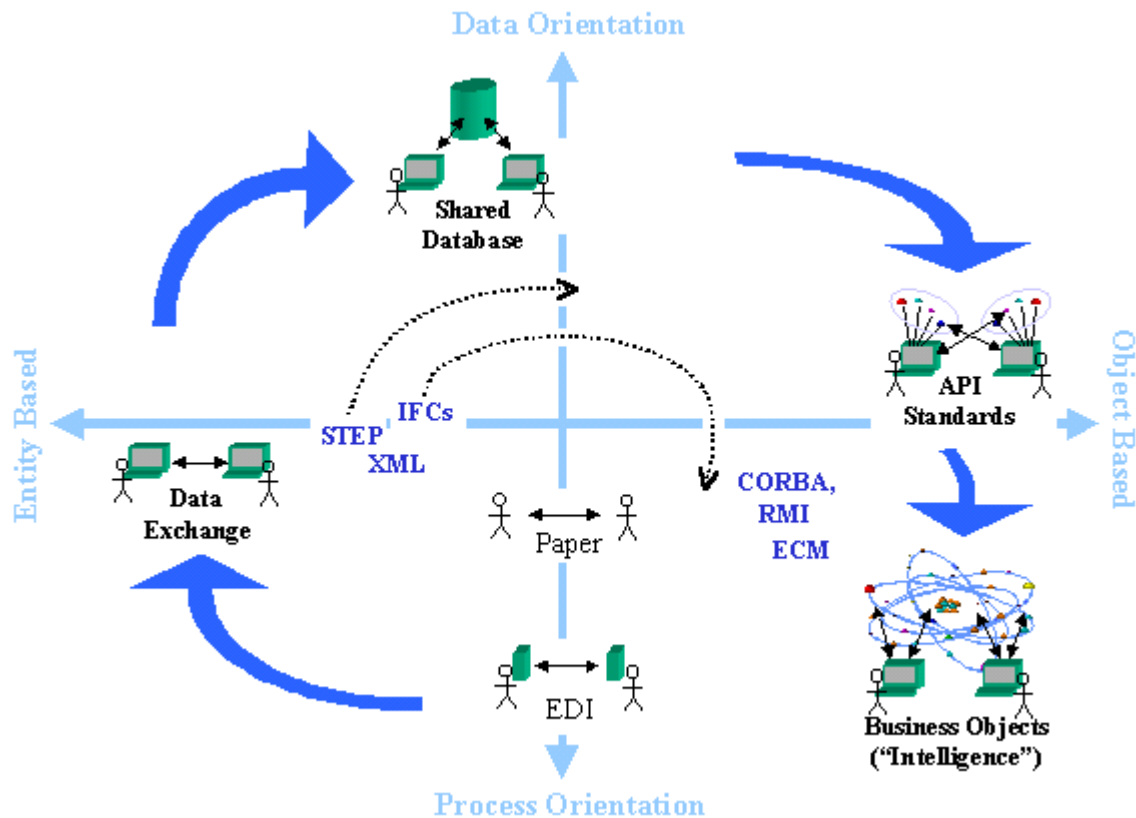


Figure 6: Methods for Sharing/Exchanging Construction-Related Data (Cooper & Rezgui, 2000)

These developments have been set against a pair of axes indicating the emphasis in relation to technological approach on the horizontal axis, and the emphasis in relation to purpose on the vertical axis. The horizontal axis makes the distinction between (left) so called “conventional” systems engineering approaches, which attempt to separate data (entities) and process, and (right) object-based approaches, which deliberately integrate data and process, to the right. The vertical axis distinguishes between (bottom) a process-oriented emphasis, geared towards facilitating interactions between actors within a business processes context and (top) a data-oriented approach, in which the emphasis is on harmonizing and sharing data structures across applications.

Electronic Data Interchange (EDI) involves the definition and exchange between organizations of a relatively small number of specific message types, which have very tightly defined semantics usually within a well-understood process model, e.g. within EDIFACT (ISO 1988), a Direct Debit or a Financial Payment Order.

Data exchange refers to the creation of standard data structures and formats for the large scale exchange of data between applications, most notably, STEP (ISO 1994) (ISO 10303, the International Standard for the Exchange of Product Model Data). This approach lends itself best to the exchange of data between similar or closely related applications because the receiving application needs to have the ability to hold and process an internal representation of the specific type of data received, with appropriate semantics, if it is to make use of it.

The shared database approach takes this a step further by separating out the data from the applications and attempting to arrive at a common data definition that may be used by all applications. Such approaches are often based on the idea of a single project database, such as STEP proposed access to a shared database using its Standard Data Access Interface (SDAI). This approach was particularly popular in the early to mid 1990s with projects such as ATLAS (Bohms et al., 1994), COMBINE (Dubois et al., 1995), RATAS (Bjoerk, 1994) and ICON (Aouad et al., 1995). A problem with this approach is the difficulty of convincing software producers to adopt a common data model, when part of their competitive advantage lies in adopting a data model that is optimised for their particular application area and their particular software implementation.

Continuing the evolutionary cycle, many medium-sized applications, such as word-processors, database packages and CAD tools, provide an API (Application Programming Interface) to allow third party software developers to extend the functionality of the software, or to interface other applications to them. The trend in recent years has been to present such an API in the form of a set of class definitions describing operations that may be used to manipulate conceptual objects. In the API of a word processor, concepts such as paragraphs, tables, paragraph styles and fonts are represented. These are then modelled in terms of operations that may be used to manipulate them such as *indent* and *set line spacing* for paragraphs, and *insert column* for tables, etc. The use of an object-based API in this way allows the application developer to take advantage of the interface definition capabilities of object-oriented software technologies to present a large, complex API in a clear and understandable way.

This approach is primarily used to allow legacy applications or large integrated corporate systems to interface to object or component based software by means of software wrappers or adapters (Gamma et al. 1994). Large-scale software systems such as ERP (Enterprise Resource Planning) systems are beginning to use this approach to support Internet-based extension and interfacing to third party systems (SAP 2000).

A fully component-based approach provides particular advantages where organisations wish to adopt a best-of-breed approach to software procurement rather than the all-in-one approach of many corporate systems. It also improves the possibility of distribution of applications across different organisations with a clear allocation of responsibilities to each software component (and therefore organisation).

The following sections will detail two major standardisation efforts in the area of Information Exchange and Sharing, namely STEP/IAI IFC and XML. These efforts are of particular interest to the OSMOS Project for the reasons that will be discussed.

3.3.1 State of the Art for Semantic Product Data Modelling

Product modelling is an important step towards the integration of information across the various disciplines of the industry, including Construction. Eastman and Augenbroe (1998) have produced a comprehensive description of the ongoing effort in the area of product modelling.

However, after over ten years of product model development through STEP (ISO 1996) and more recently the IFC (IAI 1994), the industry has not yet reached its promised ‘Nirvana’.

Several critical limitations of STEP have been identified. These limitations should not necessarily be addressed by STEP, they are however essential for the effective sharing of integrated construction project information and knowledge. The following issues are under discussion within the STEP community and need to be properly addressed:

- Support for the behavioural aspects of products,
- the support for flexible form models (Architects need flexible methods to define the shape of their project),
- the support for different views by different actors,
- the life cycle positioning and functional objectives of Application Protocols,
- the support for object states, versioning and ownership,
- the support for project history and capturing the intent behind decisions, etc.

The question of whether STEP’s remit will be widened beyond “product data” in the future remains open. A section of the STEP community feels that, due to the dynamic nature of the building design and construction process and the continued change in available materials and products, schema evolution is a prerequisite for the complete support of product description throughout its life cycle.

In fact, standards like STEP or the IFC normalise product data entities at an appropriate level for data exchange and sharing, but not for interoperable product components

However, despite the above mentioned limitations (acknowledged by the academic community), STEP and in particular the IFCs constitute one of the most promising milestone towards the standardisation of product information in the Building and Construction sector, leading to major improvements in information integration across disciplines and stages of the design and construction process. A natural extension to the IFCs would be to encapsulate within them the behavioural aspects of products. The authors of the present deliverable believe that product components (referred to, more generally, as Business Objects) can provide an opportunity to better structure knowledge in the Construction domain. This approach has already been adopted by several software vendors, including Bentley Systems through their Engineering Component Model (ECM).

3.3.2 State of the Art for Information Interchange

This section discusses the advent of the eXtensible Markup Language (XML)²⁴ as a mechanism for exchanging data. Its roots lie in the Standardised General Markup Language

²⁴ Further information about XML can be found at <http://www.w3.org/XML/>.

(SGML), a language for describing and inserting in a neutral way tags within any type of documents. XML has been developed by the W3C originally for the exchange of structured documents within Intranets or over the Internet, in a simpler way than when using SGML. Thus, XML plays a role similar to SGML, but in open and standardised networks. XML is text-based, self-describing, and, above all, gaining acceptance as a global standard. The XML language is indeed a meta-language, allowing the creation of any (XML-based) new language, and especially what can be called “data presentation language”: this leads as well to the definition of new file formats that can be instantly parsed by any XML-compliant application. Thus, for a particular domain, it is quite possible to create a new presentation (or exchange) semantics, *which is however different from the semantics of the data themselves (i.e. the content of the XML message).*

Considering the various aspects of enterprise application integration (EAI) and the 3-tier architectures as previously exhibited, the XML model can be considered from two different view-points, depending of the fact one targets the client side or the server side.

3.3.2.1 XML for cataloguing and further distributing/recovering information

The main key-points involved are concerned with using XML-compliant browsers for searching XML documents, designing DTDs for various Web documents, and controlling those documents along with DTDs and style sheets for representation (making use of XSL: *eXtensible Style-sheet Language*):

XML can be used for managing data as messages content: thanks to the various tags surrounding each data, new advanced and more powerful search engines can be envisaged through cross-search inside a structured and vast content, while today HTML only allows a “full-text” search.

XSL allows the decoration of well-formed trees based on the underlying model of XML, in order to generate, for instance, one or several HTML pages from an XML message. It is worth noticing that this translation/formatting mechanism is not limited to HTML, and it is likely to interpret the structure of the XML message so as to produce any kind of protocol that could be further analysed by a VRML plug-in, or for CD-ROMs or a WebTV, etc. XSL can also be used as a device to convert tags from a DTD into tags of another DTD, and several XSL-based transformation engines already exist that carry these types of conversion on the fly when fuelled with conversion specifications.

Some of the benefits that can be expected include:

- A clear separation between content and presentation, with a mapping to various visual representations (including HTML views), and it is quite conceivable to mix data originating from distinct sources in a single Web page on the client side: XML is a technology for flexible, dynamic document content information.
- The XML DOM (*Document Object Model*) standard object API (specified by the W3C) aims at giving developers programmatic control of XML document content, structure, and formats. The DOM is the XML underlying object model, enabling to manipulate XML

documents through an object-oriented approach (with C++, Java, etc.). XML allows to display the data, as well as to find and extract information (by programming) within an XML data set on the client (and this is even true on the server side).

- On the client side, the data can be “naturally” re-introduced in a client local database. Moreover, it should be possible to ship the semantic meta-data as contained in the DB sources when data are streamlined, then allowing a better exploitation of these data.

3.3.2.2 XML for enterprise-level services

XML can be viewed as a standard way of passing data between many heterogeneous distributed application servers, as well as across multiple operating systems, therefore as a *basic model for data exchange at level of middleware layer*. It can be considered as a protocol offering in some cases an alternative to COM and CORBA protocols, especially in the Internet context (one can keep on using classical HTTP firewalls, without having to open those firewalls to DCOM or IIOP through proxies), thus facilitating distributed computing on the Internet. Even more, it could be considered as a link for communication between a COM domain and a CORBA domain, therefore using XML as a way of bridging protocols.

XML can be used to define the container for a message content, for any type of data provided by a repository. For example, considering two MOMs that are used within a project, it is possible to establish an XML grammar to encapsulate any message, thereby allowing to distribute the same message through both middleware environments, provided that there is a process written for each environment that is able to extract the route and content of the message. Thus, XML supports *data portability* as a platform-neutral document description meta-language that offers means for data serialisation. It is worth noticing, at that point, that it can be quite beneficial to associate XML with Java, which offers *code portability*, as supporting the development of platform-neutral applications. Some of the expected benefits are:

- XML appears to promptly become a standard, with a potential role as a universally accepted format for the exchange of information between heterogeneous applications. XML is expected as one of the primary means for developers to design multi-tier applications in heterogeneous environments.
- XML seems to be appropriate in co-operative applications because dealing with documents and means to convey business knowledge. An interesting perspective is to define and use meta-data (this currently is few or not existing in most applications) that can be exposed through XML messages.
- XML supplies cost-effectiveness for implementing Internet and distributed applications based on XML software tools and components (both on client and server sides). Regarding the software market, a lot of actors (IBM, Oracle, Sun, etc..) integrate the parsing and generation of XML documents within their platforms. A lot of application servers use the XML format for sending information, and databases integrate as well an XML parser. In addition, a bunch of freeware tools are accessible through the Web. Thus,

as soon as an application is powered with XML, this should lead to minimal effort to exchange information.

- XML provides a way of tagging data and objects as they are called for on a network. Extending this feature already identified on the client side, allows an automatic way of populating databases on the fly with XML (especially local databases for specific applications connected to Intra/Extranet). Besides, in order to deal with the semantics of data that compose the content of XML messages, current efforts are undertaken, among others in the W3C with DCD and XML schemas, to define more semantics attached to an XML document content, including element names and rich data types.

3.3.2.3 The Simple Object Access Protocol (SOAP)

Preliminary note: the current description of the SOAP technology is the result of an initial study of SOAP simultaneously achieved within the OSMOS and Divercity²⁵ IST projects.

3.3.2.3a Introduction

SOAP is a protocol for enabling heterogeneous applications to communicate in a distributed environment. More precisely, those applications can run over the Internet (i.e. using existing Internet infrastructure), and they can communicate directly with each other over the Internet in a richer way than through simple HTTP-based mechanisms. This protocol is promoted by Microsoft, IBM, Sun and some other leading companies. Though still evolving and requiring further validation, SOAP seems a promising technology, and an alternative to CORBA technology through lighter protocol and implementation mechanisms.

3.3.2.3b SOAP Key features

SOAP is an XML-based protocol for accessing services, objects and servers in a platform-independent manner. There is a total freedom with SOAP regarding the underlying transfer protocol: HTTP, HTTPS, FTP, MOM, IIOP, etc.. Thus, SOAP can potentially be used in combination with a variety of other protocols, provided the specification of a binding between SOAP and a defined protocol is given: for instance, the current SOAP specification (Version 1.1) describes how to use SOAP in combination with HTTP, and with the HTTP Extension Framework. The transfer/transport protocol is indeed used to encapsulate SOAP requests over an IP-based network.

As mentioned in the SOAP specification, SOAP is a mechanism for exchanging structured and typed information between peers in a decentralised, distributed environment using XML:

- “structured” is obtained thanks to XML,
- “typed” is realised with various data encoding styles.

²⁵ IST-1999-13365.

Thus, it is a general communication-oriented protocol defining a typed serialisation format based on XML. Especially, this means that SOAP is independent of:

- any application-oriented semantics or meta-model;
- any underlying basic data transfer protocol (like HTTP or IIOP);
- any specific communication mode between SOAP-compliant applications.

Regarding the last point mentioned herein above, a fundamental point is that SOAP can be encapsulated in any transfer protocol. For instance, HTTP provides with an encapsulation that allows SOAP streams between a client desktop (that can be on any platform type, e.g. UNIX, PC, etc.) and a Web/HTTP server, and at the same time SOAP happens to use HTTP as a request/reply messaging transport (i.e. HTTP provides with a request/reply model for SOAP streaming). Anyway, SOAP can be used as well in one-way requests²⁶, multicast mode, asynchronous messaging, etc..

In a similar way, SOAP is designed to work well with the emerging XML Schema specification, and supports potential interoperation between COM, CORBA, Perl, Tcl, the C or Java languages, ASP or PHP programs running anywhere on the Internet. But this means that SOAP is not strongly tied to any of these technologies. For instance, there is no concept of Objects-by-reference, or remote activation (that should have required references on objects), as it is the case in CORBA, and SOAP is only a lightweight and extensible protocol to exchange data-oriented content between 2 peers: the SOAP encoding is used to marshal basic data types (integers, strings, etc.) from one communication end point to another.

3.3.2.3c General structure of the SOAP protocol

3.3.2.3.c.1 Main parts of a SOAP message

As the main objective of SOAP is to allow communication between applications and services on the Web, what is fundamentally required are mechanisms for describing services that are potentially to be offered (by server applications), for asking / discovering services (by client applications), and a basic data transfer format in order to exchange information between clients and servers. The SOAP protocol (and SOAP messages) basically consists of three parts:

- an envelope that defines a framework for describing what is in a message and how to process it;
- a set of encoding rules for transfer of data based on well defined (structured) data types;
- and a SOAP RPC representation, i.e. a convention for representing remote procedure calls and responses, in order to allow method calls (and response messages) with the required information (method name, signature, parameters, etc.).

²⁶ this is indeed the SOAP fundamental transmission type between a sender and a receiver.

A SOAP message is an XML stream that contains, as mandatory parts, an envelope and a body. The envelope mainly contains namespace declarations as well as attributes, in order to indicate to the receiver the serialisation rules used in a given SOAP message, and to provide him with the right URIs identifying those serialisation rules to be further used to deserialise the SOAP message (that can be a list of URIs indicated in the order of most specific to least specific). The envelope can be accompanied by a SOAP header, where extensions can be implemented in order to deal with extra information like authentication, transaction management, and so on. It can also be used to indicate which parts of the SOAP message is to be processed by the current (intermediate) application, and which parts must be discarded and forwarded to the next destination application (that can be or not the ultimate one).

The information itself (described in the body of a SOAP message, and intended for the ultimate recipient of the message) is expressed and transferred through an XML-based data description format relying on a set of encoding rules²⁷ for expressing instances of application-defined data types. These rules indicate the way to represent different data types (integers, floats, strings, arrays, compound types, etc.) and how to represent commands, i.e. operations on the data. This will be used by the recipient application to extract and interpret the right information, including marshalling RPC calls and error reporting. It is worth mentioning here that SOAP does not require that a specific object be tied to a given endpoint: rather, it is up to the developer to decide how to map the object endpoint identifier onto a server-side object. What is interesting as well in the use of XML for SOAP is that, following a common agreement, it is possible to extend the language when necessary: for instance, the extensibility and flexibility of XML precisely allow to encapsulate and exchange RPC calls through SOAP messages.

In conjunction with SOAP, the Web Service Description Language (WSDL) can be used as a way to describe the capabilities of a Web Service. WSDL is an XML format for describing network services as collections of communication endpoints (applications) operating on messages containing either document-oriented or procedure-oriented information. Thus, each application connected to the network can provide its services, by describing the various operations it can realise and with which data parameters, under an abstract form (i.e. independently of a concrete network protocol and message format to define an endpoint). Then, a WSDL specification must be bound to given message format or network protocol, e.g. towards SOAP 1.1 for SOAP-compliant applications.

3.3.2.3.c.2 Processing of SOAP messages

Processing a message requires that the SOAP processor understands, among other things, the exchange pattern being used (one way, request/response, multicast, etc.), the role of the recipient in that pattern, the employment (if any) of RPC mechanisms, the representation or encoding of data, as well as other semantics necessary for correct processing. Then, considering HTTP as the underlying network protocol, a SOAP endpoint is simply an HTTP-

²⁷ As stipulated in the SOAP specification, the SOAP encoding style is *based on a simple type system that is a generalisation of the common features found in type systems in programming languages, databases and semi-structured data.*

based URL that identifies a target for method invocation, and a SOAP method is simply an HTTP request or response that complies with the SOAP encoding rules.

In order to illustrate a SOAP messages exchange, we give herein under an example of a client requesting the position of an object using the following message:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
<SOAP-ENV:Body>
<m:getObjectPosition xmlns:m="http://www.c-s.com/hrweb">
<obj_ref>5</obj_ref></m: getObjectPosition >
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The server will respond to this request by the following message:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
<SOAP-ENV:Body>
<m:getObjectPositionResponse xmlns:m="http://www.c-s.com/hrweb">
<object obj_ref="5">
<x>0.0</x>
<y>50.2</y>
<z>-2.4</z>
</object>
</m: getObjectPositionResponse >
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

XML, along with the DOM (Document Object Model), allows SOAP to be very flexible, getting only requested information inside a large database structure. Typically, it is for instance possible and easy to get and send information on a specific room from a large building structure, without having to transfer the whole building information.

3.3.2.3.c.3 A SOAP example in the context of the OSMOS project

Like it has been mentioned above, the aim of SOAP is to enable communication between decentralised peers, in an heterogeneous environment. Since an OSMOS environment deals with distributed applications, often developed using various programming techniques and languages, the need for such a protocol becomes obvious.

Using “high-level” distributed computing technologies (i.e. CORBA) means a work overhead for every OSMOS partner, and may be “too much” if employed only as a simple communication mechanism. Thereby, the OSMOS consortium has to adopt a robust, powerful and moreover simple technique to handle communication between the “OSMOS Compliant” applications. Looking at the SOAP approach, it seems that we can inspire of it to set up an appropriate solution for the OSMOS framework.

This has been realised in WP3, and is exemplified in the corresponding deliverables. Nevertheless, to illustrate it, we introduce below an example that presents a “SOAP-like” messages exchange between an OSMOS Client application and an OSMOS server, in the case of a user authentication process: the client application requests the authentication of a user by its name and password.

The request would be like :

```
<Xcall>
  <service>
    <name>UserManager</name>
  </service>
  <method>
    <name>authenticate</name>
    <params>
      <param>
        <name>login</name>
        <value>john</value>
      </param>
      <param>
        <name>password</name>
        <value>azerty</value>
      </param>
    </params>
  </method>
</Xcall>
```

And the reply would be like (if the user has been successfully authenticated) :

```
<XReply>
  <error>O</error>
  <params>
    <param>
      <name>result</name>
      <value>USER_OK</value>
    </param>
    <param>
      <name>objectId</name>
      <value>27</value>
    </param>
  </params>
</XReply>
```

Of course the handling of types is submitted to deep investigations (arrays, structs, compound types, objects...), but we obviously see on this example that a SOAP-like approach could be useful, and would help us to avoid the implementation of more complex technologies.

3.3.3 Comparison of the Potential IT Solutions That Support the Technical Requirements of the Virtual Enterprise

The following table gives a brief overview XML and STEP/IFC technologies, concentrating on how well they answer the requirements as set out in Chapter 2. However, the reader must appreciate that the two approaches have different areas of scope, i.e. STEP and IFCs concentrate on exchanging and sharing data related to product modelling, whereas XML is more generic than that (depending on the DTD used.) The two are presented not only to allow the reader to compare them side-by-side, but to also introduce XML as a candidate for data exchange between objects in component-based middleware (see Section 3.1.3.4).

✓ Yes ✗ No ? Unknown ²⁸ ◆ Varies ²⁹ - N/A	ISO STEP & IAI IFCs	XML
Transactional		
Platform independence / interoperability	✓	✓
Language independence/ interoperability	✓ (via SDAI)	✓
Management of huge set of data	✓ (Although Inflexible)	✓
Distribution support	-	-
Communication flexibility	Low Support	✓
Schema versioning (Interface evolution)	Low Support	✓
Support For Multiple Communication protocols	✓	✓
Communication types	-	-
Code Portability	✓ (using EXPRESS and EXPRESS-G)	✓
Ability to Define Security levels	-	-
Standardisation	✓	✓ (via Industry DTDs now, and XML Schemas in the future)
Openness of Standards & Standardisation Process	✓ (STEP: ISO; IFC: IAI)	✓ (XML: World Wide Web consortium; DTDs: ◆; XML Schemas: ◆)
End user enterprise applications integration	✓ (although not widespread)	✓ (starting to become widespread)
Quality of service	✓	✗
Any data types or specific types	✓	✗
Object oriented	✗	✗

²⁸ Unknown at this present time, although research into this is continuing.

²⁹ The taxonomy presents a generic comparison between the *types* of data exchange mechanisms. It does not compare specific vendor implementations of the middleware. For cases where some vendors provide this service and others don't, a diamond symbol denotes that this varies between implementations.

✓ Yes ✗ No ? Unknown ²⁸ ◆ Varies ²⁹ - N/A	ISO STEP & IAI IFCs	XML
Description and specification	-	-
Code mobility	-	✓
Scope	Product Data Modelling	Generic, but defined by DTD.

Table 2 A Comparison between XML and STEP/IFC.

3.4 Co-operation Technologies (Groupware)

A great deal of research has been done in the field of computer support for co-operative work (CSCW). CSCW is more generally concerned with the introduction and use of groupware systems to enable and support teamwork. Groupware deals with highly unstructured data, including text, image, graphics, faxes, mail and bulletin board. Groupware solutions include traditionally a subset of the following system components: Workflow (task scheduling), Multimedia Document Management, E-mail, Conferencing, and shared schedule of appointments. A recent survey of groupware constituent technologies reveals a lack of homogeneity, and a diversity of applicable de facto standards and APIs from the leading Groupware vendors.

Groupware has the potential to flatten organisations and remove layers of bureaucracy. Groupware helps manage and track the project lifecycle throughout its various stages. It also allows the actors collaborating on specific tasks to exchange ideas and synchronise their work. It offers the potential to keep track of the project memory and record all its “learned lessons” in a way that promotes reuse. Two of the main constituents of groupware are Workflow and Document Management Systems.

3.4.1 Workflow

According to the WfMC (Workflow Management Coalition), a Workflow process consists of a collection of activities. An activity is a logical step that contributes toward the completion of a workflow process. It is executed by application software outside the workflow system. Workflow helps bring the information to the people who can act on it. It co-ordinates existing software and tracks the processes and helps to ensure that those activities are performed by the right application undertaken by the right actors that have the right skills.

However, it is well acknowledged in the Construction industry that many enterprises, taking part to collaborative work involving several non co-located actors, are often reluctant to join and adopt a workflow-based process, or to fully use a Groupware solution. There is still a strong requirement to ensure the availability of up-to-date, accurate, and relevant information, while at the same time providing access control (based on actors’ rights and role in the project) and concurrent business transaction support. In fact, the peculiarities of the

Construction sector suggest that equal consideration should be given to technical as well as social, contractual, and legal aspects (including responsibilities) relating to projects.

Workflow is discussed in considerable detail in Orfali *et al* (1999, p393-407). They acknowledge that the market is very fragmented at the moment, and that workflow solutions are gradually being incorporated into Groupware solutions provided by the likes of Lotus (Notes), Microsoft (Exchange) and Novell. Interestingly, some Enterprise Resource Planning solutions (such as PeopleSoft and SAP R/3) are incorporating workflow services into their overall architectures.

3.4.2 Document Management Systems

Most information used during the Design and Build process of a Construction project is conveyed using documents. These are most of the time exchanged, for contractual and legal reasons, on a paper-based medium, even when produced using computers. The challenge that the industry is facing today is the re-use of the knowledge and lessons learned stored within these documents. The latter is unstructured, poorly organised, and embedded within the “black-box” that constitutes the document.

Document management has become a crucial issue within modern construction companies. The various solutions proposed by some software vendors have been revealed to be unsatisfactory, to a point where many leading construction organisations, with an advanced IT department, have undertaken the development of their own tools and solutions to support the production and maintenance of project documents. Even though such proprietary tools provide many helpful facilities, including support for document storage, retrieval, versioning and approval, they don't handle any semantics of the information being processed and therefore remain limited in their support of the end-user. In fact, construction project data and documentation (including full specification documents) constitute two fragmented information sectors where compatibility and interoperability are mostly needed. Moving these pseudo-sectors closer together to support construction project documentation as part of the life-cycle of the building product is becoming an actual and urgent topic for standard bodies and industry alike.

The background to electronic document management can be roughly summarised according to the two following approaches:

- The integrated document management approach in which documents are treated as black-boxes, and the aim of computer-support is to enable easy document storage, retrieval (using reference information), versioning and approval. This approach is becoming best practice due to the proliferation of EDM systems.
- The model-based approach in which the information traditionally contained in drawings and text documents is described and represented through an object model, and is contained in integrated databases. These databases are then used as a basis for the production and authoring of project documents.

Many commercial web-based EDM systems that lend themselves to the black-box approach described above are available today. These include ProjectNet³⁰, BidCom³¹, Evolv³², and Buzzsaw³³. These systems provide document and workflow management services across the Internet. Some, such as BuildOnline (<http://www.buildonline.com>) have tailored their services specific to the Construction industry.

3.4.3 Decision Support Systems

Since the introduction of knowledge-based systems, numerous implementations have been undertaken and deployed in industry, including Construction, with a varying level of success. The UK department of Trade and Industry reported in one of its surveys over 2000 knowledge-based systems deployed in business operations in industry, including manufacturing. These systems worked fairly well on problem domains that had an explicit model-based representation implemented through rules or objects.

However, as reported in (Watson and Marir 1995), developing KBS without an explicit problem domain model remains problematic. That is where other forms of reasoning, including Case-Based Reasoning, have been explored. Case-based reasoning organises the structured archival of past experiences for future potential re-use. These experiences, commonly referred to as cases, are archived along with their unique domain characteristics expressed through well defined indexes that describe the essence of the case (Watson et al. 1994). The success of this type of reasoning is largely explained by its simplicity, along with its similarity with human problem solving mechanisms. Case-based reasoning, compared to knowledge-based systems, provide many advantages (Watson and Marir 1994), including the following ones:

- it does not require an explicit domain model and so elicitation becomes a task of gathering case histories ;
- implementation is reduced to identifying significant features that describe a case ;
- CBR systems can learn by acquiring new knowledge as the number of cases increase.

CBR is still in its infancy in the Construction domain. Several studies and prototype implementation have been proposed which highlight the benefits of decision support systems using CBR techniques [Rezgui and Farhi 1997][Farhi and Watson 1995].

In fact, most available knowledge management systems rely on users' input to orchestrate the information and knowledge discovery and elicitation. This is, however, becoming

³⁰ ProjectNet: <http://www.bluelineonline.com/program/prducts/projnet.html>.

³¹ BidCom: <http://www.bidcom.com/html/overview.html>.

³² Evolv: <http://www.bricsnet.com/>

³³ Buzzsaw: <http://www.buzzsaw.com/content/services/main.html>.

increasingly complex as the electronic sources of knowledge are vast and rapidly growing with the successful deployment of IT systems within organisations. In addition, these systems have limited collaborative functionality and do not encourage information and knowledge discovery (Berney and Fernley 1999) - they require the user to have a clear idea of the appropriate search terms. Such systems also require the tacit knowledge *giver* to be able to clearly articulate their knowledge and experiences. Prior to the introduction of technology to facilitate collaboration, an appropriate organisational culture must be in place to make the use of the technology effective (Skyrme, 1999). This is an area where the agent technology can provide potential solutions. The usefulness of the application of the agent technology has been highlighted in (Bradshaw et al. 1997).

However, most today available agent-based systems operate on proprietary frameworks. They make use of proprietary underlying models and legacy script languages, and target corporate proprietary and legacy databases. This result in limited application interoperability, poor integration in large enterprise information systems, and limited ability to extend (e.g. by direct integration of other pre-built components not based on the same agent network) and to follow the market evolution (Zarli et al. 1998).

3.4.4 Electronic Mail

Electronic mail (e-mail) gives the opportunity for an actor to send a message to another actor quickly and easily, much the same way as a letter is 'mailed' through the postal service. As email has now become unambiguous within today's society, it would be overkill to describe it in detail here.

However, there are a couple of issues that do need to be addressed. There are two main electronic mail infrastructures. The Internet-based approach uses a myriad of protocols such as SMTP, POP3, and IMAP4. This infrastructure works, is quick, simple, widely used on the Internet by end-users and allows for the sending of files (using S/MIME) as well as text messages. However this approach is relatively insecure (messages are passed around the Internet in 'plain text' and can easily be faked) although this can be overcome by the use of encryption and digital signatures (see Section 3.2). In addition, there is no guarantee that a sent message will actually be received at its destination. The other infrastructures that overcome these deficiencies are those built upon the X.400 standard, although overcoming these deficiencies has the trade-off that the infrastructure is expensive and complex to set-up and maintain. X.400 is widely used in some organisations, such as the British National Health Service. Thirdly, Novell has an infrastructure called MHS, the Message Handling System.

Orfali *et al.* (1999, p415) note that electronic mail capability is an area where a number of Application Programming Interfaces (APIs) have become available. These allow developers to easily add e-mail functionality to their applications – as long as the developer knows what infrastructure is being used (this isn't really a problem as the vast majority use Internet-based email services). Orfali *et al* describe the four main APIs as:

- **VIM** (Lotus' Vendor Independent Messaging) providing a cross-platform interface;

- **MAPI** (Microsoft's Messaging API), which unsurprisingly is very heavily reliant on COM (and therefore Windows-based platforms);
- **CMC** (the X.400 API Association's Common Mail Calls) which only provides very simple e-mail functionality;
- **JavaMail** (from JavaSoft) which is platform and protocol-independent, but is implemented as an extension to Java.

3.4.5 LDAP - Lightweight Directory Access Protocol

3.4.5.1 What is LDAP?

The Lightweight Directory Access Protocol (LDAP) is a protocol for clients to query and manage "arbitrary" information in a (hierarchical) Directory Service over a TCP connection (port 389).

The LDAP protocol was designed by University of Michigan to provide access to the X.500 Directory while not incurring the resource requirements of the Directory Access Protocol (DAP).

Many vendors are (or have announced plans) to expose their directories (and other attribute/value pair type information) through an LDAP mechanism.

RFC 1777 & 1778 [RFC1777,RFC1778] cover the basics of LDAP.

3.4.5.2 LDAP Basics

LDAP introduces a lot of new terminology, but the only terms needed to understand and to get started are *entry*, *object class*, *attribute*, and *distinguished name*.

An LDAP server contains entries, and each entry's type is defined by an object class. An object class defines attributes, both required and optional, associated with an entry of that class. Each entry is uniquely identified by a distinguished name, or DN. The DNs are organised in a hierarchy; each one consists of the name of an entry plus a path of names tracing the entry back to the root of the tree.

For example, given that I work at a company in the United States, the top entry in the hierarchy for my entry has the DN "c=FR". This top entry is of the object class `country`. A `country` entry has one required attribute, `c`, with the value of `FR` in this case. At the next level down in the hierarchy, an entry of the object class `organization` has the DN "o=CSTB, c=FR". The `organization` entry requires the attribute `o`, which is `CSTB` in this case. The parts of the DN are separated by a comma.

What we have at this point would likely be the base DN for this server. A base DN defines the top of the namespace that the server is responsible for, much like a DNS zone. As we add

entries further down the hierarchy, the DNs become longer. Remembering a long DN is not an issue for end users, because client applications will be doing the searches and then displaying the attributes the user needs to see. For most applications, the full DN does not need to be exposed to the end user.

Now that the groundwork has been laid, let's look at an entry for an individual. The entry's DN is:

```
"cn=Mathieu Marache, o=CSTB, c=FR"
```

This entry is of the object class `inetOrgPerson`, which requires a `cn` attribute (for "common name"). It also requires an `sn` attribute for the surname; this is an example of a required attribute that is not the attribute included in the DN. The `inetOrgPerson` class also defines about 50 other attributes that can be associated with a person, such as `uid`, `title`, `manager`, `telephoneNumber`, `pager`, `mail` (e-mail address), and other information you would likely want to associate with a person in an organization that has access to the Internet. It even has the attributes `jpegPhoto` and `audio`, which are possible because attribute values can be declared to be encoded into different *syntaxes*, including binary data.

3.4.5.3 *More than a protocol*

Understanding how to maintain the data is not enough to be able to put LDAP to use. There are four well-defined pieces of the overall system that simplify implementing LDAP: the LDAP open standard, the API, the LDIF text format for data, and the object class definitions.

- **LDAP** -- RFCs 1777 and 1778 [RFC1777,RFC1778] define the protocol that allows clients from different developers on any platform to talk to any type of LDAP server. Many vendors have announced support for LDAP. Notably, Netscape, Microsoft, and Novell already offer directory-enabled servers and clients. The University of Michigan, where LDAP evolved, has source code for their original `slapd` server and other tools available for download.
- **API** -- One of the important factors in the success of LDAP is that developers should be able to make use of information from an LDAP server without having to write and debug a lot of code. A well-defined application programming interface (API) helps make this possible. For a program to make use of directory information, you simply include the API libraries in the source directory, modify the program's code to call the API functions at the point where the information needs to be looked up, and recompile. The most recent version of `sendmail` already includes the API and has options to look up information through LDAP.
- **LDIF** -- Another important piece of the puzzle is the LDIF file format. This ASCII text format is used for exporting and importing data to and from LDAP servers. This not only makes it easy to migrate data from one server to another but also allows you to write scripts to create LDIF files from other data sources. You can then verify and manipulate the LDIF file before committing the data to the server.

- **Object classes** -- One other piece that's important to portability is object class definitions. If a client needs some attributes that aren't in the well-known object class definitions, a new object class can be created as an extension of a similar object class. The client could then work with any LDAP server, as long as the server has been given this new object class definition.

3.5 Wireless Applications and Protocols

3.5.1 Introduction: the Wireless Application Protocol (WAP)

Wireless devices such as mobile phones, personal digital assistants, etc. are constrained in terms of limited CPU, memory, battery life, and a small and simple user interface. At the same, the wireless networks through which they communicate are constrained by low bandwidth, high latency, and unpredictable availability and stability (WAP Forum, 2000). These problems are addressed through the Wireless Application Protocol specification, which attempts to enable industry participants to develop air interface independent, device independent and fully interoperable solutions for wireless devices. Delivered through WAP-enabled wireless devices, WAP services are tuned to provide timely information access and delivery where and when a full screen environment is not required or isn't available.

The Wireless Application Protocol (WAP), published by the WAP Forum (founded in 1997 by Ericsson, Motorola, Nokia and Unwired Planet) is an open application communication protocol used to access services and information through wireless devices such as mobile phones, personal digital assistants, etc. WAP is a protocol designed for "micro browsers" that enables the creation of web applications for mobile devices. It is based on existing Internet standards such as (HTML, XML, TCP/IP, etc.).

3.5.2 WAP Specification

The WAP specification defines an open standard architecture and set of protocols to implement wireless Internet access. WAP is based on a programming model (figure 1) that is similar to the existing WWW programming model. Some optimisations and extensions have been made to be in accordance with the wireless environment. Where possible, conformance to existing standards from W3C, ETSI, TIA, IETF, etc. has been made.

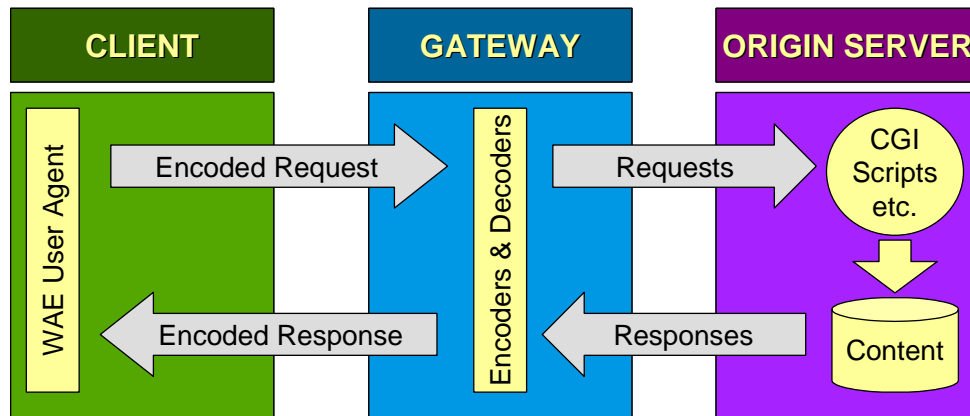


Figure 7: WAP Programming Model (WAP Forum, 2000)

WAP specifies:

1. Wireless Application Environment
 - WML Microbrowser
 - WMLScript Virtual Machine
 - WMLScript Standard Library
 - Wireless Telephony Application Interface
 - WAP Content Types
2. Wireless Protocols
 - Wireless Session Protocol (WSP)
 - Wireless Transport Layer Security (WTLS)
 - Wireless Transaction Protocol (WTP)
 - Wireless Datagram Protocol (WDP)
 - Wireless network interface definitions

Figure 8 shows the different WAP protocols and corresponding layers. A comparison between Internet and WAP technologies is also presented.

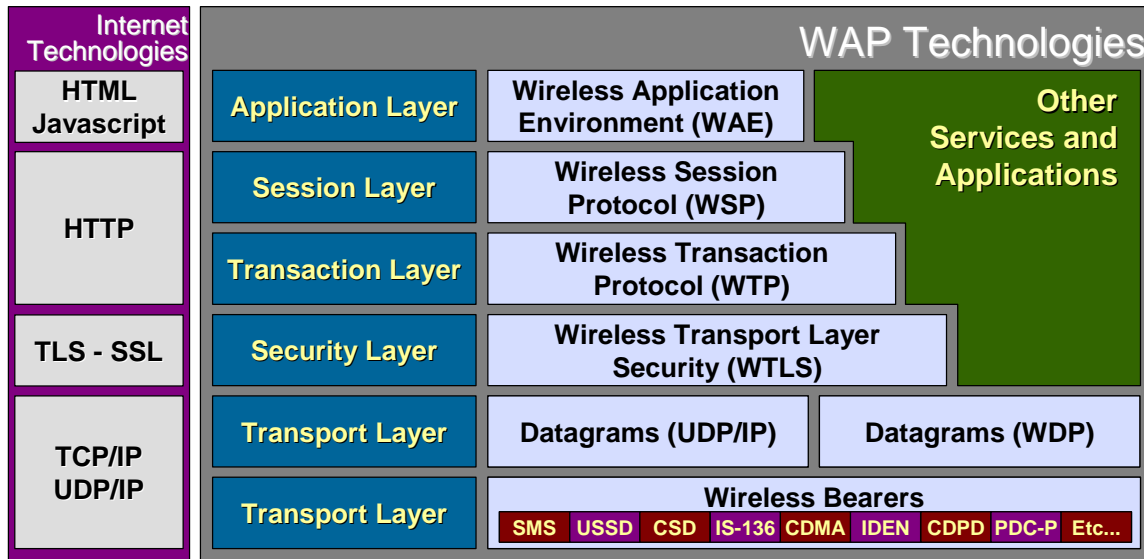


Figure 8: WAP Protocol Stack (with modifications, WAP Forum, 2000)

To ensure smooth operability within existing environments and the minimisation/elimination of the need for code rewrite, WAP makes use of XML and furthermore utilises plain HTTP 1.1 servers (this allows the leveraging of existing development methodologies, CGI, ASP, NSAPI, JAVA, etc.).

3.5.3 Wireless Application Environment (WAE)

The focus of the Wireless Application Environment (WAE) is to provide a network-neutral application environment for narrow-band wireless devices. WAE is based on the Internet/WWW programming model and offers a high degree of interoperability.

3.5.3.1 Wireless Markup Language (WML)

WML is a tag-based browsing language that is based on the W3C XML language. It inherits some technology from HDML and HTML as well. WML is used for screen management (text, images), data input (text, selection lists, etc.), and for hyperlinks and navigation support. Each WML file (referred to as a deck) contains several cards. Each deck is downloaded as a whole and as such navigation between the different cards of the deck can be done without additional requests to the content provider. Some limitations do exist, for example related to the maximum size of the deck (WML file). These are mentioned in later sections.

Each WML file consists of at least a document prologue (XML and document type declaration) and a <WML> element containing at least one card. Figure 9 presents that basic structure/layout of a typical WML deck.

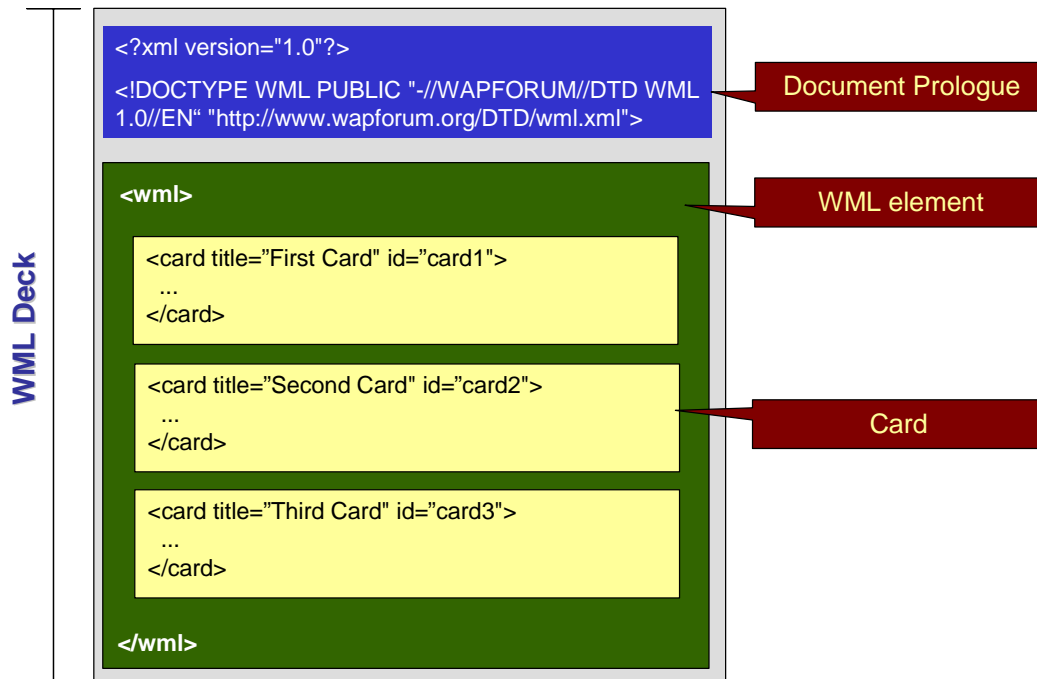


Figure 9: Layout of a WML Deck

3.5.3.2 WMLScript

WMLScript is the equivalent of Javascript (ECMA script) for WML. Derived from ECMA script, WMLScript is a scripting language for procedural logic, loops, conditionals, etc. optimised for small-memory and small-cpu devices.

Some typical uses of WMLScript include, field validation (check for formatting, input ranges, etc.), device extensions (access device or vendor-specific API), conditional logic (download intelligence into the device). One simple point to note about WMLScript is that it must be kept in a separate file, and cannot be used with inline tags like JavaScript.

3.5.3.3 Wireless Telephony Application (WTA)

WTA provides a set of tools for building telephony applications. The WTA browser for example provides extensions to the standard WML/WMLScript browser in addition to exposing additional API (WTAI). WTAI-Wireless Telephony Application Interface, includes:

- Call control
- Network text messaging
- Phone book interface
- Indicator control
- Event processing

3.5.4 Wireless Protocols

3.5.4.1 Wireless Session Protocol (WSP)

The WSP provides a shared state between the client and server used to optimise content transfer. Its semantics and mechanisms are based on HTTP 1.1. It supports and enables compact encoding, push messages, and efficient negotiation between the client and server.

3.5.4.2 Wireless Transport Layer Security (WTLS)

The WTLS provides connection security for two communicating applications. This is done through privacy (encryption), data integrity (MACs) and authentication (public-key and symmetric). WTLS is a lightweight and efficient protocol with respect to bandwidth, memory and processing power. Special mechanisms for wireless use are employed to enable long lived secure sessions, optimised handshake procedures, and provision for data reliability for operation over datagram bearers.

3.5.4.3 Wireless Transaction Protocol (WTP)

The WTP provides an efficient request/reply based transport mechanism optimised for devices with limited resources over transport networks with low to medium bandwidth. It supports the retransmission of lost packages (of data), selective retransmission, segmentation/re-assembly, port number addressing (UDP ports), and flow control.

3.5.4.4 Wireless Datagram Protocol (WDP)

The WDP provides a connection-less unreliable datagram service. The WDP is replaced by UDP when used over an IP network layer.

3.5.5 Serving Content to WAP Devices

3.5.5.1 WAP Gateway

The common way that content is served today is through a WAP gateway. The content provider adds the following MIME types to their server (Apache, IIS, Netscape Enterprise server, etc):

- text/vnd.wap.wml for .wml files (WML source files)
- application/vnd.wap.wmlc for .wmlc files (WML compiled files)
- text/vnd.wap.wmlscript for .wmls files (WMLScript source files)
- application/vnd.wap.wmlscriptc for .wmlsc files (WMLScript compiled files)
- image/vnd.wap.wbmp for .wbmp files (wireless bitmaps)

When a client makes a request for information, they make a request using WSP/WTP to a WAP Gateway, which first decodes the request and then encodes it to send as a HTTP request to the content provider. The response from the content provider follows the reverse route (Figure 10).

This approach is slow and comparatively “less secure” as it goes through an “external” WAP gateway. The telephony service provider usually provides the WAP gateway as a service to its subscribers.

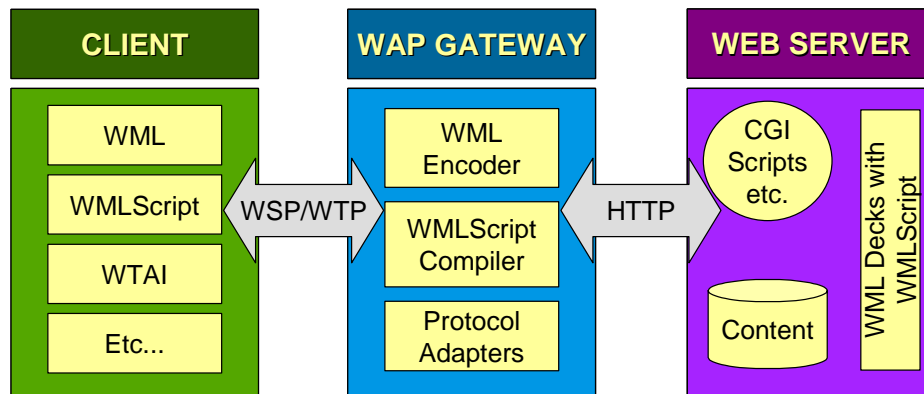


Figure 10: Serving WAP content through a WAP Gateway

3.5.5.2 WAP Application Server

The new way of serving WAP content is through a WAP Application Server. As may be noted from figure 5, the WAP Gateway is eliminated in principal and there is no more a need to use HTTP requests. This not only saves time, but also provides more security and control to the content provider hosting the WAP Application Server.

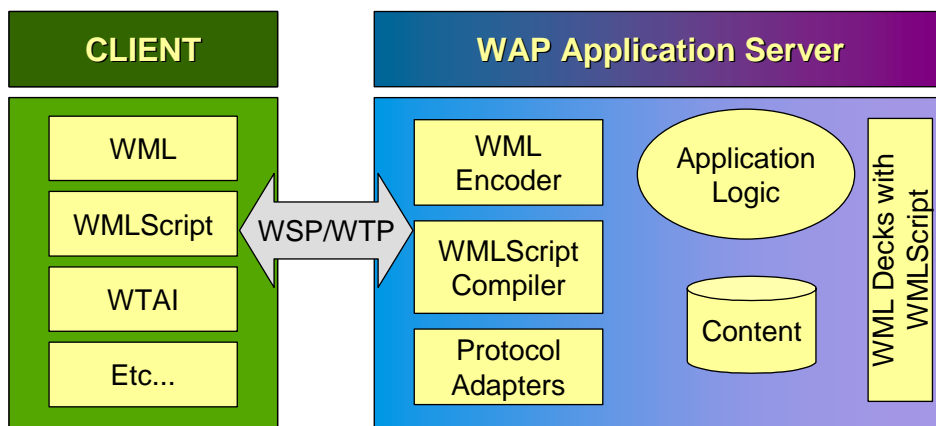


Figure 11: Serving WAP content through a WAP Application Server

3.5.6 Examples

Here we present three simple WML examples. The assumption is that we are serving the content through a WAP gateway (Figure 10).

3.5.6.1 Plain WML Example

This example is a simple and plain WML file that provides an entry point to the OSMOS WAP services. Upon entry a welcome message is displayed along with an image. A link to continue to another deck, "menu.wml" is also provided. Additionally the user is provided (through a template element), an option to return back to where they initially came from.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN" "http://www.wapforum.org/DTD/wml_1.1.xml">

<wml>
  <template>
    <do type="options" name="prev" label="Back">
      <prev/>
    </do>
  </template>
  <card title="OSMOS WAP" id="welcome">
    <p align="center">Welcome to OSMOS <br/>
      <br/>
      
      <br/>IST-1999-10491 <br/>
      <br/>
      <a href="menu.wml">Continue </a>
    </p>
  </card>
</wml>
```

3.5.6.2 Servlet Example

The following example uses a Java servlet to produce a WML deck. When a request is made, the deck is generated. As may be seen from the code, dynamic content is being served (in the form of today's date) using the java.util.Date() method. Alternatively, it would have been possible to do the same using WMLScript.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class WAPDate extends HttpServlet {

public void service (HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {
// set content type for wireless data
response.setContentType("text/vnd.wap.wml");
// get the communication channel with the requesting client
PrintWriter out = response.getWriter();
// write the data
out.println("<?xml version=\"1.0\"?>");
out.println("<!DOCTYPE wml PUBLIC \"-//WAPFORUM//DTD WML 1.1//EN\"");
out.println(" \"http://www.wapforum.org/DTD/wml_1.1.xml\">");
out.println("<wml>");
out.println("<card title=\"Hello OSMOS\">");
out.println(" <p align=\"center\">");
out.println(" Welcome to OSMOS <br><br>");
out.println("Date is: " + new java.util.Date());
out.println("</p>");
out.println("</card>");
out.println("</wml>"); }
}
```

3.5.6.3 ASP Example

The following example using ASP, demonstrates a simple login script. A user needs to provide their username and password, which are then validated from a database. If the validation fails, the user is returned to the login screen, and if the validation is successful, the user is sent to the next card in the deck.

```
<% Response.ContentType = "text/vnd.wap.wml" %>
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
  <card id="login" title="OSMOS Login!">
    <p>
<%
username=Request.Form("username")
password=Request.Form("password")
Set MyConn = Server.CreateObject("ADODB.Connection")
MdbFilePath = Server.MapPath("contacts.mdb")
MyConn.Open "Driver={Microsoft Access Driver (*.mdb)}; DBQ=" & MdbFilePath & ";"
checkSQL = "SELECT * FROM access WHERE username=" & username & " AND password=" & password & ""
set rs = MyConn.Execute(checkSQL)
  if rs.EOF then
    Response.Write "Invalid Login"
    Response.Write "<do type='accept' label='Retry'"
    Response.Write "  <go href='index.asp#login/'>"
    Response.Write "</do>"
  else
    Response.Write "Welcome to OSMOS Wap Services.<br/><br/>Successful Login!"
    Response.Write "<a href=#menu>Continue...</a>"
  end if
%>
    </p>
  </card>
  <card id="menu" title="OSMOS Menu">
    <p>
      <a href="about.asp">About</a>
      <a href="contacts.asp">Contacts</a>
      <a href="todo.asp">To-Do</a>
    </p>
  </card>
</wml>
```

3.5.7 WAP Services

Different WAP services have been developed and are available for access (some for free) through WAP-enabled devices. These services include:

- Location-based services
 - Real-time traffic reporting
 - Event/restaurant recommendations
 - Real-time news and weather reports
 - Instant messaging

- Enterprise solutions
 - Email access
 - Database access
 - Wireless intranets
 - Information updates “pushed” to WAP-enabled devices
- Financial services
 - Banking
 - Bill-paying
 - Stock trading
 - Funds transfer
- Travel services
 - Schedules and rescheduling
 - Reservations
- Gaming and Entertainment
 - Online, real-time, multi-player games
 - Downloadable horoscopes, cartoons, quotes, etc.
- M-Commerce
 - Instant comparison shopping
 - Location-based special offers and sales

3.5.8 Known Limitations

Known limitations other than support for colour and graphics are basically due to the limited capacities of client devices. While some WAP devices support deck sizes of up to 3000 bytes, others are limited to 1074 bytes, thereby making it difficult to serve content without consideration of the client device. In terms of images, only wbnp (wireless bitmap images) are currently supported. Their dimensions are again constrained based on the display size of the client device. Furthermore, since a wbnp forms a part of the deck, care must be exercised to ensure that the size of the deck is controlled.

Limitations are now being resolved as more bandwidth becomes available and the second generation of WAP-enabled devices become available.

4. Conclusion: Selection of the OSMOS Base Technology

The purpose of this document is to present to the reader a brief introduction of the candidate base technologies, with the aim of highlighting their relative strengths and weaknesses. The selection of the OSMOS base technology will be realised on the basis of a step-by-step approach undertaken in the development of the OSMOS framework. This approach will be described in Deliverable D2.2 of Workpackage 2.

At the time of writing, the three steps that are currently planned in OSMOS, and supposed to be expanded in each of the three phases of the OSMOS project, will encompass the base technologies as follows (from a purely technological point of view):

- The first iteration will concentrate on using basic Internet-based technologies, such as HTTP and HTML, and should allow the Consortium to start investigating and evaluating the set of XML technologies, by assessing their features and benefits against the requirements of the end-users.
- The second iteration will concentrate on providing extensions on the server side (because the remit of OSMOS was to ensure the client-side technologies stay at a low entry level), based on CGI technology, or preferably Java servlets technology. Investigations about the potential of MOM-based technology (especially Publish & Subscribe) will be carried out, once again evaluating this approach against users and construction projects requirements. In the context of a Java-based environment, the Java Messaging Service aims at being a specific target.
- Depending on the users' feedback and the evaluation of the previously implemented technology, the third iteration could be more technologically ambitious, utilising the advantages of fully object-oriented and component-oriented technologies. The implementation of this will be based on one of the following platforms, which are listed in order of preference at the time of writing:
 - **A Java-based approach:**
Java provides many advantages as a whole, especially with the ability of the OSMOS Solution being able to be hosted on any system connected to the Internet, regardless of platform. Such an implementation would utilise the many APIs that Java inherently provides, such as JMS, RMI/IIOP, servlets, with the possible integration of databases using JDBC, providing 'intelligence' on the server using Enterprise JavaBeans, and possibly using the Java Transaction API for a transaction-based middleware solution.
 - **Microsoft Technology (DCOM / MTS / COM+, and Active X);**
Although Microsoft's technologies do have a number of disadvantages, they are used on the vast majority of computers in the vast majority of organisations worldwide. This is compounded by the fact that Microsoft's range of development environments are ideally suited to developing solutions that will be implemented using Microsoft-based platforms. However, given the situation in the United States and European

Union regarding Microsoft being investigated for alleged abuses of its market position, the Consortium will keep an eye on developments regarding the future of Microsoft (and its products).

- **A CORBA-based solution;**

The CORBA-based solution would be an ideal one, combining the power of Java with the increased performance in security and transaction support. However, the CORBA 3.0 specification has not been fully released at the time of writing, and consequently products implementing this specification are currently unavailable. The Consortium will therefore keep an eye on developments in this area, especially with the increasing level of co-operation between the OMG and JavaSoft, enhancing the use of Java within a CORBA architecture. Additionally, it is possible as well that the specification itself is subject to evolutions in the future.

Acknowledgements

The OSMOS Consortium would like to acknowledge the financial support of the European Commission under the IST programme.

References

- Aouad, G., P.S. Brandon, T.M. Child, G.S. Cooper, S. Ford, Kirkham, J.A., Sarshar, M.(1995). ICON Final Report, University of Salford.
<http://www.salford.ac.uk/docs/depts/survey/staff/GAouad/pubs.html>
- Bernstein, P., Hadzilacos, V. & Goodman, N. (1997) *Concurrency Control and Recovery in Database Systems*. Addison Wesley.
- Bernstein, P. (1996). Middleware: A Model for Distributed Services. *Communications of the ACM*. **39** (2) (February 1996) 86-98.
- Björk, B-C., 1994. RATAS Project - Developing an Infrastructure for Computer-Integrated Construction, *Journal of Computing in Civil Engineering*, Vol. 8, No. 4, 400-419.
<http://www.vtt.fi/cic/ratas/index.html>
- Bohms, M., F. Tolman, & G. Storer, 1994. ATLAS, a STEP Towards Computer Integrated Large Scale Engineering, *Revue internationale de CFAO*, 9 (3): 325-337. <http://www-uk.research.ec.org/esp-syn/text/7280.html>
- Bray, M. (1997). *Middleware*. 25 June 1997. Cited 18 January 2000. Available on the World Wide Web at <http://www.sei.cmu.edu/activities/str/descriptions/middleware.html>.
- Birngruber , D., Kurschl, W. & Sametingler, J. (1999). *Comparison of JavaBeans and ActiveX –A Case Study*, STJA 99, Smalltalk und Java in Industrie und Ausbildung, Erfurt, Germany, September 28-30, 1999.
- Cooper, G.S. & Rezgui, Y.R. (2000). *Objects and Integration in the "Wired Society"*. To appear in: Proceedings of the UK National Conference on Objects and Integration for Architecture, Engineering, and Construction, London, 13-14 March 2000.
- Curtin, M. & Ranum, M. (1999) *The Firewalls FAQ*. 25 November 1999. Cited 9 February 2000. Available on the World Wide Web at <http://www.faqs.org/faqs/firewalls-faq/>.
- Dubois, A.M., J. Flynn, , M.H.G Verhoef & F. Augenbroe, 1995. Conceptual Modelling Approaches in the COMBINE Project, presented in the COMBINE final meeting, Dublin.
<http://erg.ucd.ie/combine/papers.html>

Eastman, C & Augenbroe, F (1998). Product Modeling Strategies for Today and The Future. *Proceedings of the CIB Working Commission W78 Information Technology in Construction Conference*, Stockholm.

Erzberger, M. & Altherr, M. (1999). *Every DAD Needs A MOM: Message Oriented Middleware*. Cited 24 February 2000. Available on the World Wide Web at http://www.softwired.com/pubs/momdad_en.pdf.

Emmerich, W. & Ellmer, E. (1998). *A Survey of Object-Orientated Middleware*. Accepted for the (cancelled) IDPT98 Conference, Turkey but obtained direct from the author.

Emmerich, W. (2000). *Software Engineering and Middleware: A Roadmap*. To appear in: A. Finkelstein (ed): *Future of Software Engineering - State of the Art Reports* given at the 22nd Int. Conf. on Software Engineering, Limerick, June 2000. ACM Press. 2000. Also available on the Internet at <http://www.cs.ucl.ac.uk/staff/W.Emmerich/publications/ICSE2000/SOTAR/index.html>.

Frankel, D.S. (1999) CORBA Components –alive and wll. *Java Report*. October 1999. 70-77.

Gallaughier, J. & Ramanathan, S. (1996). The Critical Choice of Client Server Architecture: A Comparison of Two and Three Tier Systems. *Information Systems Management*. **13** (2) 7-13.

ISO 1988. ISO 9735:1988 Electronic data interchange for administration, commerce and transport (EDIFACT). International Standards Organization. TC 154 (See: <http://www.iso.ch/cate/d17592.html>)

ISO 1994. ISO 10303-1:1994 Industrial automation systems and integration -- Product data representation and exchange -- Part 1: Overview and fundamental principles. International Standards Organization. TC 184/SC 4 (See: <http://www.iso.ch/cate/d20579.html>)

JavaSoft (2000). *The JavaBeans FAQ: General*. Author Unknown. Date Unknown. Cited 7 April 2000. Available on the World Wide Web at <http://www.javasoft.com/beans/faq/faq.general.html#Q15>.

Marir, F and Watson, I A, 1995. *CBRefurb: case-based cost estimation*, Colloquium on case-based reasoning: Prospects for application organised by Professional Group C4 (Artificial Intelligence), 7, March.

Morris, E. & Litvak, E. (1997). *Component Object Model (COM), DCOM and Related Capabilities*. 23 June 1997. Cited 17 January 2000. Available on the World Wide Web at <http://www.sei.cmu.edu/str/descriptions/com.html>.

Orfali, R, *et al.* (1996). *The Essential Client/Server Survival Guide*. Second Edition. New York: Wiley.

Orfali, R., Harkey, D., and Edwards, J. (1997). *Instant Corba*. ISBN 0-471-18333-4 . John Wiley & Sons.

Orfali, R, *et al.* (1999). *The Essential Client/Server Survival Guide*. Third Edition. New York: Wiley.

OMG (1999). *The Complete formal/99-10-07 CORBA/IIOP 2.3.1 Specification*. Available on the World Wide Web at <http://www.omg.org/corba/corbaiiop.html>.

OSMOS (1999). Proposal Part B; *Open System for Inter-enterprise Information Management in Dynamic Virtual Environments*, IST-1999-10491.

Raj (1998). *DCOM, CORBA, Java-RMI - A Step by Step Comparison*. 28 September 1998. (Cited 3 February 2000). Available on the World Wide Web at <http://www.execpc.com/~gopalan/misc/compare.html>.

[RFC1777] “*Lightweight Directory Access Protocol*”, RFC 1777, <ftp://ftp.isi.edu/in-notes/rfc1777.txt>

[RFC1778] “*The String Representation of Standard Attribute Syntaxes*”, RFC 1778, <ftp://ftp.isi.edu/in-notes/rfc1778.txt>

Schussel, G. (1996). *Client/Server Past, Present and Future*. Date unknown. (Cited 17 January 2000). Available on the World Wide Web at <http://news.dci.com/geos/dbsejava.htm>.

Sadoski, D, (1997). *Client/Server Software Architectures -- An Overview*. 2 August 1997. (Cited 17 January 2000). Available on the World Wide Web at http://www.sei.cmu.edu/str/descriptions/clientserver_body.html.

Sadoski, D, (1997b). *Two Tier Software Architectures*. 10 January 1997. (Cited 17 January 2000). Available on the World Wide Web at <http://www.sei.cmu.edu/str/descriptions/twotier.html>.

Sadoski, D, (1997c). *Three Tier Software Architectures*. 10 January 1997. (Cited 17 January 2000). Available on the World Wide Web at <http://www.sei.cmu.edu/str/descriptions/threetier.html>.

SAP, (2000). SAP Tools & Components. <http://saplabs.com/usa/devarea/auto.htm>

SOAP:

- SOAP on MSDN: <http://www.msdn.microsoft.com/xml/general/soaptemplate.asp>
- SOAP Specification: <http://msdn.microsoft.com/workshop/c-frame.htm#/xml/index.asp>
- WSDL specification: <http://msdn.microsoft.com/xml/general/wsdld.asp>

Socolofsky, T, and Kale, C, (1991) *A TCP/IP Tutorial* RFC 1180. Available on the Internet at <ftp://ftp.isi.edu/in-notes/rfc1180.txt>.

Sun (1997). *RMI and IIOP in Java*. Author unknown. 26 June 1997. (Cited 3 February 2000). Available on the World Wide Web at <http://java.sun.com/pr/1997/june/statement970626-01.faq.html>.

Sun (1999). *What is the Java™ Platform?* Author unknown. 19 October 1999. (Cited 3 February 2000). Available on the World Wide Web at <http://java.sun.com/nav/whatis/>.

Sun Microsystems, (2000). <http://java.sun.com/>

TechWeb (1999). *Intranet*. Author unknown. Date unknown. (Cited 26 January 2000). Available on the World Wide Web at <http://www.techweb.com/encyclopedia/defineterm?term=intranet>.

Vanhelsuwé (1997). *Mastering JavaBeans*. Alameda: Sybex. Also available on the World Wide Web at <http://www.lv.clara.co.uk/masbeans.html>.

Vinoski, S. (1998). New Features for CORBA 3.0. *Communications of the ACM*. **41** (10) 44-52.

Vondrak, C. (1997). *Message-Orientated Middleware*. 10 January 1997. (Cited 26 January 2000). Available on the World Wide Web at <http://www.sei.cmu.edu/activities/str/descriptions/momt.html>.

Vondrak, C. (1997b). *Remote Procedure Call*. 10 January 1997. (Cited 26 January 2000). Available on the World Wide Web at http://www.sei.cmu.edu/activities/str/descriptions/rpc_body.html.

Wallnau, K. (1997). *Common Object Request Broker Architecture*. 10 January 1997. (Cited 28 January 2000). Available on the World Wide Web at <http://www.sei.cmu.edu/str/descriptions/corba.html>.

Wallnau, K. and Foreman, J. (1997). *Object Request Broker*. 25 June 1997. (Cited 28 January 2000). Available on the World Wide Web at <http://www.sei.cmu.edu/str/descriptions/orb.html>.

WAP:

- WAP Forum, “Wireless Application Protocol: White Paper”, Wireless Internet Today, June 2000
- WAP Specifications: <http://www.wapforum.com/what/technical.htm>
- WAP Forum: <http://www.wapforum.com>
- Nokia WAP Development Zone: http://www.forum.nokia.com/main/1,,1_1,00.html
- WAP Tutorials: <http://www.waplinks.com/>

Watson, I and Marir, F, 1994. *Case-based reasoning: A review*. The Knowledge Engineering Review. **9**. (4).

Webopedia (1998). *Local Area Network*. Author unknown. 16 May 1998 (Cited 26 January 2000). Available on the World Wide Web at http://webopedia.internet.com/TERM/l/local_area_network_LAN.html.

Whatis.com (1999). *What Is ... the Internet (a definition)*. Author unknown. 25 October 1999. (Cited 25 January 2000). Available on the World Wide Web at <http://www.whatis.com/internet.htm>.

Yee, A. (1999). Making Sense of The COM vs. CORBA Debate. *Performance Computing*. June 1999. Also available on the World Wide Web at <http://www.performancecomputing.com/features/9906dev.shtml>.

Zarli, A. & Richaud, O. (2000). Requirements and technology integration for IT-based Business-oriented frameworks in Building and Construction. Submission to the *Electronic Journal of Information Technology in Construction*.