VTT PUBLICATIONS 375

Embedded middleware:

State of the art

Eila Niemelä, Tomi Korpipää & Arno Tuominen

VTT Electronics

TECHNICAL RESEARCH CENTRE OF FINLAND
ESPOO 1999

ISBN 951–38–5359–4 (soft back ed.)
ISSN 1235–0621 (soft back ed.)

ISBN 951–38–5360–8 (URL: http://www.inf.vtt.fi/pdf/)
ISSN 1455–0849 (URL: http://www.inf.vtt.fi/pdf/)

Copyright © Valtion teknillinen tutkimuskeskus (VTT) 1999

JULKAISIJA – UTGIVARE – PUBLISHER

Valtion teknillinen tutkimuskeskus (VTT), Vuorimiehentie 5, PL 2000, 02044 VTT
puh. vaihde (09) 4561, telekopio (09) 456 4374

Statens tekniska forskningscentral (VTT), Bergsmansvägen 5, PB 2000, 02044 VTT
tel. växel (09) 4561, telefax (09) 456 4374

Technical Research Centre of Finland (VTT),
Vuorimiehentie 5, P.O.Box 2000, FIN–02044 VTT, Finland
Telephone internat. + 358 9 4561, telefax + 358 9 456 4374

VTT Elektroniikka, Sulautetut ohjelmistot, Kaitoväylä 1, PL 1100, 90571 OULU
puh. vaihde (08) 551 2111, faksi (08) 551 2320

VTT Elektronik, Inbyggd programvara, Kaitoväylä 1, PB 1100, 90571 ULEÅBORG
tel. växel (08) 551 2111, fax (08) 551 2320

VTT Electronics, Embedded Software, Kaitoväylä 1, P.O.Box 1100, FIN–90571 OULU, Finland
phone internat. + 358 8 551 2111, fax + 358 8 551 2320

Technical editing Leena Ukskoski

Libella Painopalvelu Oy, Espoo 1999

Niemelä, Eila, Korpipää, Tomi & Tuominen, Arno. Embedded middleware: State of the art. Espoo 1999, Technical Research Centre of Finland, VTT Publications 375. 102 p. + app. 7 p.

Keywords embedded systems, middleware, information systems

Abstract

Java, WWW, Internet and CORBA technologies will be enabling software technologies which make it possible to develop independent applications capable of communicating over various kinds of networks. Thus, embedded programs form networked client/server applications. Within these applications, they perform tasks, functioning as a part of a larger entity in a distributed network.

Currently, firms have their own specialized solutions which require considerable maintenance resources. During development phase, the interoperability and extendibility of the systems are not given enough attention. Because of this, the solutions become rapidly obsolete. For several enterprises, a standardised solution for communications is in demand.

Middleware is a general term that has come to represent a variety of distributed computing services in application development environments. Middleware products operate between the application logic and the underlying physical network. Embedded middleware provides standard communication services and object-oriented integration interfaces for networked embedded applications.

The maturity of the commercial middleware products based on CORBA is insufficient and some of the services required in embedded systems are still unavailable. DCOM-based OPC will be a suitable solution for base stations that control data acquisition, monitoring and presentation. However, before it can be considered a suitable solution for client application executed in laptops or handheld PCs, OPC requires commercially available driver software.

Embedded middleware requires communication services for remote procedure calls, events and messaging. If the system is connected to the Internet or other open networks, naming and security services are required as well. These services, and the generic interfaces which isolate the middleware from the operating system and protocols, have to be provided as component-based software which enables restricted adaptability during product life cycle. In future, applications will be composed of different types of components such as Java beans or applets, ActiveX components or CORBA components. Each of these component types requires its own interface technology.

Preface

This report is based on a technology survey in the VERSO project during the spring 1998 at VTT Electronics. The objective of the VERSO project was to investigate new software technology concepts for networked embedded applications. The project was carried out by surveying the advanced middleware technologies and by developing a new middleware concept for embedded networked applications. The VERSO project is a three- year strategic research project funded by VTT Electronics.

This state-of-the-art report describes the results of a technology survey concerning current techniques for networked systems based on client/server architectures and middleware software. One of the main trends in embedded systems is system integration by Internet. Therefore, the focus is on the Web and object technologies as well as on commercially available integration software for heterogeneous networked systems.

Oulu, Finland, July 1998

Eila Niemelä Tomi Korpipää Arno Tuominen

VTT Electronics

Embedded Software

Contents

List of abbreviations

ADE	Application Development Environment
AIP	Active Internet Platform
API	Application Programming Interface
ATM	Asynchronous Transfer Mode
AWT	Abstract Windowing Toolkit
BLOB	Binary Large Object
CDMA	Code Division Multiple Access
CGI	Common Gateway Interface
CICS	Customer Information Control System [IBM]
CMC	Common Mail Calls
CMIP	Common Management Information Protocol [OSI]
CORBA®	Common Object Request Broker Architecture
CPI-C	Common Programming Interface for Communications
DBMS	Database Management System
DCE	Distributed Computing Environment [OSF]
DCOM	Distributed Component Object Model
DCS	Data Capture System
DME	Distributed Management Environment
DMF	Distributed Management Framework
DMI	Desktop Management Interface
DNS	Domain Name Service
DTP	Distributed Transaction Processing
EDI	Electronic Data Interchange, Electronic Document Interchange [DEC]
FDDI	Fiber Digital Device Interface Fiber Distributed Data Interface
FAT	File Allocation Table
FSM	Finite State Machine
FTP	File Transfer Protocol
GSM	Global System for Mobile Communications
GUI	Graphical User Interface
HDML	Handheld Device Markup Language
HPC	Handheld PC
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IETF	Internet Engineering Task Force
IIOP™	Internet Inter-ORB Protocol
IIS	Internet Information Server
IMS	Information Management System
IPC	Interprocess Communication
IPX/SPX	Internet Packet Exchange/Sequenced Packet Protocol

®CORBA is a registered trademark of Object Management Group, Inc.

IOR	Interoperable Object Reference
ISAPI	Server Application Programming Interface [Microsoft]
ISDN	Integrated Services Digital Network
ISO	International Standards Organization
ITU	International Telecommunication Union
JDBC	Java Database Connection
JOE	Java Object Environment
LAN	Local Area Net
MAPI	Messaging API
MIME	Multipurpose Internet Mail Extensions
MMU	Memory Management Unit
MOA	Message Object Adapter
MOM	Message-Oriented Middleware
MOMA	Message-Oriented Middleware Association
MORB	UI Management ORB
NDIS	Network Driver Interface Specification
NDR	Network Data Representation
NetBEUI	Transfer protocol used with IBM/Microsoft LANs
NetBIOS	Interface for transfer protocols with several OSs [IBM]
NSAPI	Netscape's Server Application Programming interface
OCX	OLE Custom Control
ODBC	Open Database Connectivity [Microsoft]
ODI	Open Datalink Interface [Novell]
	Open Device Interconnect [NetWare]
OLE	Object Linking and Embedding
OMG™	Object Management Group
OOUI	Object Oriented User Interface
OPC	OLE for Process Control
ORB™	Object Request Broker
OSI	Open Systems Interconnection Model
PCMCIA	Personal Computer Memory Card International Association
PIDL	Pseudo Interface Description Language
POA	Portable Object Adapter
POSIX	Portable Operating Systems Interface
PPP	Point-to-Point Protocol
PVC	Permanent Virtual Circuit
QoS	Quality of Service
RMI	Remote Method Invocation
RPC	Remote Procedure Call
S-HTTP	Secure Hypertext Transfer Protocol
SCADA	Supervisory control and data acquisition
SDK	Software Developer's Kit
SFA	Server Framework Adapter
SGML	Standard Generalised Markup Language
SLIP	Serial Line Internet Protocol
SMDS	Switched Multimegabit Data Services
SMP	Symmetric Multiprocessing
SMTP	Simple Mail Transfer Protocol
SNA	System Network Architecture

SNMP	Simple Network Management Protocol [IETF]
SQL	Structured Query Language
SSI	Static Invocation Interface
SSL	Secure Session Layer
SVC	Switched Virtual Circuits
TCP/IP	Transmission Control Protocol/Internet Protocol
TI-API	Transport-Independent API
TLI	Transport Layer Interface
UDP/IP	User Datagram Protocol/Internet Protocol
UI	User Interface
URL	Unified Resource Locator
UUID	Universal Unique Identifier
VRML	Virtual Reality Modelling Language
W3C	World Wide Web Consortium
WAIS	Wire Area Information Server
WAN	Wide Area Network
WAP	Wireless Application Protocol
WPMS	Workflow and Process Management Systems
XMP	X/Open Management Protocol
XOM	X/Open Management APIs, X/Open Object Manager
X.400	Data Exchange protocol, BLOBs, EDI, CMC API
X.500	Directory Services, Security

1. Introduction

Owing to the standardisation and price reduction of embedded system platforms, state-of-the-art hardware technology is becoming increasingly accessible. Similar development can be detected with software; although fitting together software from various vendors may prove troublesome.

In future, software systems will be based on Java, WWW, Internet and CORBA. These technologies enable developing independent applications capable of communicating and interoperating over networks. This being the case, embedded programs form networked client-server applications. Within these applications, the programs perform as parts of a larger entity in a distributed network.

Nowadays, firms have their own specialised solutions, which requires considerable maintenance resources. Development is not given sufficient emphasis and therefore, the solutions become rapidly obsolete. Therefore, standardised solutions for communication are required.

Embedded middleware, a standardised object-oriented application interface for supporting distribution of networked embedded applications, would solve the networking problems of communication applications. Firms have a demand for an embedded middleware framework that would enable building up embedded networked systems. Such systems enable using new operations models such as remote testing, updating, and marketing during product life cycle.

This report is a state-of-the art survey concerning the middleware and techniques used in client/server architectures. The purpose of this report is to

- describe different middleware technologies used in information systems,
- evaluate technologies which are commercially available for embedded Web-based systems,
- describe features and experiences of commercial products for heterogeneous distributed client/server systems and
- define what services are required by embedded client/server systems which are based on component-based software architecture and standard interfaces.

Chapter 2 describes the background of heterogeneous client/server architectures as well as current solutions applied for achieving interoperability. Architectures, distribution media, operating systems and software are discussed.

Chapter 3 focuses on commercial products which can be used in heterogeneous client/server information systems. Products covered include three ORBs and a DCOM based solution.

Chapter 4 describes experiences gained of the evaluated commercial middleware.

Chapter 5 describes the services required in embedded middleware.

Chapter 6 offers some conclusions based on the theory and experiences.

2. Background of heterogeneous networked systems

2.1 Client/server architecture

Clients and servers are separate logical entities that work together over a network in order to accomplish a task. The typical characteristics for client/server systems are the following (Orfali et al. 1996):

- Client/server is a relationship between server and client processes. The server process provides services for clients, which are the consumers of services.
- A server controls clients' access to shared resources.
- Servers are passive in that they stand by for messages from clients. Clients initiate the dialogue by requesting a service.
- Client/server software masks the server location from the clients. A program can be a client, a server, or both.
- The client/server software is independent of hardware and operating system software platforms.
- Clients and servers are loosely coupled and interact through a message-passing mechanism. The server code and data are centrally maintained in order to reach independent clients.
- The services are encapsulated to the server which can be upgraded without affecting the clients. The server determines independently how perform the requested task.
- Client/server systems can be scaled horizontally and vertically. Horizontal scaling refers to adding and removing clients. Vertical scaling refers to employing faster or greater multiple servers.

There are several types of client/server systems owing to the diversity of target applications. A file server lends itself to sharing files across a network. Database servers return the results of each SQL command to the client. The processing code and the data are located on the same machine. With a transaction server, the client/server application can be built by writing the code for both the client and the server components. Groupware servers address the management of semi-structured information. In most cases, applications are created by using scripting language and form-based interfaces. As for an object server, the client/server application is written as a set of objects, which communicate by using Object Request Brokers (ORB). The World Wide Web is a universal client/server

application which consists of thin, portable, universal clients that communicate with fat servers by using HTTP (Hypertext Transfer Protocol), which is a protocol resembling RPC (Remote Procedure Call).

2.1.1 2-tier and 3-tier client/server architectures

The most typical functional units in a client/server architecture are the user interface, the business logic, and the shared data. In 2-tier client/server systems, the application logic is hidden either inside the user interface on the client side, or within the database on the server, or both. For example, file servers and database servers with stored procedures are 2-tier clinet/server architectures.

In 3-tier client/server systems, the application logic is executed in the middle-tier and remains separated from the data and the user interface. 3-tier client/server systems are more scaleable, robust, and flexible. They can also integrate data from multiple sources. The Web, TP Monitors and distributed objects are examples of 3-tier clinet/server systems. They manage the application processes independently both from the database and the GUI front-end. In a networked embedded system area, a typical 3-tier client/server architecture consists of

- a data acquisition and process control tier, which is a hard real-time level,
- a data analysis and storage tier, which is often a soft real-time level, and
- an interface and co-ordination tier without real-time requirements.

As an example of a 3-tier client/server architecture, the Open Object Web combined with embedded systems is presented in Figure 1. Clients use component-based browsers to visualise or execute HTML documents, forms, Java applets, Java beans, compound documents and shippable places through OpenDoc titles. If an OpenDoc title is a shippable place, it can contain ActiveXs, OpenDoc parts, Java applets, and regular HTML content. Clients communicate via the Internet by using different protocols such as HTTP and CORBA IIOP. The second tier services both HTTP and CORBA clients. CORBA objects encapsulate the application’s logic by for example storing and analysing measured data. The objects interact with clients via a Java ORBlet or through any regular CORBA ORB that can run IIOP over the Internet. They can also communicate with existing hard real-time applications on the third tier by using CORBA wrappers for heterogeneous communication media (Niemelä & Holappa, 1998).

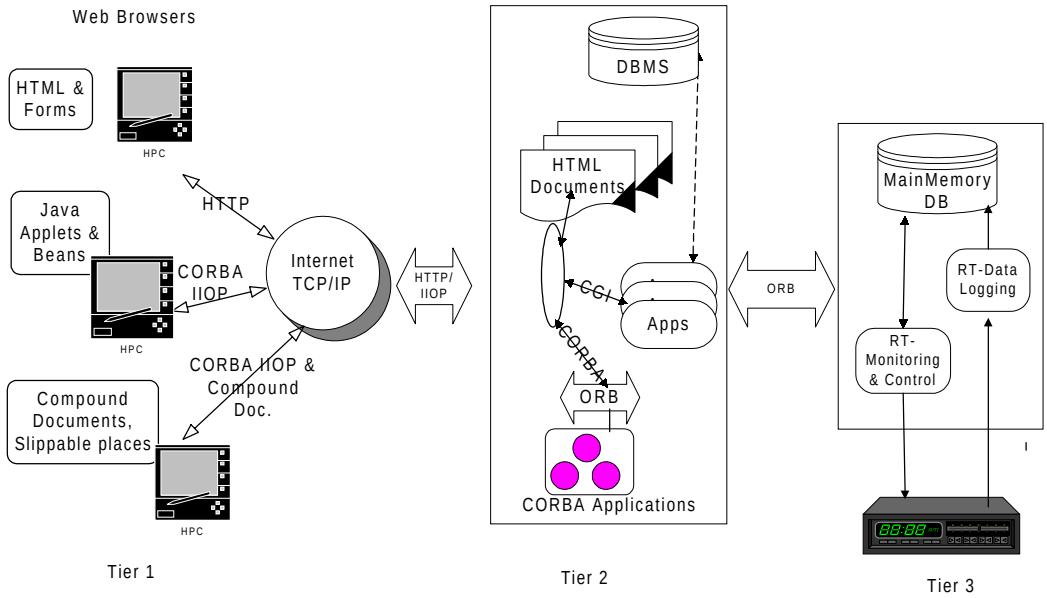

Figure 1. The 3-tier client/server architecture of the Open Object Web in embedded systems.

2.1.2 Middleware

Middleware is a general term referring to a variety of distributed computing services in application development environments. Middleware products operate between the application logic and the underlying physical network and encompass a wide range of services and products including message-queuing, application development environments, object development environments, database access, distributed transaction processing, message communications, and RPC-based communications. Many of these products and services provide overlapping functionality. However, in order to acquire all the necessary tools to develop and deploy an enterprise-wide distributed application, several vendors have to be contacted. Middleware services can be classified as follows (MOMA, 1998):

- **Application development environments** (ADEs) include rapid application development tools and cross-platform development tools. ADEs provide a high-level development language and include tools that facilitate cross-platform applications by accommodating differences in operating environments and user interfaces. ADEs require additional services such as network communication, application partitioning and distribution services, as well as component location services. These services can be included in the ADE, or the ADE may rely on other middleware and communication products.

- **Object development environments** are designed for developing reusable software components. In a distributed environment, components interact through an object request broker (ORB). ORBs can interact also with, and rely

- on other forms of middleware for application communication and distributed services.

- **Database management systems** need middleware which allow developers to view disparate data sources in a consistent way. Database middleware is targeted at providing a solution for two-tier architectures where the data flow across the network to and from a database server on a remote machine.

- **Message-Oriented Middleware** (MOM) is an enabling software layer between the business applications and the network infrastructure. It supports high-performance interoperability of large-scale distributed applications in heterogeneous environments. In addition, it supports multiple communication protocols, languages, applications, as well as hardware and software platforms. It resides between the business applications and the network infrastructure, or between applications themselves. MOM refers to the process of data and control distribution through the exchange of messages. It provides message passing or message queuing models, supporting both synchronous and asynchronous communications. Time dependent and time-independent processing, as well as memory and disk-based systems are available.

- **Electronic mail** offers point-to-point storage and forward technology that generally requires a desktop and a LAN-centric environment. Key standards in this area include X.400, X.500, SMTP, MAPI, and MIME.

- **The Remote Procedure Call** (RPC) is a well-established method for distributing application processing while simultaneously concealing the complexities of networking communications. In a RPC, control is passed from one procedure to another, and the call is blocked until control is returned, which means that RPC communication is inherently synchronous. Asynchronous communication is managed by using threads. RPCs generally provide data format translation services.

- **Distributed Transaction Processing** (DTP) systems typically target high-end businesses with critical customer requirements. DTP systems offer services that focus on application management, administrative controls, and application-to-application message passing. DTP includes global transaction co-ordination, distributed 2-phase commit, resource manager support, co-ordinated recovery, high availability, security, and workload balancing. The key DTP standards are driven by ISO OSI and X/Open Company Ltd.

- **Workflow and process management systems** (WPMS) are designed to facilitate the automation of repetitive processes within an organisation or enterprise. These systems focus on how work is passed from one state to another (*routing* algorithms), how the routing decision is made (*rules*), and what are the relationships between the steps (*roles* played by each step of the process). WPMSs are usually state-table based or database-engine based,

- depending on how they store routing, rules, and role information. One of the key organisations in this area is the Workflow Management Coalition.

- **Other Distributed Services** are typically based on middleware solutions. They include time services which ensure that all internal clocks in distributed client and server machines are synchronised within an acceptable level of variance. Directory services provide the essential elements to be named and identified, thereby enabling location and routing independence. Security services provide authentication, authorisation, auditing, and encryption.

- **Mobile Computing** refers to the capability of distributing application solutions over wireless networks. Wireless message-oriented middleware uses messages to communicate between applications and across the wireless networks, without the necessity to embed low level communications protocols in the application. The middleware can support multiple protocols, accessibility to wire-line applications, multiple hardware and software platforms. Furthermore, it enables making modifications to the underlying environment without the need to alter the application.

- **Generic application platforms** for distributed systems, e.g. flexible manufacturing systems, data acquisition and diagnostics systems etc., consist of highly independent applications at different real-time levels, co-operating with each other through standard communication services, e.g. an embedded middleware.

2.2 Distribution media

The middleware should hide the diversity of protocols and communication media. Multiprotocol API facilitates developing client/server applications that run over multiple protocol stacks. A standard interface between network interface and protocols is required by middleware developers (Figure 2). This implies that the developers can use or change the distribution media and protocols without affecting the applications.

The logical network driver provides a single interface for all the network adapters, and is required between the transport stack and network drivers. Microsoft/3Com's NDIS and Novell’s ODI are the two most widely supported de-facto standards for interfacing protocol stacks to network adapter device drivers.

The transport-independent APIs are located on top of the transport stacks and enable plugging applications into a single interface that supports multiple protocols. The sockets interface and the transport layer interface (TLI) are the most widely used TI-APIs (Windows, NetWare, Unix). The following stacks can be used: SNA and TCP/IP in CPI-C API and NetBIOS, IPX/SPX, TCP/IP in Named Pipes.

The protocol matchmakers enable an application, which is written for a specific transport, to run across other networks. They eliminate the need for gateways, as can be seen for example with IBM's AnyNet product. (Orfali et al. 1996).

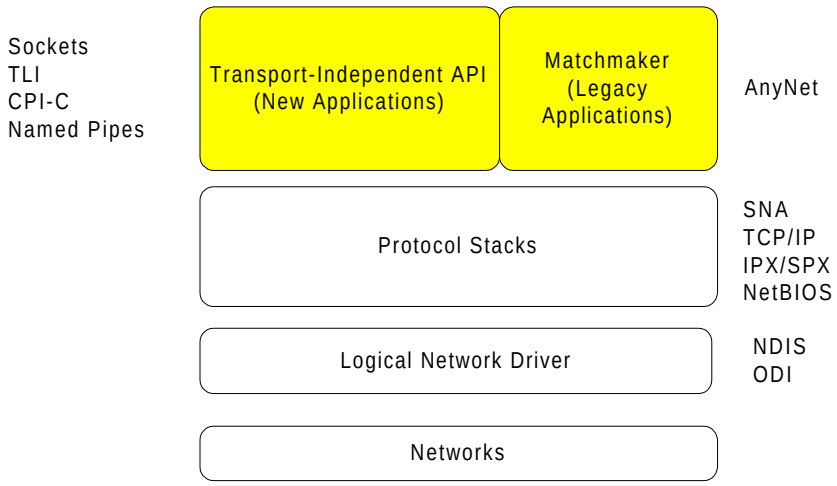

Figure 2. The bottom layers of the middleware.

2.2.1 LAN and WAN

The most common types of LANs and WANs are presented in Table 1. The Ethernet is widely used (75 % market share), but Fast Ethernet is growing rapidly. The use of ATM (Asynchronous Transfer Mode) is limited but growing, due to the increasing number of multimedia applications which require rapid data and video transfer. SMDS is used mostly in Europe as a precursor to ATM. SMDS supports variable-length packets that can be broken into ATM-size fixed cells in order to facilitate the transfer. Currently available SMDSs run at 45 Mbit/s. As for WANs in USA, Frame Relay is currently the most popular packet-switching technology. Frame Relay can route variable-length packets over existing routers at speeds between 1.54 / 20.4 Mbit/s. In addition, Frame Relay routes and assigns error checking to be executed by upper protocol layers (Orfali et al. 1996).

Currently, it would appear that ATM local area networks will be used only in special applications that require strict QoS management. ATM technology appears to be dominant in backbone networks, but in LANs, Gigabit Ethernet solutions will be dominant, owing to simpler implementation and greater cost effectiveness.

Table 1. LAN and WAN transmission technologies.

Type	Speed	Typical applications	Used as
WLAN	1-2 Mbps (IEEE802.11 compliant)	data, audio, video	LAN
WLAN	11 Mbps, IEEE is working for a standard	data, audio, video	LAN
Ethernet	10 Mbit/s	data, audio, video	LAN
Token Ring	4/16 Mbit/s	Data	LAN
Fast Ethernet	100 Mbit/s	data, audio, video	LAN
FDDI	100 Mbit/s	data, audio, video	LAN
Gigabit Ethernet	1 Gbps	data, audio, video	LAN
ATM	25 Mbit/s - 2.4 Gbit/s	data, voice, video	LAN, WAN
Frame Relay	1-2 Mbit/s	Data	WAN

ATM is a packet-switching protocol that achieves very high speed by using fixed-length data cells or packets on the top of virtual circuits. Permanent Virtual Circuits (PVCs) are statically assigned; Switched Virtual Circuits (SVCs) are dynamic. A virtual circuit guarantees a good quality of service, bandwidth and priority included. (Orfali et al. 1996).

Wireless LANs enable short-range (i.e. max. of few hundred meters) communications for laptops and handheld devices, with typical speeds of 1-2 MBps to 11 MBps. Although WLANs are usually used to extend existing LANs, they also provide standalone networks for ad hoc networking or for mobile environments.

2.2.2 Cellular data communication

Cellular communication enables a remote access between mobile terminals and the Internet. World-wide mobile cellular access is not yet available but for many practical applications, cellular coverage (e.g. GSM having coverage in 100 countries) is sufficient (World-wide coverage with low speed data will become possible via e.g. Iridium satellite system). The diversity of cellular standards follows major market areas which divide the world into the following user

systems: Europe (GSM, GSM1800), Japan (JDC, PHS) and USA (IS-95, IS-54, GSM-1900). However, for the purposes of end-users, speed, reliability, connection manner, package and circuit connection are the only relevant features.

Mobility in cellular systems cannot be achieved without expenses. If compared to fixed telephone line communications, mobile cellular systems have some limiting features. For example, costs for Internet usage are higher in cellular connections than in fixed line connections. This is a severe limitation for recreational web browsing, if not for professional usage.

In addition, if analog modems are used, the transmission speed is about one-fourth of that offered by fixed telephone lines, while it is much higher with ISDN cards. Therefore, various groups are working to enable faster speed over digital cellular networks. For example, it is very likely that in a few years' time, GSM will provide data transmission rates up to 384 kbits/s for urban areas.

The shortcomings of wireless technology performance can be attributed to limited bandwidth. Bandwidth limitations can be eliminated by

- Optimising bandwidth usage by stripping graphics and nonessential data from transmission.
- Creating a document-formatting language for wireless-applications.
- Creating new networks equipped with more bandwidth.

One solution that optimises wireless connections is Spyglass's Prism, a content conversion product which downloads and displays data two to four times faster. Web pages are cached and only those elements which are altered, are updated from the server.

Unwire Planet has developed the UP.Link Platform, which includes Handheld Device Markup Language (HDML). HDML is similar to HTML but has been scaled down and optimised for secure wireless Internet or intranet transmissions over packet-sized devices.

ITU is working on open standards that will enable wireless multimedia communications with bandwidth-on-demand capabilities. Wireless vendors have created a Wireless Application Protocol (WAP) which makes mobile wireless Internet technology open and scalable. The specification covers layers 4, 5, and 6 of the OSI stack, as well as security features. WAP is based on UP.Link Platform. (Parker 1998).

2.3 Operating systems

2.3.1 Requirements for the client-side

Client programs require an object-oriented user interface which is an extension of the operating system's user interface. This being the case, the Object-Oriented User Interface (OOUI), and the operating system can not be clearly distinguished. Reality-simulation is managed by drag-and-drop mechanism, icons and direct manipulation. Compound documents (OLE, OpenDoc) exist already, but 3-D compound documents (such as VRML, Virtual Worlds) are yet to be realized. Owing to Web technology, servers can send Java-manipulated HTML pages to their clients. Shippable compound documents can contain shippable places, which form Java components. Unlike a page, a place does not disappear after switching pages; on the contrary, it can exist on the desktop for as long as is required.

OOUI clients set the following requirements for the operating system (Orlafi et al. 1996):

- Local/remote transparency request/reply mechanism.
- File transfer mechanism which enables text, pictures, movies and database snapshots to be moved.
- Pre-emptive multitasking.
- Task priorities.
- Interprocess communications.
- Threads for background communications with server and for receiving callbacks from servers.
- OS robustness, including intertask protection and re-entrant OS calls.
- Window 3.x GUI, OOUI and compound documents.

2.3.2 Requirements for the server-side

The server which serves multiple clients sets the requirements for the operating system concerning real-time scheduling, task execution or service management (Table 2).

Table 2. Classified requirements for servers.

RT Scheduling	Pre-emptive scheduling allows fixed time slots for the execution of each task and enables easier creating of safer server programs.
	Task priorities enable servers to differentiate the level of service on basis of client priority.
	Concurrence control with semaphores is used to synchronise the actions of independent server tasks and shared resources.
Task Execution	Interprocess communications (IPC) allow independent processes to exchange and share data.
	Threads provide the concurrence control within a process itself and are used to create event-driven server programs.
	Intertask protection. The operating system must protect tasks from interfering with each other's resources. Protection also extends to the file system and calls to the operating system.
Service Management	Multi-user high-performance file system. The file system must support multiple tasks and provide locks to protect the integrity of the data. In addition, a large number of open files must be supported without a decrease in performance.
	Efficient memory management is needed for large programs and large data objects which must be easily swapped to and from disk, preferably in small granular blocks.
	Dynamically linked run-time extensions must be supported by a mechanism which, without recompiling, provides additional services at run time

There are also some extended services that provide flexible access to shared information and make the system easier to manage and maintain (Figure 3). The extensions must provide a rich set of protocol stacks that allow communication with a large number of client and other server platforms. (Orfali et al. 1996).

File and print services must be provided over the network. Binary large objects (BLOBs) require extensions of intelligent message streams (protocols) and object representation formats. Databases and file systems must be able to store and access BLOBs. Network resources must be locatable by name. Furthermore, the extensions must provide a way for clients to locate servers and their services on the network by using a global directory service. A client requires an authentication service to identify itself to the server. An authorisation service determines if the authenticated client is given permission to obtain a remote service. The extensions must provide an integrated network and a system management platform. The system management includes services for configuring a system and facilities for monitoring the performance of all elements, including distributing and managing software packages on client workstations. The operating system extensions must provide a mechanism which co-ordinates the global time and synchronises client and server clocks.

Transaction services are required for a robust multi-user Database Management System. The system can be supported by a Transaction Processing Monitor (TP Monitor) for managing stored procedures as atomic work units that execute on one or more servers.

For Internet services the server requires Hypertext Transaction Protocol (HTTP) daemons, Secure Sockets Layer (SSL), firewalls, Domain Name Service, Hypertext Markup Language (i.e. HTML)-based file systems, and electronic commerce frameworks (which, however, are optional). The extended services of the operating system must include object interchange services and object repositories. (Orfali et al. 1996).

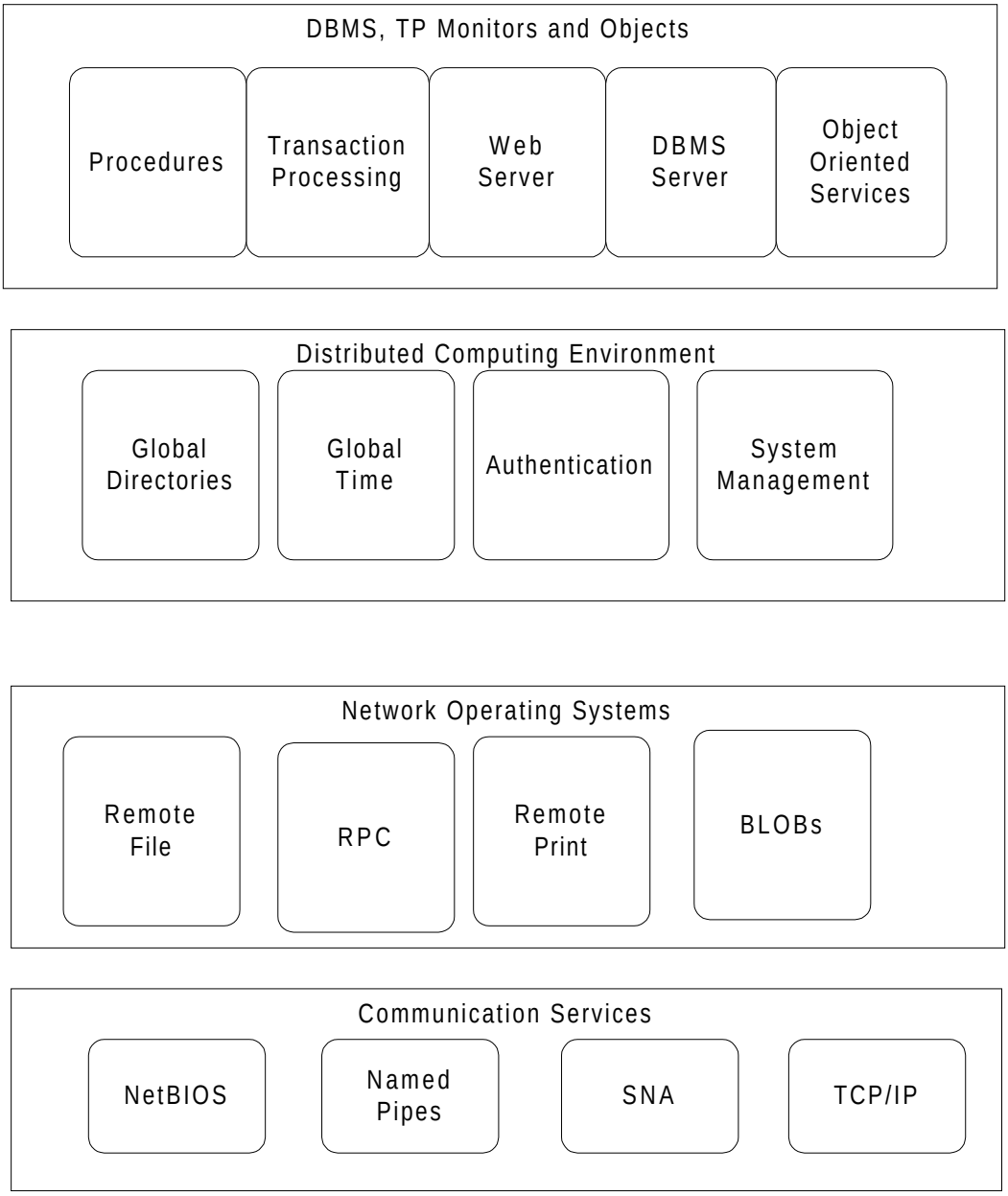

Figure 3. Server programs' requirements for the extended operating system.

2.3.3 Operating systems for clients

2.3.3.1 Windows 95 (Microsoft)

Pros:

OOUI, pug-and-play, hardware autodiscovery, network neighbourhood, remote registry editor, built-in SNMP agent, minimal TCP/IP stack, NetBEUI, IPX/SPX and PPP.

Cons:

Inconsistent OOUI (mixed OOUI and GUI paradigms), built on DOS (16-bit application, no crash protection, limited multitasking, no pre-emptive scheduling), not robust enough for corporate client market. (Orfali et al. 1996).

2.3.3.2 Windows NT workstations (Microsoft)

Pros:

32-bit client OS, pre-emptive multitasking, multithreading, memory protection, transactional file system, completed network: TCP/IP, NetBEUI, IPX/SPX, PPP and AppleTalk. C2-level security, and NT 4.0 supports DCOM.

Cons:

Resource hog (minimum of 16 Mbytes of RAM and 512 Mbytes of disk) and an expensive client platform. Poor support for the following: laptops (limited PCMCIA support and power management), emulation of DOS and 16-bit Windows applications, device drivers. Difficult configuration owing to lack of support for virtual device drivers (VxDs) and plug-and-play components. (Orfali et al. 1996).

2.3.3.3 Java OS (Sun)

Real-time operating system industry standardises on PersonalJava and EmbeddedJava which are designed for environments with limited resources, with the addition of specific features required by consumer applications. (Sun 1998).

Pros:

Dynamic linking and loading (remote updates and batching), object-oriented, security and reliability features, portability across platforms. PersonalJava API includes: Java Virtual Machine, Java classes, internationalization (I18N) support, Java beans, Java applets, networking, and PersonalJava AWT. Java AWT is targeted at and tuned for consumer product purposes.

Cons:

Slow performance; it has benchmarked as slow as 1/40th the speed of C++. Windows systems would appear to require at least 20M of RAM to run Java adequately. A Java virtual machine with a Motif-based GUI, MPEG, JPEG, Internet support and other class libraries runs to over 11MB of code. A smart-card Java requires a dozen kilobyte. Personal and Embedded Java will be somewhere between the two mentioned last. (Dibble, 1997).

Applications:

Hand-held computers, set-top boxes, game consoles, mobile hand-held devices and smart phones. The platform is scalable and configurable including a Java Virtual Machine, a set of core libraries, and optional libraries. Runs also on enterprise Java platforms.

2.3.3.4 Windows CE (Microsoft)

Pros:

Low memory costs due to a ROM file system from which OS is booted. RAM is used only for stack, heap, and global variables. Typical memory requirements are 2 KB of ROM, and between 300KB and 1MB of RAM. OS supports multithreading and multitasking with memory protecting (MMU required). It supports TCP/IP, UDP/IP and SLIP protocols as well as WinSock 2.0. A PPP layer will be underneath the IP stack for a serial cable and modem, and IrDA 1.0 stack for infrared connections. (Murphy, 1997).

User interaction is supported by a display, a touch panel or mouse, a keyboard, audio output and PC cards. Windows CE supports both volatile and non-volatile storage systems and includes a FAT file system and ATA drivers. A custom block device driver can be used with the FAT file system for providing a block storage device on the Flash card. Windows CE SDK is a subset of the Microsoft Foundation Classes (MFC). Windows CE 2.0 with ActiveSync synchronises data between a HPC (Handheld PC) and Windows-based computers. Beta of the Windows CE Toolkit for Visual J++ 1.1. is available. (Murphy, 1997).

Cons:

No nested interrupts (interrupt latency), no hard RT OS. No support for the Network Driver Interface Specification (NDIS). No support for modifying the protocol stacks. Adding third party built-in peripherals or drivers to a Window CE is not possible, except by supplying drivers for PC Cards and using Windows CE SDK. A new file system cannot be added.

Applications:

Handheld computers, consumer products and various non-consumer products. (Murphy 1997).

A summary of the operating systems for embedded applications is presented in Appendix A.

2.3.4 Operating system products for servers

2.3.4.1 NetWare 4.1 (Novell)

Pros:

Effective, well-supported file server, global directory, threads across processors (NetWare 4.1 SMP), C2-level security in future releases, built-in support for the Internet, including a server-side Java environment.

Cons:

Lack of openness, poor application support; NetWare Loadable Modules in NetWare 3.1, limited memory protection, lack of memory management, virtual memory not provided, no support for pre-emptive multitasking. (Orfali et al. 1996).

2.3.4.2 NT server (Microsoft)

Pros:

Additional features to the NT workstations: file/print server support, built-in Internet server, disk mirroring and striping, SMP. Processors: Intel, Alpha, PowerPc, MIPS. TCP/IP, support for NetWare, Network OLE, enhanced security, multiprotocol routine and ISDN communications. Natural server in Microsoft environments. Easy installation, management and configuration.

Cons:

Limited scalability, no enterprise directory server. Security holes, limited backup facilities , unsatisfactory enterprise services and support. (Orfali et al. 1996).

2.3.4.3 OS/2 Warp server (IBM)

Pros:

32-bit OS (including Lotus Notes and CORBA services, database, middleware, system management and communications offerings) is as well-equipped as its Unix counterparts, but easier to install, use, and manage. Fast file and print server, OOUI (installed with autodetecting hardware, configuration, and system management), disk mirroring, remote administration, remote software distribution, backup server, and software metering. Eagle: database, Lotus Notes, Internet commerce, TP Monitor, CORBA object services, DCE security and directory.

Cons:

Intel-only platform, maximum file size of 2 Gbytes, maximum disk partitions of 512 Gbytes (multimedia). C2 level security, unicode support (for internationalisation), support for megaclusters, transactional file systems, and memory-mapped files are unavailable (although may be included in OS/2 Merlin Server). (Orfali et al. 1996).

2.3.4.4 Unix

Pros:

Scalable from the desktop to the supercomputer. Java, CORBA, 64-bit standard in the future.

Cons:

45 variants of Unix, lack of binary compatibility, functional difference among the Unixes.

Running the same OS on clients and servers makes LANs simpler to administer. Similarly, moving programs between clients and servers is more simple. The same installation procedures can be used. (Orfali et al. 1996).

2.3.5 Summary

Operating systems vary both in the client and the server side with reference to software flexibility, supported protocols and their interfaces, concurrence support by multitasking, multithreading and memory protection, required memory size and offered memory protection. In many respects, this is reflected on the requirement to isolate the operating-system-dependent and protocol-dependent parts from the embedded middleware. However, this enables changing underneath software platform with minimum side-effects to the embedded middleware itself. Furthermore, services common to both the operating systems and the protocols should be identified. If a generic solution for embedded middleware is explored, only these combined features should be considered to form the basis of the middleware.

2.4 Frameworks and components

Distributed computing relies on a set of services for accessing and managing shared services, information, and computing resources. Application components require the following types of distributed services (Krieger & Adler 1998):

- *Remote communication protocols* enable components to interact at the application layer. Protocols can be synchronous or asynchronous.

- *Directory services* provide a global scheme for naming, organising, and accessing shared services and resources.

- *Security services* protect shared resources by authentication and authorisation and block third parties from intercepting messages.

- *Transaction services* co-ordinate concurrent updates and ensure that updates leave data in correct and consistent states.

- *System management services* provide a unified set of facilities for monitoring, managing, and administering services and resources across the enterprise.

The existing standards support the framework development for distributed computing by providing specifications for all or for some of the required services. In addition, services for specialised application components are provided. This section describes commonly used frameworks and components which they support.

2.4.1 CORBA (Common Object Request Broker)

Object Management Group (OMG) has specified a de-facto standard for the CORBA framework. It provides an integration infrastructure for distributed business objects. The Object Management Architecture (OMA) describes the enterprise integration by providing a component-based software environment which uses the services generated by the underlying Object Request Broker (ORB). The OMA is divided into three major components: lower-level CORBAservices, intermediate-level CORBAfacilities, and CORBAdomains which are used by distributed application objects.

CORBA has the following objectives (Sigel 1996):

- To provide a *location-independent* software platform for distributed objects,

- to provide a *language-independent* application development environment,

- to provide *platform-independent* communication mechanisms, i.e. operating systems, and protocols that do not affect applications,

- to provide a *standard interface* description language and a thin layer of wrapper code for legacy applications,

- *to maximise* programmer *productivity* by transparent distribution and by easily accessible components,

- to re-use software by black-box and white-box methods,
- to enable *mixing and matching tools* within a project by smooth interoperability.

The key characteristics that enable a software component plug-and-play to exist as an object in the CORBA environment are encapsulation, inheritance, and polymorphism.

The encapsulated software component consists of two distinct parts: an interface, which the component presents to the outside world, and implementations, which are kept hidden. The interface presents the contract of the object. If the interface contains more than one implementation of an object, the object can be substituted by another. The client would remain unaware of the alteration, since the responses to its messages would not change. Encapsulation enables CORBA to provide location transparency. A client sends the invocation to the local ORB, and not to the target object itself. Finally, the ORB routes the message to its destination. Encapsulation is the cornerstone of CORBA. It is used also when legacy applications are wrapped.

Inheritance is an object-oriented method that reduces the workload of designers and programmers, since it enables existing object templates to serve as the basis for new ones. Object-oriented languages take advantage of inheritance which is included in CORBA as well. By polymorphism, an operation is invoked on a set of objects. The operation generates different outcomes, since with each object, the message triggers individual methods. As inheritance and polymorphism enable CORBA to work with object-oriented tools and languages, these features have been included in OMG IDL (Siegel 1996).

CORBAservices provides basic functionality required by any object (Figure 4). CORBAfacilities provides upper-level services common for all application domains. CORBAdomains include domain-dependent services. These, in their turn, use the services provided by CORBAservices and CORBAfacilities. Applications can use services generated by all of the levels. Services can also be added into each category in order to provide all necessary services.

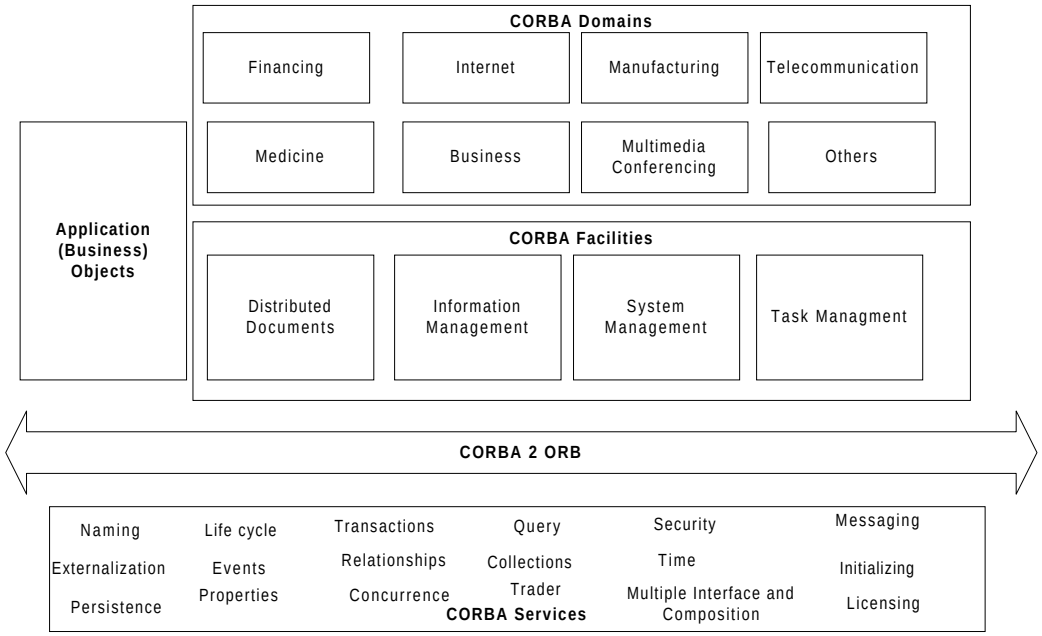

Figure 4. An overview of the CORBA framework.

2.4.2 COM and DCOM (Distributed Component Object Model)

COM (Component Object Model) by Microsoft provides a system aimed for providing joint functionality between applications. COM performs this requirement by dividing an application into a series of objects that can be passed around and used by any other application. COM is the technology underlying OLE (Object Linking and Embedding) which enables data from a number of different applications to appear in the same document. (Blaszczak 1997).

Component Object Model is a specification that describes what an object is, how it functions during its lifecycle, and how it communicates with the outside world. COM is a binary specification: it is not a language, and it does not require the usage of any particular language. Any language which has support for arrays of function pointers and which can call functions through those pointers, can be used directly. This includes languages like C, C++, and Pascal. Other languages that do not have direct support for function pointers, like VB and J++ (Microsoft's Java implementation), have extensions that allow them to call or create COM objects. J++ has classes that use native code for accessing COM objects whereas VB has developed a syntax that hides function pointers from the user. (Grimes 1997).

The terminology employed by COM and OLE is rather complicated. COM objects are different from C++ objects, even though one could use a C++ object to represent a COM object in code. COM objects are recognised by the system; in contrast, the system cannot be guaranteed to identify C++ objects that float around. The operating system does not provide the applications with any standard method of communicating what C++ objects are currently active. The term

‘object’ has one meaning for users and another for developers. By ‘object’, OLE users refer to the material that is embedded in another document. From the programmer’s point of view, an embedded ‘object’ stands for at least one COM object. The object can, however, consist of several COM objects, which in their turn almost always include more than just one C++ object. The term ‘method’ is similarly overloaded. To a C++ developer, it refers to a member function of a class. In COM, a ‘method’ is a specified function in an interface. This is not perfectly compatible with the C++ definition, since a given object may have many interfaces and each of these interfaces has many interchangeable methods. In COM, an interface is a collection of methods and an individual method is strictly a function. (Blaszczak 1997).

COM permits grouping together associated functions to form an interface which can be named and registered. A COM interface includes function prototypes and a protocol for their use. No specific implantation is defined . (Grimes 1997).

A COM class defines the implementation of one or more interfaces and acts as a template or recipe by which COM objects can be produced. COM objects are instantiated from COM classes and combine the implementation defined by that class with instance data specific to the object in question. This feature resembles the relationship between classes and objects in any object- programming language. COM is a language-independent binary standard. (Grimes 1997).

All COM objects must implement an interface called IUnknown, which provides lifetime management as well as the ability to query objects for functionality. A client can ask a COM object whether it supports a particular interface; if so, it will return an interface pointer to the client. (Grimes 1997).

COM was designed especially for protected memory systems and for passing data between applications located on different machines. As interface methods, COM uses the DCE RPC definitions of *[in]* and *[out]* parameters. Both of the definitions are one-way, but in *[in, out]* parameters, data is passed bidirectionally. All parameters, whether standard data types or buffer pointers, must be specified as one of these. (Grimes 1997).

The IUnknown interface has the following functions (Grimes 1997):

```
HRESULT QueryInterface (REFIID iid, void** ppvObject);
ULONG AddRef(void) ;
ULONG Release(void) ;
```

COM servers are executables that implement one or more COM objects; COM objects implement one or more interfaces. COM objects share functionality between various client applications. Somewhere, there is a code that can be used by several processes simultaneously. *Inproc* server stands for in-process. In other words, the object is executed in the same process space as the client, and hence the

server code must be implemented in DLL. In-process servers are fast and efficient. The drawback, however, is typical of DLL: the 'foreign' object is run in client's protected memory space, with the same privileges as any other client code. This requires that the object must be executed in a way that is compatible with the client. In particular, it must have the same threading model as the client. Similarly, since the object has complete access to the client's memory, an errant object can crash the entire process. The process is extremely dependent on the *inproc* server and sensitive about the how it calls it. (Grimes 1997).

Microsoft has implemented DCE RPC into its own operating systems. Microsoft RPC is compliant with DCE RPC. This means that although Microsoft RPC provides (almost) all the facilities of DCE RPC, its programs require some additional software to communicate with DCE RPC programs. Microsoft has its own agenda for handling security and naming services, which explains the requirement for additional software. With a few exceptions such as attributes like [*async*], Microsoft's IDL-compiler (i.e. MIDL) supports all functions of the DCE IDL compiler. (Grimes 1997).

Directory services is the mechanism by which a server can be found on the network. DCE uses the Cell Directory Service (CDS) for dividing the network into administrative units and for providing a hierarchical naming system. These units store security information concerning which users can log on, and include a central repository (i.e. the Directory Service) for the current servers to register themselves. NT administers computer groups through domains which are analogous to DCE cells. The domain has log-on facilities and can therefore define groups so that particular users can be given access to particular resources. DCE provides a naming service which, when a DCE client requests a server, enables locating the server machine and the running server. This is carried out via an intermediate program, called a CDS clerk, which resides on the local machine. When the client wants to find the server, it sends a request to the clerk which checks its local cache. If the clerk cannot find the name, it sends a query to the CDS server located on a networked machine. The CDS server uses the database of its named server, called a clearinghouse. If the CDS finds the server, it returns this information to the CDS clerk, which caches the result and passes it on to the client. A CDS cell can have more than one clearinghouse, and individual clearinghouses can hold information about locations of the other clearinghouses. The CDS clerk can utilise this information to extend its search further, if the server cannot be located at first attempt (Grimes 1997).

The same applies to NT. Microsoft RPC uses a service called 'the locator' for providing this functionality. The locator uses a dynamic database of running servers. When a request is made for a particular server, the locator checks its own database first. If the server cannot be found, the locator will then forward the request, via RPC, to a designated master locator. If the master locator fails to find the RPC server, it sends a mailslot broadcast to all the computers in the domain. If one of these computers finds the server, it replies to the master locator via a directed mailslot. All of the replies are collated, and the location of the server is

returned to the client locator. This is done by VIA, which passes the location back to the client. (Grimes 1997).

The COM architecture for object distribution is similar. When a client wants to connect to a server object, the server's name is stored in the system registry. In reference to distributed objects, the server can be implemented as an in-process .DLL, a local executable module, or as an executable module, or .DLL running remotely. A component called Service Control Manager (SCM) is responsible for locating and running the server . Making a call to an interface method in a remote object involves co-operation from several components. An interface proxy is a piece of interface-specific code that resides in the client's process space and prepares the interface parameters for transmittal. It packages or marshals the components so that they can be recreated and understood in the receiving process. The interface stub resides in the server's process space. It is a piece of interface-specific code and reverses the work of the proxy. The stub unpackages or unmarshals the transmitted parameters and forwards them to the server. It also packages reply information to be sent back to the client. The actual transmission of data across the network is handled by the RPC runtime library and a channel which is a part of the COM library. The channel works transparently with different channel types and supports both single and multi-threaded applications. (Microsoft 1995).

Since DCOM is identical to COM, COM objects or clients can be used with DCOM without recompilation (Grimes 1997).

Although DCE RPC does not provide support for objects, it can be used as an underlying mechanism for object distribution. With reference to Windows and Solaris, DCOM sits on Microsoft RPC. The Open Group is currently developing DCOM which runs on top of true DCE RPC for other platforms, particularly DCE Unix. This implies that DCE features, if not already available in DCOM, will appear at some time. The features will include DCE security (Kerberos) and DCE authentication. DCOM is effectively a version 2 of the Microsoft RPC, which can be detected even from the names of its structures. The main structure in DCOM packet is ORPC, short for Object RPC. (Grimes 1997).

2.4.3 CORBA with DCOM

The COM/CORBA Interworking Specification by OMG is a standard for interoperability between ActiveX and CORBA. The interworking specification contains two parts. The first part specifies OLE Automation-CORBA interoperability (i.e. ActiveX-CORBA). OLE Automation was designed for users wanting to manipulate objects in scripting languages such as Visual Basic. The second part specifies DCOM-CORBA interoperability. Automation-CORBA interoperability consists of a subset of DCOM interfaces.

For each part, the specification includes two levels of interoperability: mapping and interworking. One-way-interoperability is called a mapping solution, and a bi-directional (two-way) solution is called an interworking solution. By using a mapping solution, a CORBA object can be made available to ActiveX. However, Corba cannot have ActiveX objects (Expersoft 1998).

2.4.4 Real-time CORBA

Initial submissions have been made for RT CORBA (January 1998). In Table 3, a summary of the proposed extensions for RT CORBA is presented (Alcatel et al. 1998; Lockheed Martin 1998; Highlander Communications 1998; Objective Interface Systems 1998; Northern Telecom & Iona Tech. 1998):

Table 3. Summary of the proposed extensions for RT CORBA.

Feature	**Alcatel et al.**	**Lockheed Martin**	**Highlander et al.**	**Objective Interface Systems, Inc.**	**Northern Telecom, Iona Technologies**
Extensions	Interceptor Facility			Implementors should document typical and max. execution times of RT-CORBA services and PIDL operations	Explicit binding
Scheduled entities	threads, requests, replies, messages, transport end-points	process, threads, messages	threads	Threads	thread (tasks), provided by the OS, dispatch_priority
Scheduling services	SCOS: pluggable schedulers	Encapsulated scheduler (ES), fixed priority	Thread priority management (OS-RTPOA priorities)	Thread priority, pre-emptive scheduling	
Propagating priorities	Interceptor categories (client, server, POA, transport, thread, initialisation, message interceptors); priorities and their ceiling, QoS	Execution and communication priorities (client, communication and server), passed in GIOP	RT QoS, Realtime Common Object Service	RTQoS; GIOP and priority	

Feature	**Alcatel et al.**	**Lockheed Martin**	**Highlander et al**.	**Objective Interface Systems, Inc.**	**Northern Telecom, Iona Technologies**
APIs for	RT threads, Request queue, Transport Management, Buffer Management, Interceptor	Encapsulated scheduler, Resource management, Flexible communications	RT threads, Protocols, ORB, Current, RTPOA	RT threads, action, executive (abstract state machine), bind (QoS)	RT_POA
Resource control	APIs for end-to-end QoS	NVList, name and quantity, Get, Free, Release	Maximum resources defined		
Thread control	priorities, management, storage, pools	priority selection	Priorities, management	priority selection, sync and async thread control	thread pools (fixed, high water)
Transport management control	TMAPI: create, delete, get/set attributes, open/close connection. Transport attributes	Open/close connections, parameter passing, connection management, queues	ORB-API: access and specify protocol information		MsgQueues
Protocol control	ORB Flexibility Enablers; pluggable protocols (Object Reference Component and OCI Open Communication Interface)	Berkeley Socket protocol, selectable protocol mapping, service management	Set and get protocol parameters, add prior the creation of RTPOA, dynamic protocol selection	TCP and IIOP cannot be used to build predictable distributed systems	REO (Remote execution object) for a protocol-pair, get_protocol (REO_Name)
Memory control	Buffer API (not ready)		Constant definitions	InitMemory	Constant definition for buffers, stacks
ORB resource control	StrategyFactory Interface	Queue and locking techniques	ORB-API		RT_POA-API: threads, pools, dispatcher concurrence and queues

Feature	Alcatel et al.[1]	Lockheed Martin[2]	Highlander et al.[3]	Objective Interface Systems, Inc.	Northern Telecom, Iona Technologies[4]
POA	Simplified and extended POA	POA, Extensions not defined. Dispatch task within BOA or POA.	RTPOA, extended POA with priorities		RT_POA
Synchronisation	ORB and application data by mutexes, semaphores, multiple readers/single writer, condition variables; object factory	Timers, mutexes	Referred to Concurrence, Timing, Transaction, Event, Security and Messaging services	Ceiling locking, dynamic priorities, mutexes and semaphores in executive, sleep	
Asynchronous invocation	Query async request status by polling, one-way, upcall				Async Event notification
Other considerations	POSIX			POSIX, Emb-CORBA	POSIX

[1] Alcatel, Hewlett-Packard Company, Lucent Technologies, Inc. Object-Oriented Concepts, Inc. Sun Microsystems, Inc., Tri-Pacific. Collaboration with Deutsche Telekom AG, France Telecom, Humboldt-University, Mitre, Motorola, Inc., and Washington University. The specification is based on combined experience of prototypes and products from the submitters: ChorusORB r5, HP, Fixed Priority Scheduling, Pluggable Protocols, and TAO.

[2] Lockheed Marting Federal Systems, Inc. The specification is based on a real-time, fault tolerant CORBA for use in defense applications called HARDPACK.

[3] Highlander Communications, L.C., Visigenic Software, Inc. The specification is based on the experience in the CORBA domain and the real-time domain.

[4] The submission is based on the companies' experience of implementing and deploying CORBA in real-time systems. Used internally as a component of several product developments with Nortel's ORB.

The Message Transfer Interface in the Alcatel et al. submission is based on the following commonly used design patterns: Connector, Acceptor and Reactor. In addition, the specification includes some examples of how scheduling service is intended to be used.

2.4.5 Desktop Management Interface

The Desktop Management Interface (i.e. DMI 2.0), published by the Desktop Management Task Force (DMTF), defines a standard way for DMI agents to send management information across a network and ORBs. In addition, it defines IDL-specified APIs that can be invoked via the DCE RPC. DCE is an OSF protocol and provides security and naming across networks, as well as threads, distributed time, and a Distributed File System (DFS). DMI allows a component to define events within the MIF (Management Interface File). It defines also a standard interface for setting an event filter in the Service Layer. The event filter provides generic agent services, utilising the local MIF database.

The MIF facilitates managing, registering, installing, and uninstalling software applications on PCs, Macs, and workstations. The focus is on a computer system that specifies preinstallation file lists, software IDs and CRC signatures, version numbers, installation dependencies, and support information. With MIF, agents have a possibility to check hardware and software environments, install applications, inform a user where to get support, and finally, to uninstall redundant programs . (Orfali et al. 1996).

2.4.6 Distributed Management Framework

OSF's DME (Distributed Management Environment) has taken a new approach by building the Object Management Framework (OMF) on CORBA and OMG services. The OMF transcends the traditional manager/object relationship. An object may at any time assume one of two roles: that of a client requesting a service, or that of a service provider. The communication between objects may occur over any standard CORBA ORB that uses DCE for its core communications. The legacy systems are encapsulated by adapter objects. An application object can invoke operations on the adapter by using the typical remote invocations and proxy methods, which are invoked on the adapter call XMP (X/Open Management Protocol), in order to perform SNMP (IETF's Simple Network Management Protocol) or CMIP (OSI's Common Management Information Protocol) functions. The adapter object provides a higher level of abstraction for the legacy objects. Tivoli Management Environment is a commercial version of this architecture.

2.4.7 OCX and ActiveX

An OLE component is defined by a class that implements one or more interfaces and by a class factory (i.e. the interface that knows how to produce a component instance of that class.) An OLE component is not a predefined, self-contained unit, but rather a group of interfaces.

An OLE Control (OCX) is a custom control that contains a set of predefined interfaces. OCXs often have User Interfaces (UIs) which allow developers to set values for variables within the OCX. In addition, OCXs have methods which are callable routines within the OCX. An OCX is a well-defined, medium-grained component that is packaged like an OpenDoc part. However, unlike OLE containers, an OCX cannot embed any other parts. An OCX is a set of contracts between an OLE visual component and its container. It cannot function independently like an executable, as it is controlled by OLE containers such as Visual Basic, PowerBuilder, Delphi, Microsoft Excel, Microsoft Word and iXpress. OLE/COM provides all the facilities to locate, load, inquire, and remove OCXs. (Orfali et al. 1996).

An ActiveX is a minimalist OLE object for the Internet. OLE's component categories define the interface which an ActiveX supports. An ActiveX is a fine-grained component; OCX is a medium-grained component and OLE containers are large-grained, application-size components. (Expersoft 1998; Orfali et al. 1996).

ActiveX Controls are lightweight COM objects without the necessity to carry all the characteristics of OLE Controls. The mapping is one-way: an OLE Control is an ActiveX control, but an ActiveX Control is not necessarily an OLE Control. At one level, ActiveX is the active content, a way of making web pages that provide more than just text and pictures. (Grimes 1996).

ActiveX controls can run in a variety of containers, for example Visual Basic, Visual C++, Microsoft Access, Microsoft Internet Explorer, Delphi and Netscape Navigator. ActiveX controls feature two-way communication with their containers, and therefore the controls can interact with each other via scripting on the page. There are four ways to write an ActiveX control (Johns 1996.):

- Microsoft Foundation Classes (MFC)
- ActiveX Template Library
- BaseCtrl framework
- Visual J++ (COM objects only)

MFC controls are neither large, nor small. While developing controls that do not require the full functionality of an OLE control, it is useful to investigate what other options are available after prototyping in MFC. To run an ActiveX control written with MFC, the correct version of the MFC and C runtime DLLs must be

installed on the user system. The DLLs total at over one megabyte. Internet Explorer 3.0 ships with version 4.1 of the MFC DLLs. For hooking an ActiveX control to a web page, JavaScript or VBScript can be used. (Johns 1996.)

MFC controls are considerably simpler than MFC applications. The most simple MFC control has only the following three classes (Table 4) (Johns 1996):

Table 4. The main classes of MFC controls.

Description	*Class*	*Derived from*
Control module	ColeControlModule	CwinApp
Control window	ColeControl	CWnd
Property page	COlePropertyPage	Cdialog

Internet Explorer 3.0 implements two mechanisms to ensure that users can load and run controls without harming their systems. To begin with, it verifies the signature of any code it installs. Secondly, the browser checks if the control is marked safe for initialling and/or safe for scripting. Code signing requires only about 3,5K and as a result, it will not significantly affect the download time for web pages (Johns 1996).

Unlike Java applets, ActiveX controls can access the full power of the machine. There are two categories of safety: safe for scripting and safe for initializing. Safe for scripting means that regardless of what scripts do to manipulate a control, the user's machine will not be harmed. Safe for initializing means that regardless of what data gets passed from the web page to the control, the user machine will not be harmed. (Johns 1996).

Code signing is the process of adding data to the control in order for the Internet Explorer to verify:

- the identity of the author and
- that the control has not been modified after signing.

Signing a control requires an electronic certificate provided by a certificate authority (CA). The certificate and the private key must be carefully protected.

2.4.8 Java Beans and Java applets

JavaBeans is a platform-neutral component architecture for the Java application environment. It is used for developing or assembling network-aware solutions for

heterogeneous hardware and operating system environments, either within the enterprise or across the Internet (JavaBean 1998). The objective of the JavaBeans APIs is to define a software component model for Java. Within this model, third party ISVs are able to create and ship Java components to end users, who in their turn can merge the components into applications (JavaAPI 1998).

JavaBeans extends a so-called "Write Once, Run Anywhere" approach to reusable component development. JavaBeans are connected, via bridges, to other component models such as ActiveX and OpenDoc. As a result of this, software components that use JavaBeans APIs allow transporting to containers including Internet Explorer, Visual Basic, Microsoft Word, Lotus Notes, etc. (JavaBean1998; JavaEmb 1998).

JavaBeans is portable, and can run without modification on any platform with appropriate virtual machine layer. It functions also if embedded into either an ActiveX or OpenDoc component, although it is then limited by the platform which supports these components. In a similar manner as the ActiveX component, JavaBeans supports properties, events, and Bean state persistencies. Owing to its object nature, JavaBeans supports inheritance. The derivatives contain the same properties and event managers as their base class (JavaEmb 1998).

Some JavaBean components can be used as building blocks in composing applications. Some components will be used for more regular applications, which can then be merged together to form compound documents. These two aspects are overlapping (JavaAPI 1998).

Individual Java Beans will vary with respect to the functionality they support, but the following features are typical of a Java Bean (JavaAPI 1998):

- Support for “introspection” which enables a builder tool to analyse how a bean works.
- Support for “customisation”: while using an application builder, the user can customise the appearance and behaviour of the bean.
- Support for “events” in order to form connections between beans.
- Support for “properties”, both for customisation and for programmatic use.
- Support for persistence: a bean can be customised in an application builder and then have its customised state saved and reloaded.

A bean is not required to inherit from any particular base class or interface. Visible beans must inherit from java.awt.Component if they are intended to be added to visual containers. Invisible beans are not required this (JavaAPI 1998).

Beans are appropriate for software components that can be visually manipulated and customised to achieve an effect. Class libraries are an appropriate way of providing functionality which is useful to programmers, but which does not benefit from visual manipulation (JavaAPI 1998).

The three most important features of a Java Bean are the set of properties it exposes, the set of methods it allows other components to call, and the set of events it fires. Basically, properties are named attributes associated with a bean. They can be read or written by applying the appropriate methods on the bean. The methods that a Java Bean exports are normal Java methods which can be called from other components or from a scripting environment. Events provide a way for one component to communicate to other components that something interesting has happened (JavaAPI 1998).

Different platforms will vary in their ability to support the full JavaBean API. However, when a platform is unable to provide the full functionality, it must provide a reasonable, harmless default instead. This means that JavaBeans component writers can program to a consistent set of APIs and trust them to function everywhere (JavaAPI 1998).

The three primary network access mechanisms that will be available to Java Beans developers on all Java platforms are (JavaAPI 1998):

- Java RMI. The distributed system interfaces can be designed in Java and clients and servers can be implemented against those interfaces. Java RMI calls will be automatically and transparently delivered from the client to the server.

- Java IDL. The Java IDL system implements the standardised OMG CORBA distributed object model. All system interfaces are defined in CORBA IDL interface definition language. Java stubs can be generated from these IDL interfaces, allowing Java Bean clients to call IDL servers, and vice versa.

- JDBC. The Java database API, JDBC, allows Java Bean components to access SQL databases. These databases can either be on the same machine as the client, or on remote database servers. Individual Java Beans can be written to provide tailored access to particular database tables.

Java Beans are subject to the standard Java security model. The basic run-time model for Java Bean components requires that they run within the same address space as their container (JavaAPI 1998).

A Java applet is a Java program that can be included in an HTML page. In this respect, its behaviour is quite similar to that of an image. When a Java-compatible browser is used to view a page containing a Java applet, the applet's code is transferred to the receiving system and executed by the browser (JavaApp 1998).

2.4.9 OPC

OPC (OLE for Process Control) was developed in order to satisfy the demand for integrating plant floor data into business systems. Plant floor devices and data have to be easily accessible instead of forming standalone “islands” without methods for distribution. What is required is a common way for applications to access data from any device on the plant floor, thereby creating a seamless data access in a manufacturing environment (OPC Taskforce 1996).

OLE for Process Control keeps hardware providers separate from software developers. It assigns data collection and distribution to one single developer. The developer provides software components for those devices which provide data to clients in a standard manner. Developers can write the software components in C and C++, and these components can be used by business application developers for example in Visual Basic without concern for the actual data access (OPC Taskforce 1996).

OPC is based on Microsoft’s OLE/COM technology. The specification describes the OPC COM Objects and their interfaces which are implemented by OPC Servers. OPC Servers are provided by different vendors. The code written by the vendor determines the devices and data to which each server has access, the way in which data items are named and the details about how the server physically accesses that data (OPC Taskforce 1996).

Within each server, the client can define one or more OPC Groups. This provides a way for the clients to organise the data in which they are interested. Within each group, the client can similarly define one or more OPC Items which represent connections to data sources within the server (OPC Taskforce 1996).

OPC interfaces can be used at both the lowest and the highest level. On the lowest level, raw data from the physical devices can be transferred into a SCADA or DCS. On a higher level, data from the SCADA or DCS can be added into the application. OPC is a specification for two sets of interfaces: the OPC Custom interfaces and the OPC Automation interfaces. OPC Automation interfaces are generally used by programs created by some form of scripting language. OPC Custom interfaces are generally used in programs made by C++ to attain maximum performance (OPC Taskforce 1996).

OPC Server will generally be a local or a remote EXE including code that is responsible for data collection from a physical device (OPC Taskforce 1996).

2.5 Web client/server

Hypertext Web came into being in 1993, when Mosaic's graphical Web browser introduced the first client/server application environment on the top of the Internet. Extended inter-action between Web with HTML forms and the CGI (Common Gateway Interface) server protocol began two years later. New and more secure protocols such as SSL (Secure Socket Layer), Secure HTTP (S-HTTP), and firewalls were also introduced. Java is the first step towards creating a client/server Object Web. The next step requires Java equipped with a distributed object infrastructure: CORBA ORBlets, OLE COMlets, and compound document frameworks such as OLE and OpenDoc. (Orfali et al. 1996).

Portability, platform-independence, and content-independence are based on four technologies at the top of the Internet infrastructure:

- graphical Web browsers,
- HTTP RPC,
- HTML-tagged documents, and
- URL global naming convention.

The URL consists of a protocol scheme, server address, port number and target resource. The protocol scheme tells the Web browser which Internet protocol to use. In addition to HTTP (Web's native protocol, web pages and server programs), URL supports Gopher (precursor to the Web, hierarchical menus), FTP (files), News (newsgroup or article), Mailto (mail to a designated e-mail address), and WAIS (domain name of a target database and list of search criteria) (Orfali et al. 1996).

HTML is rooted in the ISO SGML standard. An HTML document is an ordinary text file whose appearance is controlled by embedded tags. Tags are non-case-sensitive commands surrounded by angle brackets. A tag pair contains a command. The HTML protocol standards specifies the features of HTML documents (HTML 1.0, 2.0 and 3.0). World Wide Web Consortium (W3C) issues piecemeal standards for new features such as tables, applets, frames, and active objects.

The Hypertext Transfer Protocol (HTTP) introduces self-describing messages by using a variant of the Internet Mail's MIME (Multipurpose Internet Mail Extensions) protocol, which provides extensible mechanisms for transmitting multimedia e-mail. MIME supports seven types of data: plain text, audio, video, still images, messages, and application-specific data. HTTP /1.1 consists of several extensions to HTTP 1.0 (Orfali et al. 1996).

In 3-tier client/server Web-style, the server passes a method request and its parameters to the back-end programs by using a CGI (Common Gateway Interface) protocol. The back-end program (e.g. DBMS, Lotus Notes or TP

Monitor) executes the request and returns the results in HTML (2.0 or 3.0) format to the Web server by using CGI protocol. The Web server acts as a conduit between Web client and back-end programs. CGI technology enables Internet clients to update databases on back-end servers. Furthermore, the databases can be used in electronic commerce. (Orfali et al. 1996).

A form consists of three components: the HTML document as the input form, the server CGI application that acts on the data from the input, and the reply document a user sees after submitting the form. The HTML for the reply document is dynamically generated by the CGI program. The action attribute of the form describes the URL to which the form's content is sent. The URL must be the name of a CGI program or a script. CGI-bin is a special executable directory where CGI programs reside. The CGI-bin is usually located under the web master's direct control. Figure 5 represents an end-to-end client/server scenario using *HTTP-POST* method and a CGI program. (Orfali et al. 1996).

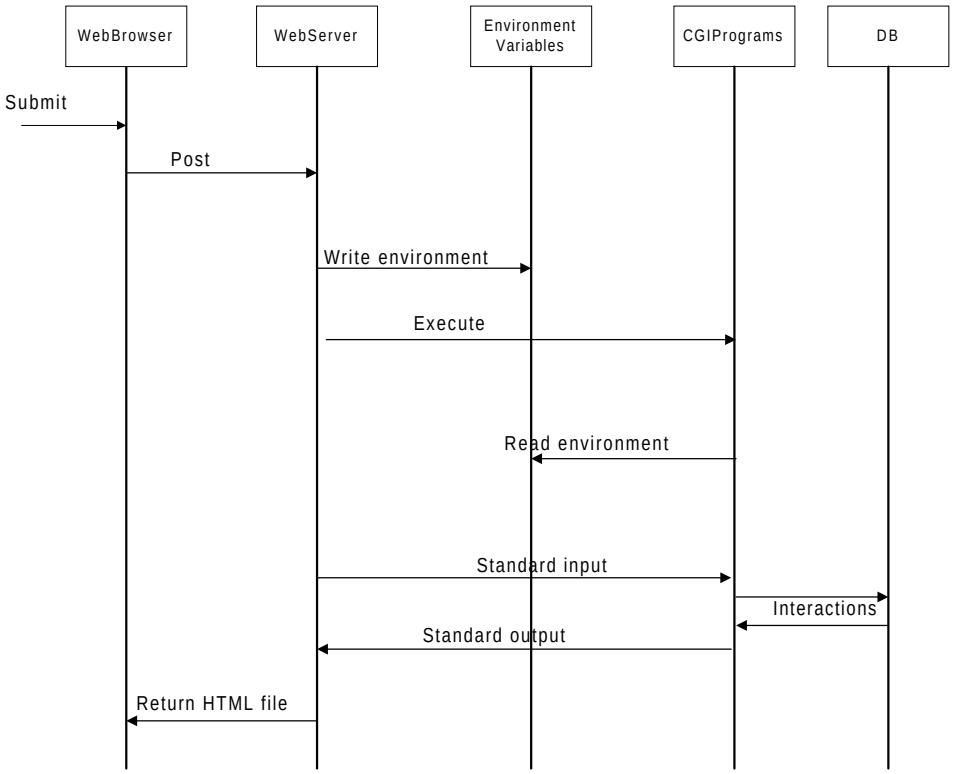

Figure 5. An end-to-end client/server scenario with CGI.

Windows CGI 1.1 specification is a Windows-friendly version of the CGI protocol. The HTTP server uses a WinExec call to launch the CGI program. This protocol uses a set of files to exchange information instead of environment variables or standard input/output. (Orfali et al. 1996).

2.5.1 Security

Secure Sockets Layer (SSL) is a secured socket connection that provides a security layer between the TCP/IP transport and sockets, without involving the applications. In the Netscape implementation, the HTTP server that implements SSL must run on socket address 443 as opposed to the standard 80.

The SSL protocol provides

- private client/server interactions which use encryption,
- server authentication, and
- reliable client/server exchanges via message integrity checks that detect tampering.

S-HTTP is a security-enhanced variant of HTTP. S-HTTP adds application-level encryption and security on top of ordinary sockets-based communications. Clients and servers use a MIME-like protocol to encrypt the contents of the messages. S-HTTP provides the following security features (Orfali et al. 1996):

- It authenticates both clients and servers.
- It checks for server certificate revocations.
- It supports certificate chaining and certificate hierarchies.
- It supports digital signatures.
- It allows an application to negotiate the security levels it needs.
- It provides secured communications through existing corporate firewalls.

S-HTTP marks individual documents as private or signed. In contrast, SSL ensures that the channel of communication between two parties is private and authenticated. SSL and S-HTTP can coexist if S-HTTP is layered on top of SSL. Terisa Systems offers a toolkit that supports both S-HTTP and SSL. (Orfali et al. 1996).

Microsoft has published its own security specification called Private Communications Technology (PCT), but is also planning to support both SSL and PCT in its forthcoming Internet products. (Orfali et al. 1996).

A firewall is a gatekeeper between the Internet and the private network. There are two types of firewalls: packet-filtering routers and proxy-based application gateways. A packet filter mechanism provides a basic level of network security at the IP level and is implemented in routers. Packet filters drop, reject or accept packets in accordance with the rules they have been programmed to follow.

Another guideline contributing to this process is the IP header of the current packet (Orfali et al. 1996).

Proxy firewalls, i.e. application firewalls, are the most secure form of firewall. All informing Internet traffic is funnelled to the appropriate proxy gateway for mail, HTTP, FTP and etc. The proxy makes the decisions based on context, authorisation, and authentication rules. It operates at the highest level of the protocol stack and functions as a relay between the Internet and the private network. Transparent proxies are completely transparent to end-users. Additional security is provided by combining router and proxy techniques. (Orfali et al. 1996).

2.5.2 Electronic payments

Most banks have created their electronic payment infrastructure on top of SSL and S-HTTP. CyberCash uses its own secure channel with the bank and uses neither SSL nor S-HTTP, but instead realises on its own encryption. VirtualPIN can similarly be used as an alias for a personal credit card, since the application itself includes a credit card number and an e-mail address. Every purchase made by VirtualPIN is confirmed by First Virtual, the supplier of the VirtualPIN application.

Digicash provides e-cash, the Internet equivalent of cold hard cash. E-cash bills can be acquired from a bank account and are used for anonymous purchases. E-cash cannot be used for credit purchases.

EDI-MIME specification will enable exchanging invoices over the Internet. It defines a common method for sending EDI documents on the Net. In addition to SSL and S-HTTP, there are two standards-based security options: the MIME Object Security Services (Moss), known also as Privacy Enhanced Mail, and Pretty Good Privacy (PGP). Both offer encryption, authentication, and privacy features.

Visa and Mastercard are working on a common solution called Secure Electronic Transaction (SET). Joint Electronic Payments Initiative (JEPI), as announced by W3C and CommerceNet, seeks to achieve compatibility among the different electronic payment schemes - including e-cash, e-checks and electronic credit cards. (Orfali et al. 1996).

2.5.3 Java objects in Web

HotJava is a special Web browser that can interpret Java-generated code. Netscape and Spyglass have adopted some features of HotJava.

The following scenario portrays a Web client/server interaction including a Java applet (Orfali et al. 1996):

1. A Web browser requests the applet as it encounters the new HTML <APPLET> tag.

2. The browser initiates a separate TCP/IP session to download each applet it encounters within a Web page.

3. The browser loads the applet into client memory and executes it.

4. The browser deletes the applet from memory when it exits the Web page.

In addition to exchanging traditional content (text, graphics, audio, and video), Java enables Web applications to exchange mobile code. Other examples of mobile code systems are Safe-Tcl, Colusa's Omniware and General Magic's Telescript. Microsoft provides a portable version of its Visual Basic Script (VB Script) engine. Mobile code systems are the founding technology for mobile agents. A mobile code system is expected to provide the following services (Orfali et al. 1996):

- *Safety.* The system must be able to control the environment of an applet, i.e. its access to memory, system calls, and server function calls.

- *Portability.* The system must provide cross-platform memory management, threads, synchronisation, communications, and GUI services including a compound document framework.

- *Life-cycle.* The system must provide a run-time environment for loading, unloading, and executing the code.

- *Distribution.* The system must provide facilities for moving applets across the network. In addition, it must prevent mobile code from exposure to viruses.

Java is only a partially compiled language, since 80% of the compilation work involves creating the bytecode. The remaining 20% is interpreted by the Java run time. The client machine includes a Java verifier which runs the tests for the bytecodes. After testing, the loader transfers the bytecodes to an interpreter. Just-in-time compilers are designed to improve Java performance.

2.5.3.1 Protection mechanisms

Memory layout decisions are made during run time. The Java compilation codes access memory via symbolic names. These are resolved to real memory addresses at run time by the Java interpreter (late-binding). The verifier protects the Java run-time environment from a range of external threats. The loader divides the set of Java classes into separate name locations: one for classes from the local file system, and others for each network source (separation). Access control lists (ACLs) are used for additional levels of security on top of language and run-time

base. Either imported code, or code which is invoked by imported code, controls read and write access to files. With HotJava browsers, security mode is used to set security levels: *none* - applets cannot access the network; *applet host* allows applets to access data on the home server from which they originate. *Firewall* mode allows applets to access resources that are located outside the firewall. *Unrestricted* mode allows applets to connect any host on the Internet. (Orfali et al. 1996).

Web browser controls the life cycle of an applet by invoking *Init* after loading and by invoking *Start* to begin preparation for run. *Paint* asks the applet to display the contents. *Stop* and *Destroy* are used to instruct the applet to terminate all its threads, stopping the execution, and releasing resources. The APPLET tag contains the required information (code, width, height, codebase, align) that connects the Web browser to the embedded applet. (Orfali et al. 1996).

Active objects can also be inserted inside an HTML document by using Netscape's EMBED tag for compound document embedding and Microsoft's DYNSRC attribute for video and audio. HTML 2.0 provides IMG tag for inserting media into HTML documents. INSERT tag (from W3C) is an universal tag to be used for inserting different kinds of objects such as HTML images, Java applets, OLE components, OpenDoc parts, media handlers, in addition with a wide rage of plug-ins. (Orfali et al. 1996).

Java bytecode engines are available as PicoJavas, MicroJavas and UltraJavas. The Java "operating system" consists of Java language, Java libraries, JDBC and ORBs. Java Beans, plug-ins, applets and Java IDL are provided for desktop Java applications. Java ORBlets are Java component services provided for CORBA infrastructure. Sun is developing Java Object Environment (JOE) which is a portable CORBA ORB written entirely in Java. JOE allows any Java applet to access any CORBA services. Java ORBs are available also from some ORB vendors (Iona, PostModern, Expersoft, Visigenic). PostModern's BlackWindow was the first Java ORBlet on the market. It implements the client and server sides of a CORBA 2.0 ORB in less than 100 Kbytes of Java bytecode. (Orfali et al. 1996).

2.5.4 Embedded WebBrowsers

The operating system for embedded web browser should support Unix or POSIX API. A microkernel architecture can be used to provide a POSIX API (for example, QNX/Neutrino basic POSIX OS services in about 32k of code). Java runtime-engine favours an essentially POSIX-compliant OS with asynchronous I/O, generic thread support, file system, networking support, and windowing system (Hildebrandt 1997).

At present, the following embedded Web browsers are available (www-links are included in the references):

1. Spyglass Device Mosaic 2.0
    - Designed to run on VxWorks, pSOS, QNX, LynxOS, OS-9, and PowerTV
    - 633kb code space
    - Under 2Mb of memory
    - Modular design -> scalable
    - A minimal amount of platform dependent code -> portable
    - Can display HTML documents encoded with any character set
    - Fully HTML 3.2 compliant
    - HTTP 1.0 compliant
    - GIF, animated GIF and JPEG support
    - Basic authentication, cookies and server push support
2. HotJava by SUN (JavaProd. 1998)
    - 2 versions: The HotJava Browser is a full featured, lightweight Web browser with a highly-customisable user interface. The HotJava HTML component is a JavaBeans component for displaying HTML. It can be embedded into other applications or coupled with a custom user interface
    - Written completely in Java
    - Runs on any JDK 1.1.4-compliant Virtual Machine
3. ICE Browser (Icesoft 1998).
    - ICE Browser is not compatible with Java version 1.02. Providing backward compatibility is virtually impossible, since versions prior to 1.1 cannot interpret Java Beans.
    - HTML 3.2 compliant
    - NLS Internationalization
    - Tables
    - Frames

- HTTP Authentication
- Applets
    - Applet communication between web pages by using ICE Connect
    - Variable repository
    - Passing objects to and from applets
    - Persistence of variable repository between sessions
- Forms
- Forgiving HTML Parser
- Printer support
- Java archive (JAR) format support
    - JAR archives for use in <APPLET> tag
    - JAR files as contents resources. Accessing a JAR file URL retrieves all resources, in a similar manner as with web sites.
- Superb scrolling performance
- Client side image maps
- Multithreaded for asynchronous loading and document parsing.
- Asynchronous image and applet loading
- Persistence - serialises current state
- Background images
- Document history
- Lightweight (only 120kb)
- Fast rendering.

2.5.5 Embedded WebServers

An embedded web server must function as a process that is subordinate to the main purpose of the device. It is not necessarily required to support a very large number of connected users, the usual number being from one to a few dozen. Embedded web servers work with a fixed set of content which is usually “frozen”

at the time that the equipment is manufactured. The servers are required to run on a broad range of equipment (Wingard 1997).

Embedded web servers use security primarily as a means to limit access to sensitive information or configuration controls. A relatively straightforward way to enforce access control to sensitive information is to implement access control lists which are based on the clients' IP addresses or DNS host names. Another way to manage security is using digest authentication. In digest authentication, the client transmits only a "digest", which is a combination between user name, password, URL named in the request method and "nonce" value supplied by the server in the original challenge. The server forms its own version of the digest from the same information and compares it to the offered digest (Wingard 1997).

The architecture must minimise the requirements for such system resources as threads, since the Web server acts as a "secondary" feature of the system and should not interfere with the primary purpose. Support for multiple connections can be accomplished in a single processor/thread by implementing a finite state machine, which processes an HTTP request as a sequence of discrete steps. The FSM is run by a small scheduling system that uses lightweight task structures. With this system, several connections can remain active at once. Each of these connections is represented by a specified task in the scheduler (Wingard 1997).

One of primary tasks of the server is to perform mapping, which involves transferring the server's virtual name space into a physical name space (Wingard 1997).

The most common application for embedded Web server is providing an interface for status-reporting, configuration, and control features of devices. When a request arrives, the embedded server determines the status of the device and transforms that information into an HTML page. After that, the information is returned to the browser. One method to generate this HTML content is using server APIs, which are conceptually similar to the APIs provided by mainstream Web servers (Wingard 1997).

The flexibility of a server API enables applications to perform processing at different stages. If the Web application is required to process large amounts of data, this is an important advantage (Wingard 1997).

An embedded Web server should be equipped with a capacity to implement standard functions of HTTP protocol. In addition, the server must be autonomous from the type of the in-system communication network. Embedded Web server should be a scalable software system and capable of satisfying a wide range of time constraints. It can be built as a multiprocess system, which can run on one processor in multiprogram mode, as well as on parallel multiprocessor or on a distributed system (Sheinin et. al., 1998).

A distributed real-time operating system is required in supporting real-time mechanisms for an embedded Web server. In addition, it should support

distributed parallel data processing. The network interface for communications should be network-independent. Network protocols in real-time systems should be protocols with small overheads, and should have small time latency (Sheinin et. al., 1998).

On the following pages, Commercial Web servers are described (www-links in the references):

1. Spyglass MicroServer 2.0 (Spyglass 1998).

    - Delivered as Windows NT or Solaris source code. Designed to run on LynxOS, QNX, OS-9, VxWorks, and pSOS
    - HTTP 1.1 compliant
    - Minimum configuration 10kb, full-featured 36kb
    - Support for multiple simultaneous users
    - Optional security, including basic and digest authentication
    - Dynamic content generation and HTML translation utilities

2. Lava by Magma (Magma 1998).

3. EmWeb by Agranat Systems (Agranet 1998).

    - Supports Solaris, SunOS, HPUX, Linux and Windows NT+95

4. Nuceleus WebServ by Accelerated Technology (Nucleus 1998).

    - A general purpose HTTP server that can process multiple requests simultaneously by interleaving buffers back to the web browser. This function permits transferring data to the web browser. It does not, however, require memory to store the entire HTML file during transfer. Nucleus WebServ provides also the ability to upload files from the web browser.
    - HTTP 1.0 Compatible
    - GET Method
    - HEAD Method
    - POST Method
    - Extremely Small Size
    - Fully Functional Server in Small Package
    - Supports Multiple Concurrent Requests

- File upload (on-line document update)
- Document Compression
- Optional Encrypted Authentication
- Hard Encryption (DES)
- Java Applet Interface
- Dynamic Web Page Content
- CGI (plug-in) Support
- Forms support (POST Method)
- Server Side Include support
- NSAPI Like Plugin Interface
- Flexible Page Storage
- Pages in Memory or on Disk

5. RomPager by Allegro Software Development Corporation (Allegro 1998).

   - The RomPager embedded web server functions with any OS or TCP package and has been ported to a number of environments including the ATI Nucleus, Epilogue, Lynx, Microtec VRTX, Pacific Softworks Fusion, PSOS, QNX, SNMP Research and VxWorks stacks. RomPager has also been ported to Windows, Macintosh and Unix development environments. Since RomPager comes with its own internal scheduler, any OS can be used to support the engine.

6. WebControl by Rapid Logic (Rapid 1998).

   - WebControl is specifically designed to be RTOS and TCP stack independent. It is compatible with a system which:

     - has at least a rudimentary memory management system. If the system can provide WebControl with initial memory block, WebControl's own memory management system function after this.

     - has at least a primitive TCP stack. WebControl requires only a minimalist interface to a TCP port

7. Quiotix Embedded WebServer by Quiotix (Quiotix 1998).

   - One basic requirement of Quiotix Embedded WebServer is a TCP/IP protocol stack. It requires few other services from the embedded environment, which allows it to work even in platforms without a real operating system. (It will run well with an operating system as well.) The

standard QEWS distribution includes system-dependent modules for Microsoft Windows, Sun Solaris, DEC Unix, and popular embedded operating systems such as VxWorks, pSOS, and LynxOS. Source code and complete documentation for porting to other environments are also included.

- Full HTTP support
- Small, fast, and easy to integrate with environment
- Automatically generates a minimal server subset for each application. Eliminates the need for multiple server products or "one size fits all" compromises
- InternalCGI 1.1
- SSI
- Basic and Digest authentication, ACLs
- Automatic forms processing
- Image map support
- Virtual file system
- Host-based environment for content development

2.6 Communication mechanisms

This chapter describes the communication mechanisms used in different types of middleware. The type of communication mechanisms required depends on the application. Real-time applications use mostly asynchronous communication, whereas embedded systems require synchronous communications as well.

2.6.1 Synchronous communication

In synchronous communication, the sender specifies the receiver and the message is transmitted without delay. Normally, the sender waits for the reply and the transmission is completed after a reply is received. CORBA and DCOM clients use synchronous invocations in passing parameters to servers and stand-by for the result information.

CORBA SII (Static Invocation Interface) enables an invocation to be declared as a one-way call, which simulates asynchronous message passing. The one-way

invocations are processed as best effort, but the transmission to the end-point cannot be guaranteed. The deferred-synchronous mode applied by DII (Dynamic Invocation Interface) allows the sender to continue processing while the invocations are being processed. At-most-once execution is the most reliable characteristic.

DCOM proxy and stub mechanism is very similar to the method by which CORBA implements local/remote transparency. This method applies static stubs on the client side and interface skeletons on the server side. DCOM requires that every OLE component implements the Iunknown interface, which is used by clients to determine at run time which interfaces a component provides, and the connections between them. This corresponds the CORBA DII mechanism.

2.6.2 Asynchronous communication

2.6.2.1 Events

Event-based communication is asynchronous communication without a queue at the receiving end of the event. If the receiver is not available, the communication fails, i.e. the event is lost. The object that supplies the event is called the supplier and the receivers are called consumers. The event channel object decouples communication between suppliers and consumers.

Event service in CORBA supports a push model and a pull model at both ends of the channel. A push supplier sends an event to the channel by invoking a push operation, which the channel must accept. The pull supplier supports a pull server interface. The channel functions as a client and requests an event from the supplier, for example at timed intervals. A push consumer supports the push server interface invoked by the channel client. Simultaneously, a pull consumer invokes the pull operation on the channel. The pull operation responds to the event if another event is on queue on the object channel. (Figure 6). The channel maintains the event information in its queues until all the pull mode consumers have received the event. (Siegel 1996).

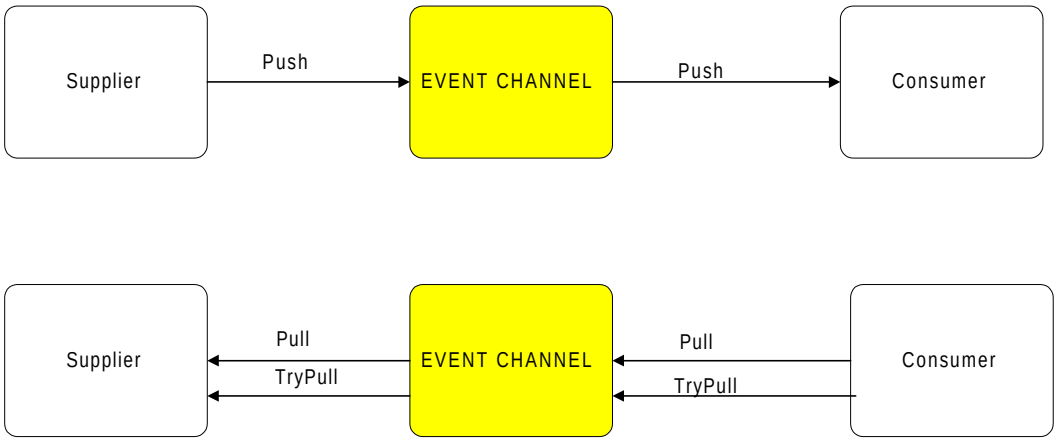

Figure 6. Event-based communication with push and pull models.

CORBA Event service which co-operates with the Relationship service can be used to implement publishing and subscribing capabilities. An event can implement a dynamic linkage, in which an object is connected to a hot-link update of information. Events can be used in a windowing environment to pass input events synchronously to applications. In real-time systems, the notifications are real-time. By using events, it is possible to decouple event suppliers and consumers; another advantage is that events provide looser coupling among objects. (Mowbray 1997).

2.6.2.2 Messaging

Messaging is asynchronous communication in which the sender does not know the identity of the receiver but only the message queue in which it resides. Delivery to the queue is guaranteed (minimum quality-of-service). CORBAs Messaging Services are not yet defined, but two revised joint submissions have been sent to the next meeting (deadline 31 March 1998).

Message-based communication is used when large quantities of data must be asynchronously transferred between the sender and receiver objects. This communication model is normally used in mainframe and real-time systems. Message-oriented middleware is based on this communication model. The main benefit of this model is that the sender and receiver are loosely coupled and cannot be simultaneously available. Messages are buffered in queues and delivered when the receiver is available. Synchronisation at the application level guarantees reliable communication..

2.6.3 Communication mechanisms

2.6.3.1 Remote procedure call

RPC is the basic communication mechanism in CORBA, although it is widely used in non-CORBA applications too. RPC itself has a close resemblance to the

session layer of the OSI model. It contains also aspects of its presentation layer, since it converts user data into a platform independent format that is suitable for transmission over the network. RPC is supported as a standard library in the following environments: Sun ONC RPC, IBM AIX, OSF DCE RPC, Microsoft RPC (which is DCE RPC compliant), and Novell's TI-RPC. CORBA is based on ONC (Open Network Computing), whereas DCOM uses its own RPC which is based on DCE RPC. The main difference between ONC and DCE is that ONC uses the object-oriented model whereas DCE is a procedural approach capable of providing naming and security services. (Sydorowicz 1997).

The communication is based on interface descriptions which are generated by an interface generator. The interface generator, RPCgen, acts as follows: IDL -> RPCgen -> server stubs, client stubs, header files, XDR data type conversion file. RPC utilises generated XDR functions for encoding, decoding, and destroying data. Data conversion forms a machine-specific definition to XDR representation, which is called serialisation or marshalling. (Sydorowicz 1997).

TI-RPC is a version of RPC. It interfaces with the transport layer by using the Transport Layer Interface instead of sockets. TI enables having a consistent programming interface across different machines and transport protocols. Earlier versions of RPC supported TCP and UDP. TI-RPC enables specifying a transport at any time, including prior to run-time. The strong point of RPC is its tolerance for transport alterations. The transport can be modified to suit a particular hardware choice without affecting the applications. (Sydorowicz 1997).

RPC utilises XDR in converting complex data structures into transferable form. Session layer functionality is provided by RPC library. (Sydorowicz 1997).

RPC Specification (Open Group 1997) provides specifications for portability and interoperability. Portability consists of API, stubs, and interface definition language. Interoperability is defined by the protocol and service specifications. The protocol specifications describe how RPC clients and servers communicate and the service specifications describe a set of abstract services which must be implemented by RPC run-time systems.

RPC mechanism maps the local procedure call paradigm onto an environment where the calling procedure and the called procedure are distributed between different execution contexts. Usually, these contexts reside on physically separate computers linked by communications networks.

A procedure is defined as a closed sequence of instructions that is entered from, and returns control to, an external source. Data values may be passed in both directions along with the flow of control. A procedure call is the invocation of a procedure. A local procedure call and an RPC behave similarly. There are,

however, semantic differences due to several following properties of RPCs (Open Group 1997):

- **Server/client relationship (binding).** While a local procedure call depends on a static relationship between the calling and the called procedure, the RPC paradigm requires more dynamic behaviour. As with a local procedure call, the RPC establishes this relationship through binding between the calling procedure (client) and the called procedure (server). However, in case of RPC the binding usually depends on a communications link between the client and server RPC run-time systems. A client establishes a binding over a specific protocol sequence to a specific host system and endpoint.

- **No assumption of shared memory.** Unlike a local procedure call (which commonly uses the call-by-reference passing mechanism for input/output parameters), RPCs with input/output parameters have copy-in, copy-out semantics, due to differing address spaces for calling and called procedures.

- **Independent failure.** Besides execution errors arising from the procedure call itself, RPC introduces additional failure cases which originate from execution on physically separate machines. Remoteness introduces issues such as remote system crash, communications links, naming and binding issues, security problems, and protocol incompatibilities.

- **Security.** Executing procedure calls across physical machine boundaries has additional security implications. Client and server must establish a security context based on the underlying security protocols, and they require additional attributes in order to authorise access.

The basic operations performed by a Remote Procedure Call (RPC) are:

- The **Invoke** service primitive is used to invoke an RPC.

- The **Result** service primitive is used to return the output and input/output parameters at the end of a normal execution of the invoked RPC.

- The **Cancel** service primitive is used to cancel an outstanding RPC. This operation forwards a client cancel request to the server application thread. If the server application thread does not return within a caller-specified time, the RPC fails. In most cases, this type of processing is specific to the associated protocol machines.

- The **Error** service primitive can be used by the server manager routine to indicate an error in the response to a previous Invoke indication.

- The **Reject** service primitive indicates that there is a problem with the underlying communications or the RPC protocol machines. The reject reason (parameter, Call_Reject_Reason) can indicate the state of a particular RPC and therefore can determine whether the call has already been executed at the server.

The specification describes communication protocols that are RPC connection-oriented or RPC connectionless. It includes also an abstract specification of the underlying transport services required by the RPC protocols. It includes the format specifications for those RPC Protocol Data Units (PDUs) which are used by the protocols and common authentication verifier encodings. The Network Data Representation (NDR) specifies a set of NDR data types and byte stream formats in which clients and server are transmitted. IDL data types are mapped to NDR data types. A RPC stub specification defines the stub characteristics required for interoperation. In addition, the specification defines what information is stored in and retrieved from name services.

Table 5. UUID Format.

Field	**NDR Data Type**	**Octet #**	**Note**
time_low	Unsigned long	0-3	The low field of the timestamp.
time_mid	Unsigned short	4-5	The middle field of the timestamp
time_hi_and_version	Unsigned short	6-7	The high field of the timestamp multiplexed with the version number.
clock_seq_hi_and_ reserved	Unsigned small	8	The high field of the clock sequence multiplexed with the variant.
clock_seq_low	Unsigned small	9	The low field of the clock sequence.
node	Character	10-15	The spatially unique node identifier.

The specification defines a UUID (Universal Unique Identifier) format. A UUID is an identifier that is unique across both space and time, with respect to the space of all UUIDs. A UUID can be used for multiple purposes, from tagging objects with an extremely short lifetime, to reliable identification of persistent objects across a network.

The generation of UUIDs does not require registration authority for each single identifier. Instead, what is required is a unique value over space for each UUID generator. This spatially unique value is specified as an IEEE 802 address which is usually already applied to network-connected systems. Selecting this 48-bit address can be done by obtaining an address block through the IEEE registration authority. This UUID specification requires that an IEEE 802 address is available (Table 5).

IDL type includes definitions for a variety of data types. The endpoint mapper protocol defines how the endpoint mapper listens on a well-known endpoint for each supported protocol. Registered endpoints are listed in Endpoint Mapper Well-known Ports. Conversation manager protocol and the remote management interface have also been described. (Open Group 1997).

The portability specification describes the behaviour that is common to all implementation, providing the concrete syntax and semantics of the Application Programmer's Interface (API) to RPC. The specification consists of the RPC programming model, data types used in the RPC API, the RPC run-time library routines, IDL and its mapping to ISO C data types, and the stub characteristics required for portability. (Open Group 1997).

2.6.3.2 Remote method invocation

In order to match the semantics of object invocation in distributed systems, RMI (i.e. Remote Method Invocation) is required. In systems such as these, a local surrogate (stub) object manages the invocation on a remote object (RMI 1998).

RMI is a function which invokes a remote interface method on a remote object. A method invocation on a remote object has the same syntax as a method invocation on a local object (RMI 1998).

Java RMI system is specifically designed to operate in Java environment. Other RMI systems can be adapted to manage Java objects, but these systems cannot be seamlessly integrated, since they require interoperability with other languages (RMI 1998).

The RMI system uses Java interfaces and a special "stub" compiler to provide transparent access to remote objects. To begin with, an interface is defined by specifying the methods which a remote object provides. Next, a server class is defined to implement the interface. The stub compiler is invoked to generate classes which act as connectors between a local representation of an object and an object residing on the server. The RMI system also provides a naming service that permits servers to bind object references to URLs (Morrison et al. 1997).

Goals (RMI 1998):

- Seamless support for remote invocation on objects in different virtual machines.

- Call-back support from server to Applet..
- Natural integration between distributed object model and Java language.
- Overt difference between the distributed object model and the local Java object model.
- Rendering writing reliable distributed applications as simple as possible.

RMI system should permit extensions such as garbage collection of remote objects, server replication, as well as activating persistent objects to service an invocation. These extensions should be transparent to the client, and should not increase implementation requirements for the servers that use them. To support these extensions, the system should also support (RMI 1998):

- Several invocation mechanisms.
- Various reference semantics for remote objects.
- Security managers and class loaders for safety.
- Distributed garbage collection of active objects.
- Capability for multiple transports.

The RMI system consists of three layers: the stub/skeleton layer, the remote reference layer, and the transport layer (Figure 7). The boundary at each layer is defined by a specific interface and protocol. It is therefore independent of the next layer and can be replaced by an alternate implementation without affecting other layers in the system (RMI 1998).

A remote method invocation from a client to a remote server object travels down through the layers of the RMI system to the client-side transport, then up through the server-side transport to the server (RMI 1998).

The stub/skeleton layer is an interface between the application layer and the rest of the RMI system. This layer transmits data to the remote reference layer via a marshal stream abstraction. Marshal streams employ a mechanism called object serialisation, which enables Java objects to be transmitted between address spaces (RMI 1998).

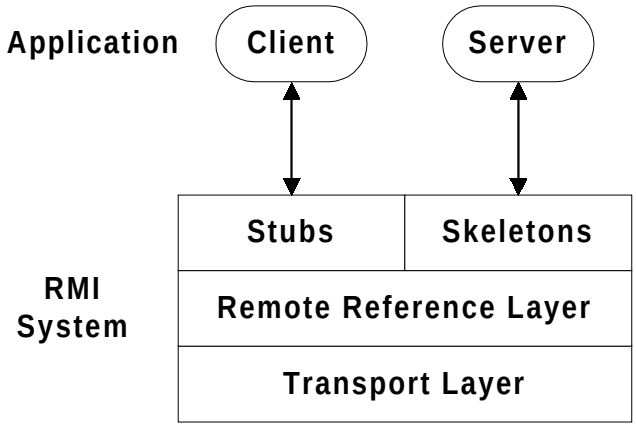

Figure 7. The RMI system.

The remote object stub functions as a client-side proxy for the remote object. It is responsible for (RMI 1998):

- Initiating a call to the remote object.
- Marshalling arguments to a marshal stream.
- Informing the remote reference layer that the call must be invoked.
- Unmarshalling the return value or exception from a marshal stream.
- Informing the remote reference layer that the call is complete.

A skeleton for a remote object is a server-side entity that contains a method which dispatches calls to the actual remote object implementation. It is responsible for (RMI 1998):

- Unmarshalling arguments from the marshal stream.
- Making the up-call to the actual remote object implementation.
- Marshalling the return value of the call or an exception onto the marshal stream.

The appropriate stub and skeleton classes are defined at run time and are dynamically loaded when needed (RMI 1998).

The remote reference layer deals with the lower level transport interface. This layer is also responsible for carrying out a specific remote reference protocol, which is independent of the client stubs and server skeletons (RMI 1998).

Each remote object implementation chooses its own remote reference subclass to operate on its behalf (RMI 1998).

The remote reference layer includes co-operating components for both the client-side and the server-side. During each method invocation, the client and server-

side components perform the semantics for specific remote reference (RMI 1998).

The remote reference layer transmits data to the transport layer via abstraction of a stream-oriented connection (RMI 1998).

The transport layer of the RMI system is responsible for (RMI 1998):

- Setting up connections to remote address spaces
- Managing connections
- Monitoring connection “liveness”
- Listening for incoming calls
- Maintaining a table of remote objects
- Setting up a connection for an incoming call
- Locating the dispatcher for the target of a remote call and passing the connection to this dispatcher

The Java distributed object model (through RMI) is similar to the Java object model in the following ways (RMI 1998):

- A reference to a remote object can be passed on as an argument or returned as a result in any method invocation.
- A remote object can be cast to any remote interface sets that are supported by the implementation. Casting is done by using the built-in Java syntax.
- The built-in Java *instanceof* operator can be used to test the remote interfaces supported by a remote object.

The Java distributed object model differs from the Java object model in the following ways (RMI 1998):

- Remote object clients interact with remote interfaces, never with the implementation classes of those interfaces.
- Non-remote arguments to and results from a remote method invocation are transmitted by copy, because references to objects are useful only within one virtual machine.
- A remote object is passed on by reference.
- Some of the methods defined by class Object include semantics that are specialised for remote objects.

- Since the failure modes for invoking remote objects are inherently more complicated than the failure modes for invoking local objects, clients must resolve the additional exceptions that can occur during a remote method invocation.

The following comparison relates RMI to the base ORB of the CORBA system. The other CORBA system services are beyond the scope of RMI (JavaFAQ 1998).

Java RMI is not CORBA-compliant, but rather the "pure Java" solution. Interfaces are defined in Java as opposed to CORBA's interface definition language (IDL) which is language-neutral (JavaFAQ 1998)

CORBA was designed for a language-independent distributed computing environment in which the underlying systems are expected to be heterogeneous and in which the objects are written in compiled languages. Java RMI, on the other hand, was designed for a single language environment where the objects are running in a homogeneous environment and where new code can be downloaded at any time (JavaFAQ 1998).

Differences arising from language dependence/independence can be detected in each model. The CORBA object model is defined by IDL and differs from any other object model in object-oriented language. Mapping the IDL object model into the object model of a particular language involves considerable effort. Even with C++, after which IDL was modelled, matching is not easily done (JavaFAQ 1998).

On the other hand, Java RMI has no separate interface definition language; remote interfaces are defined simply by using Java. This indicates that there is no translation from one language to another, since the language will remain the same as long as the Java object model is used (JavaFAQ 1998).

The differences caused by heterogeneous and homogenous environments are subtler. In the CORBA world, references are passed around and the code associated with a particular reference has to exist on both sides of the call. Owing to this, objects are passed as versions of declared types (as opposed to more derived types), since only codes of the declared type can be guaranteed to exist on the receiving side. For example, if a B-type object is passed on in a call which requires an A-type object, the object is perceived as an A in the receiving end. If it is further passed on as an A to a destination requiring a B, it will be rejected (JavaFAQ 1998).

Java RMI, on the other hand, always passes an object in its real type. If the code does not exist on the receiving side, it is simply downloaded. In the scenario above, when the object of type B is sent to a receiver asking for an A, it is still perceived as type B. When passed along to a third receiver, the receiver accepts the object as a representation of a B. If the right code for handling a B is not available, that code will be loaded (with all the appropriate security checks) (JavaFAQ 1998).

Another major difference is that Java remote objects are garbage collected, which is not possible with CORBA objects. CORBA objects are language independent, and therefore have to be consistent with languages that do not support garbage collection. A CORBA remote object, once it is created, exists until it is terminated. One of the major complications in writing a CORBA server is deciding when objects can be eliminated. As for Java RMI, object elimination is a part of the Java garbage collection system, extended by the RMI system (JavaFAQ 1998).

Yet another difference is that Java RMI does not match the interface repository of CORBA.. This is not required, since all the objects are Java objects whose types, methods, and fields are described by the loadable class files (JavaFAQ 1998).

2.6.3.3 InfoBus

InfoBus is a small Java API which permits data communication between JavaBeans and co-operating applets on a Web page. The technology standardises the manner in which applets and JavaBeans communicate within a Java Virtual Machine. InfoBus is a small specification with only 30 method calls. Presently, the implementation requires barely over 4k. InfoBus defines a small number of interfaces between InfoBus compatible components, and specifies the protocol for interface use (Infobus 1998).

InfoBus 1.1 is available for JDK 1.1.X. Version 1.2 is compatible with JDK 1.2.

InfoBus provides each applet with a mechanism to publish and receive data by name. InfoBus uses generic interfaces that enable data exchange between two mutually unaware applications. Which of the applications produces the data is irrelevant. In other aspects, Infobus is functionally quite similar to Dynamic Data Exchange (DDE) by Microsoft's Windows (Infobus 1998).

InfoBus architecture addresses interacting applets in one single Java Virtual Machine (and unlike RMI), not across multiple JVMs (Infobus 1998).

In contrast to an event/response model, where the semantics of an interaction depend on understanding an applet-specific event and responding to that event, with applet-specific callbacks, InfoBus interfaces have very few events and an invariant set of method calls for all applets. The semantics of the data flow are based on interpreting the contents of the data flowing across InfoBus interfaces, as opposed to responding to events and names from callbacks or events (Infobus 1998).

Components that form an InfoBus application can be divided into the following three types (Infobus 1998):

- Data producers

- Data consumers
- Data controllers

A data controller is an optional component that regulates or redirects the flow of data between data producers and consumers. An individual applet can simultaneously be both a data producer and a data consumer (Infobus 1998).

InfoBus protocol for data exchange supports the following major elements (Infobus 1998):

- InfoBus participation. Components are given a unique InfoBus context identifier.
- Rendezvous on the data to be exchanged. Data producers announce the availability of new data as it arrives. Data consumers solicit data from producers on acquiring it.
- Data encoding. Different data producers manage different types of data while consumers may request for this data in simple or complex ways. To accommodate the needs of both producers and consumers, InfoBus defines a determinate number of data access interfaces.

In order for a data provider to establish participation in InfoBus, the object must actively initiate the participation for sending or receiving data. Once the connection is made, the object can send DataBusEvents to announce and obtain data. If the DataBus Event has a data item name matching an item from a data producer, the Event sends a response. Some data producers can provide only one data item, some provide various items (Infobus 1998).

Data Consumers can request data on start-up by sending a DataBusEvent to InfoBus. The data consumer stands by for any data announcements from the producer, and as a response to the announcement, issues a DataRequest event to obtain the data (Infobus 1998).

2.6.3.4 Message Oriented Middleware

The CORBA 3.0 ORB will include the functions of Message Oriented Middleware (MOM). CORBA provides the following functions for MOM (Orfali et al. 1996):

- Uniform object-based interfaces. Currently, each MOM product provides its own non-standard API. Most of these APIs are procedural. CORBA offers a standard object interface, IDL support and a complete distributed object infrastructure.
- Technology for describing the message content. Currently, the message contents are opaque to MOM. CORBA enables MOM messages to be typed via IDL and provides generic data types.

- Endpoint definitions for a message exchange. Currently, each MOM has its own model of defining what constitutes an endpoint. An endpoint can be either a queue, a process, or an object. In CORBA, an endpoint is a unique object reference.

- Interoperable middleware. Each MOM vendor must define its own proprietary wire-level protocols. CORBA IIOP defines wire protocols that enable interoperation between products by multiple vendors. IIOP supports transactions and security as well.

- Internet foundation. CORBA IIOP is becoming a standard Internet protocol. CORBA will have well-known ports, URL mappings, and standard proxies for firewalls.

- CORBA provides MOMs with a strong object foundation and standard interfaces. CORBA will allow writing portable and interoperable MOM applications.

MOM allows clients to make asynchronous requests that do not block the client's execution thread. Writing and maintaining will be simple, since clients do not require multithreading.

MOM allows clients and servers to run at different times. This is a useful feature in loosely-coupled inter-enterprise situations. Owing to the rising popularity of the Internet, servers will be required to run in continuous mode.

Disconnected clients will be able to accumulate outgoing transactions in queues and perform a bulk upload as a connection with a server is established. MOMs does not require clients and servers to be simultaneously available.

MOM enables servers to determine when to select messages from their queues. The servers can use objects (servants) to choose messages either on FIFO basis or according to priorities. Filters are used in order to reject or re-direct unwanted messages. Queues can be persistent or non-persistent, depending on QoS definition.

QoS attributes are defined by both sides. Clients can set the QoS based on an individual call and they can define a callback object for managing the responses. Queue-aware servers must provide a MOA (Message Object Adapter) which can be derived from POA (Portable Server-Side Adapter). The MOA allows a queue-aware server to control in what order it processes the received requests. Messages themselves may be objects which can be stored, queried, retrieved, and passed around by using CORBA 3.0's pass-by-value, together with object services such as Query and Persistence. CORBA MOMs will most likely be built on top of IIOP, but will provide gateways to existing MOM products such as IBM's MQSeries, Digital's MessageQ, PerLogic's Pipers, Covia's Integrator, and Tibco's TibNet. MOMs can be integrated with such CORBA Services as Events,

Query, and Trader in order to provide powerful publish-and-subscribe systems for the Object Web. (Orfali et al. 1997).

MOM-style communications will make CORBA more flexible and time-tolerant. A modern ORB should support both MOM and request-reply styles of communication. SFAs (Server Framework Adapter) are language-specific frameworks for the server-side of CORBA (e.g. SFA/C, SFA/C++, and SFA/Java; maybe in CORBA 3.0). (Orfali et al. 1997).

The OMG TC document for CORBA Messaging covers three general topics: Quality of Service, Asynchronous Method Invocations and the specification of interoperable Routing interfaces. The specification includes asynchronous support for request transport and replies. The specification defines a set of new interfaces, extensions to existing interfaces, and language mapping for asynchronous method invocation (AMI). AMI is treated as a language mapping issue concerning client side. The specification proposes also alterations and additions to GIOP, including for example a definition of a standard Messaging Service Context and a new standard routing IOR Component. IIOP will also include some of the features available in MOM products, for example invocation and replies in a time independent fashion. The OTS must also be modified to support a new asynchronous model for transactions. (BEA Systems et al. 1998).

2.7 Standard interfaces

2.7.1 IDL

Interface Definition Language defined by OMG is a language for describing the interfaces of software objects. IDL is independent of programming languages, and can be used to describe objects which are implemented by using a variety of programming languages, compilers or operating systems.

The IDL object specification is used for automatic generation of “stub” and “skeleton” programs for the object. The stub provides an interface for other client objects to request services from the object via an ORB. The skeleton acts as an interface between the ORB and the object in its server role.

In addition, the IDL specification includes information concerning all objects that are connected with the ORB. The contents of the specification are also compiled and stored in the repository service provided by the ORB.

2.7.1.1 Structure

The following definitions determine the syntax for the IDL:

- **Interface Definition Structure**

  [interface_attribute, …] interface interface_name { declarations }

- **Import Declarations**

  import file, …;

- **Constant Declarations**

  const integer_type_spec identifier = integer | value | integer_const_expression;

  const boolean identifier = TRUE | FALSE | value;

  const char identifier = character | value;

  const char* identifier = string | value;

  const void* identifier = NULL | value;

- **Type Declarations**

  typedef [[type_attribute, …]] type specifier type_declarator, …;

- **Operation Declarations**

  [[operation_attribute, …]] type_specifier operation_identifier (parameter_declaration, …);

  [[operation_attribute, …]] type_specifier operation_identifier ([void]);

- **Parameter Declarations**

  [parameter_attribute, …] type_specifier parameter_declarator

2.7.1.2 Compilers

Commercial products that are CORBA-compliant include normally an IDL compiler which generates C++, Java, or both. The following compilers are independent of the CORBA product in question and provide a variety of possibilities for implementation languages.

- IDLtoJava (Sun) [http://java.sun.com/products/jdk/idl/index.html]

  The IDLtojava compiler generates portable client stubs and server skeletons that work with any CORBA-compliant Object Request Broker (ORB) implementation, provided that it includes the Java IDL ORB and is equipped with JDK™ 1.2. An ORB allows distributed web-enabled Java applications to invoke transparent operations on remote network services, by using the

industry standard Internet Inter-ORB Protocol (IIOP) defined by the Object Management Group.

- Flick (University of Utah) [http://www.cs.utah.edu/projects/flux/flick/]

  Flick is a compiler (or a "stub generator") for an interface definition language (IDL). It supports remote procedure call (RPC) and remote method invocation (RMI) for either client/server systems or for distributed object systems. What sets it apart from other IDL compilers is its capacity for a high degree of optimisation. In addition, it is capable of supporting several IDLs, message formats, and transport mechanisms. Currently, Flick has front ends for CORBA, Sun ONC RPC, and Mach MIG IDLs, as well as middle and back ends that support CORBA IIOP, ONC/TCP, MIG-style Mach messages, and Fluke IPC (see below). Flick produces stubs in C language.

- IDL to Modula-3 Translator
  [http://www.infosys.tuwien.ac.at/Research/Corba/sw-idl.html]

  The IDL to Modula-3 translator can translate OMG IDL definitions into Modula-3 interfaces, which can be used to form "CORBA-compliant" distributed applications.

2.7.2 MIDL

IDL is a language that has its origins in OSF DCE. It was originally used to define functional RPC interfaces in DCE Microsoft RPC. In Windows NT 3.5, the Microsoft Interface Definition Language (MIDL) compiler was extended to support COM interfaces as well. For both RPC and COM interfaces, the MIDL-generated code marshals the parameters, which have been passed onto a call stack, into packets that can be transmitted across process or host boundaries. Therefore, remote execution is possible. With the most recent version of MIDL, these same interface descriptions can be used to generate type libraries. Type libraries are binary descriptions of COM interfaces and implementations which are often used in OLE automation. (Box 1996).

MIDL produces headers and proxy-stub code in C (C++), but this does not imply that only clients of these languages can utilise the object. One method would involve converting the C header into such a version that would match the language in question, but this is hardly a satisfactory solution. Genuine language independence is achieved if an object provides a type library. This is based on the fact that all mainstream languages have either transparent or direct access to the type libraries and objects which they describe (Grimes 1997). Table 6 describes some common IDL attributes used in COM (Box 1996).

Table 6. Common IDL attributes in COM.

Attribute	*Context*	*Meaning*
uuid	interface	The IID of the interface, to be used by QueryInterface
object	interface	Indicates that the interface is a COM interface, not a DCE RPC function-based interface
pointer_default(ptype)	interface	Sets the default pointer attribute for pointers used in the interface
local	interface, method	Prototype production for the generated header file, no proxy/stub implementation
in	parameter	The parameter value must be transmitted from the caller to the object; can be combined with "out"
out	parameter	The parameter value must be sent from the object to the caller; can be combined with "in"
ref	parameter	The pointer parameter always points to valid memory (cannot be null), never to the same memory another method parameter
unique	parameter	The pointer parameter points to valid memory or is null, never pointing to the same memory as any other method parameter
ptr	parameter	The pointer parameter points to valid memory or is null, and may point to the same memory as any other method parameter
size_is(cElems)	parameter	The pointer parameter is an array that has a capacity for cElems elements
length_is(cElems)	parameter	The pointer parameter is an array. The first cElems elements are valid and should be marshalled
string	parameter	The pointer parameter points to a null-terminated string, and the result of strlen or wcslen should be used to determine the number of valid bytes to be marshalled

ALSTRA is a framework for rapid application development and deployment. It is defined for team development and extends the Visual basic environment by template-driven code generation. As an alternative to a Vbclient, ALSTRA can generate a web client for Microsoft Transaction Server using HTML and Vbscript. Business objects are generated as COM/DCOM server objects which are used by both VB clients and web clients. (Digital 1998).

2.7.3 Common Gateway Interface

Common Gateway Interface (CGI) is a slow, cumbersome, and stateless protocol. For this reason, it does not provide a good match for object-oriented Java clients (see Figure 5). It renders Web clients smarter and more interactive, whereas Java enables developers to create platform-independent client applications for mass distribution over the Internet. However, Java does little to improve the server side. The server can distribute some of its functions to the client by using applets it controls, but primarily it is still an HTTP/CGI server engine.

An electronic transaction is a process that requires multiple form submissions and leads to an electronic payment. The state of the transaction is maintained across form invocations. The CGI program processes a form and presents the user with the next form. This continues until the user makes the final payment or aborts the transaction. The CGI program stores information from old forms by using invisible fields. Therefore, information from old forms can be transported into new forms. In essence, the CGI program stores the state of the transaction in those forms it sends back to the client, instead of storing the information in its own memory. (Orfali et al. 1996).

Some server vendors are trying to extend CGI with proprietary server APIs, e.g. Netscape's NSAPI, Microsoft's ISAPI, NeXT's WebObject Framework, and Oracle's Web Server API. JOE by Sun is designed for creating a CORBA-based open Web server environment (Orfali et al. 1996).

2.7.4 Database interfaces

Databases have their own native APIs, but most of them support a generic ODBC or/and JDBC as a generic interface as well.

2.7.4.1 ODBC

Open Database Connectivity (ODBC) is Microsoft's API standard for SQL access under Windows. Visigenic received a license from Microsoft for proved ODBC SDKs on non-Windows platforms. In addition to Visigenic, Intersolv and OpenLink offer ODBC driver suites on Windows and Unixes.

These drivers match a variety of database servers. ODBC 2.0 defines API calls that fall into three levels:

- Core provide 23 base calls that enable connection to a database, execute SQL statements, fetch results, commit and rollback transactions, handle exceptions, and terminate the connection.

- Level 1 provides additional calls that enable retrieving information from a database catalogue, fetch large objects (BLOBs), and operate with driver-specific functions.

- Level 2 provides additional calls for retrieving data. This is managed by using cursors which include forward and backward scrolling.

X/Open CLI includes ODBC's Core and some of the functions described in Levels 1 and 2. Most database server vendors support the ODBC 2.x API in addition to the native SQL API. ODBC drivers for the respective servers are also included. The problem is that ODBC always seems to be the second option to the native interfaces on the client and server sides.

The most serious drawbacks are that the specification is controlled by Microsoft, and evolves constantly. In addition, ODBC drivers are difficult to build and maintain. The current drivers have different ODBC conformance levels, which are not well documented. The ODBC layers introduce considerably overhead, especially for SQL updates and inserts. Furthermore, they are not as fast as the native APIs. (Orfali et al. 1996).

2.7.4.2 JDBC

The specification for Java Database Connection (JDBC) is published by Sun 1996. JDBC is a set of Java classes that provides an ODBC -like interface to SQL databases. JDBC uses a driver manager, which automatically loads the right JDBC driver for interaction with a given database. Unlike ODBC, JDBC must address the mobile code issues. A driver that is downloaded as an applet can only be allowed access to its home database. An applet finds its JDBC driver by using a URL-based naming scheme: jbdc:<subprotocol><domain name>. Sun will bundle a JDBC driver manager with future releases of Java and it will also provide a JDBC-to-ODBC bridge. In the future, JDBC might become a more important database API standard than ODBC or OLE/DB. (Orfali et al. 1996).

2.7.5 Application interface library

2.7.5.1 JAPI

Java libraries extend Java language by providing a portable environment for writing thread-safe Java applications. Classes of Java.lang support basic Java objects and native types which are always needed in every application. Java.io supports reading and writing streams, files, and pipes. Java.net consists of classes for network programming, including sockets, telnet interfaces, HTTP, and URLs. Java.util has a collection of utility classes: dictionaries, hash tables, stacks, dates, strings, etc. Java.awt is an abstract windowing toolkit (AWT) and provides a portable GUI layer for writing applications. Java.applet is a subclass of AWT and supports animation and audio. (Orfali et al. 1996).

2.8 Java clients with CORBA orbs

The ObjectWeb model is a 3-tier client/server application model that consists of Java clients in the first tier, CORBA business object in the middle tier, and traditional servers in the third tier (Figure 1).

CORBA business objects provide the application logic and encapsulate existing database, TP Monitor, and groupware servers by replacing CGI applications in the middle tier. The Java client can communicate directly with a CORBA object by using the Java ORB and by replacing HTTP/CGI as the middleware layer for object-to-object communication. Because CORBA's IIOP uses the Internet as its foundation, HTTP and IIOP can run on the same networks (= live-and-let-live environment). This implies that a Java ORB is incorporated with the Web browser, and that HTTP continues to service HTML documents and the legacy CGI applications. The benefits offered by CORBA are:

- CORBA avoids the CGI bottleneck, since client/server overhead is kept to a minimum, especially in comparison to HTTP/CGI.

- CORBA provides a scalable server-to-server infrastructure. Server business objects communicate by using the CORBA ORB, and can run on multiple servers in order to provide load-balancing for incoming client requests.

- CORBA extends Java by a distributed object infrastructure. Currently, Java applets cannot communicate across address spaces by using remote method invocations. CORBA enables Java applets to communicate with other objects across address spaces and networks and irrespective of the language (Orfali et al. 1996).

2.9 Compound documents and Object Webs

However, in addition to Java and CORBA, the Object Web must be augmented with compound documents as well. Compound document frameworks, such as OLE and OpenDoc, provide two key technologies (Orfali et al. 1996):

- A visual component foundation for creating open Web browsers, and
- container technology for distributing, caching, and storing groups of related components and their data, i.e. shippable places.

Components and Java applets will be able to co-operate seamlessly within a browser window. The contents of any component can be edited for drag and drop within browser, as well as between the browser and the surrounding desktop. In a compound document browser, components can take any shape, and they can be moved and embedded.

Compound documents provide an open, well-defined, and extendible architecture for plug-ins. Java applets and Netscape plug-ins provide some form of in-line rendering which means that they can display their contents within a web browser window. Java applets and plug-ins can only display their visual contents within a pre-assigned rectangular area. The contents cannot be freely embedded other components, and it is not possible to move them around. Plug-ins function only according to the rules of the browser which they were designed for (Orfali et al. 1996).

2.9.1 OpenDoc

OpenDoc is an open, multiplatform architecture designed for component software. OpenDoc enables building lightweight container applications and component parts called Live Objects. The emphasis is therefore on creating components that meet a specific need, instead of investing time and resources in delivering a complicated application. This approach improves desktop computing by providing an object-based framework, which develops applications that are fully integrated and interoperable across platforms and distributed networks (OpenDoc 1998; PCweek 1998).

OpenDoc is based on CORBA, which has been developed and maintained through an open industry standard group OMG. OMG is currently defining a Compound Document Framework which will essentially be an abstract standardisation of OpenDoc. The OpenDoc technologies themselves are maintained by the open industry group CI Labs, which welcomes new members to participate in technology developing task forces (Braddock 1998).

OpenDoc enables also seamless interoperation with Active/X objects by usage of OpenDoc-glue. Active/X has made no such effort to interoperate with OpenDoc (AFAIK). Both provide similar functionality, but since OpenDoc is based on a

superior object system (IBM SOM/DSOM, which is an extension of CORBA), its potential for distributed computing is probably greater (Braddock 1998).

OpenDoc implementations already exist on Windows, Mac, AIX, and OS/2 platforms, which means its portability is secured. Active/X (OLE) is only available on Windows platforms (AFAIK), though Microsoft is working on porting it to Unixes and Macs. (Braddock 1998).

2.9.1.1 OpenDoc terminology

Document

A collection of OpenDoc parts, assembled by a user or a developer. A part becomes a document, if dragged from its document to the desktop. A document becomes a part, if dragged from the desktop into an open document (ITU 1998).

Part

The fundamental building block of an OpenDoc document. This is the content that users are able to see in their documents. The functionality in the associated part editor or part service allows the user to manipulate the part. Part viewers allow the user to view the part, although editing is excluded (ITU 1998).

Container Application

A monolithic application that has been modified to support embedded OpenDoc part editors and part services (ITU 1998).

Part Editor

A part editor displays the contents of a part, facilitates content manipulation and provides an user interface for modifying that content. This user interface may include menus, controls, tool palettes, rulers, and other modes of interaction. A text editor, for example, is a part editor (ITU 1998).

Part Service

A part service provides "back-ground" functionality for a part, and provides the necessary user interface for manipulating the content of that part. Database access functionality is an example of a service that could be added to an OpenDoc document. Its user interface would be a menu item or a database query screen (ITU 1998).

Part Viewer

A part viewer offers some features of the functionality provided by part editors: it enables users to display and print the content of a part, although editing is not

possible. Viewers can be useful in situations where documents are shared. For example, if the recipient of a document does not have authorisation for some of its content, or when the person sending the document does not want the recipient to alter the document (ITU 1998).

2.9.1.2 OpenDoc and JavaBeans

JavaBeans is a new technology that Sun and IBM are currently developing. The objective is to provide Java applications with similar compound document functionality than currently available in both OpenDoc and Active/X. Its implementation is "abstract", i.e. it can be based on either OpenDoc or Active/X (OLE) (Braddock 1998).

JavaBeans is useful for implementing small pieces of component software (called beans). These include small interface elements such as buttons, check boxes, scroll bars, lists, sliders, and text fields. This feature corresponds to Microsoft's Visual Basic (Apple 1997).

JavaBeans is useful also for creating non-visual components that provide services to the developer. One relevant example of a non-visual component is the FTP-service component. It has no visual interface. However, the component is able to retrieve files from an FTP site, after having received their addresses from the developer (Apple 1997).

One characteristic of a Java bean is that the user can manipulate it (for example, by clicking a button bean). However, moving or resizing the bean inside its container is not possible. Similarly, neither adding a new button to the container, nor changing the attributes of the button during actual usage are available. A bean is limited to a rectangular shape, and two beans cannot overlap. In addition, a bean including other beans cannot very well manipulate its own content outside the embedded beans (Apple 1997).

Live Objects are OpenDoc parts that have passed a certification process set up by CI Labs. CI Labs, in its turn, is the group responsible for setting standards for OpenDoc. Live Objects have several characteristics which are not available in JavaBeans. Users can move and resize Live Objects within their containers. In addition, the attributes of Live Objects can be altered. Adding or copying Live Objects can be managed by a simple drag-and-drop operation. Furthermore, Live Objects can have any shape and can overlap (Apple 1997).

A more important difference with reference to JavaBeans is that by using OpenDoc, larger component-based solutions can be created. The following OpenDoc characteristics are not available in JavaBeans (Apple 1997):

- Multilevel undo and redo of actions.

- A document storage model based on Bento, a cross-platform file format that can contain any type of data and can store alternate representations of a given data type.

- Versatility in viewing data: Live Objects can store their data in standard formats, which allows the same data in a document to be handled by different editors, depending on the set-up and preferences of the user. The content of a Java bean, on the other hand, is inextricably connected to the JavaBeans object which is used to display it. The user is therefore not allowed to define how the content can be viewed and manipulated.

- Binding enables a qualifying part editor of the user's choice to become activated when the user clicks on a given type of data.

- Embedding, which enables one user to include certain data type, for example a QuickTime movie, in an OpenDoc container document. Furthermore, one user can send the document to another user, who in turn can open the document and view the embedded data type.

Once the necessary programming infrastructure is built, combining Java Beans and Live Objects within the OpenDoc artefact is possible. This enables creating component-based solutions quicker than with OpenDoc alone (Apple 1997).

Table 7. Features of Java, JavaBeans and OpenDoc.

Feature	**Java**	**Java Beans**	**OpenDoc**
Drawing	*		
Platform independence	*		
Security	*		*
Lightweight embedding	*		*
Lightweight layout	*		*
Lightweight data interchange		*	*
Lightweight persistent storage		*	
Visual property editing		*	
Simple scripting		*	
Sophisticated OSA (Open Scripting Architecture) scripting			*
Undo			*
Palettes			*
Shared menus			*
Replaceable editors			*
Arbitrary embedding			*
Sophisticated layout			*
Rich data interchange			*
Structured persistent storage			*
Language neutrality			*
Based on open standards			*

Creating interface elements is quicker with JavaBeans than with OpenDoc. To begin with, Java language is easier to program in and debug than C++, which is used in creating Live Objects. In addition, writing interface-element beans might not be required at all. Since beans are inherently cross-platform, there will probably be a developer-to-developer market for preconstructed, commonly used interface-element beans (Apple 1997).

Once the Java beans and Live Objects are available, it is possible to combine them within the OpenDoc architecture in order to create complete and robust component-software solutions (Apple 1997). In Table 7, features of Java, JavaBeans and OpenDoc are compared.

2.9.2 DCOM Object Web

Microsoft Object Web is an Active Internet Platform (AIP), which consists of a 3-tier client/server architecture. Clients are able to access the Internet via component-based browsers such as the Internet Explorer 3.X. In addition to ordinary HTML pages, the browser is able to play DocObject titles which are three-ring binder documents containing pages. A page can contain OLE ActiveXs, Java applets, Visual Basic applets, in addition to regular HTML content. DocObjects can play inside other OLE containers. In theory, a title is a shippable place. To achieve safety, Microsoft has signed up VeriSign to issue digital signatures for ActiveXs components. A VeriSign certificate ensures that a component has not been tampered with after leaving its creator. OLE document titles can be created by using the Microsoft Internet Studio framework, which is a drag-and-drop tool for assembling and laying out components within pages and binders. Jakarta tool supports Java within the framework. Jakarta includes an interactive debugger, App Wizard for generating new Java applications or applets, Class View for browsing classes, and simple tools for importing foreign classes such as Java classes or COM objects. (Orfali et al. 1996; DiLascia 1998).

The second tier consists of Microsoft's Internet Information Server (IIS), which is bundled with NT server and is practically free. IIS provides an application framework for running OLE-based business objects which enable invoking server business objects via the Network OLE ORB. Business objects use OLE for interaction. Legacy applications are supplied by ISAPI, ODBC, and APIs for CGI.

The third tier consists of Microsoft's BackOffice. Microsoft is planning to provide an OLE-based TP Monitor (called the Component Coordinator) for organising transactions across different resource managers. (Orfali et al. 1996).

DocObjects are OLE documents that function as their own containers. A binder can be saved, viewed, printed, copied and moved as a single entity. DocObjects are being repositioned as mobile OLE component containers for the Internet. DocObjects provide a standard method for traditional applications to plug

themselves into a Microsoft Web browser's window frame. A DocObject page can contain lightweight OCXs - or ActiveXs - as well as other component types. A DocObject provides its own storage and introduces new interfaces: IOleDocument and IOleDocumentView on the component or OLE server side, and IOleDocumentSite on the container side. Technically speaking, a OCX (or ActiveX) is required to function as a self-registering DCOM object; it must implement OLE's *IUnknows* and *IClassFactory* interfaces. Component categories enable controls to give directions to their containers (Orfali et al. 1996).

Controls belonging to the InternetAware category load their data via asynchronous monikers, which is a persistent naming service. Browsers use an asynchronous moniker to download files by using separate background threads. The moniker notifies the browser via a callback when it has completed the download. The URL moniker is an implementation of the asynchronous moniker; it is used to encapsulate Internet URLs. OLE's URL moniker is remarkably similar to the Cyberdog construct (Orfali et al. 1996).

2.9.3 CORBA Object Web

The technology in the open Object Web is very similar to Microsoft's Object Web, with the exception that all technologies are based on open standards. Object Web is a 3-tier client/server architecture. Clients belong to Java components, ORBlets, OpenDoc compound documents, and shippable places.

Clients are able to access the Internet via component-based browsers (Cyberdog or Netscape). An OpecDoc title is a shippable place; it is able to contain ActiveXs, OpenDoc parts, Java applets, and regular HTML content. OpenDoc parts can contain also OLE OCXs and ActiveXs.

The second tier services both HTTP and CORBA clients. CORBA objects act as a middle tier and encapsulate the application's business logic. They interact with clients by using IIOP over the Internet. The third tier includes TP Monitors, DBMSs, Lotus Notes, etc. The second tier acts as a store for component titles and shippable places which can be stored in shippable Bento files managed by a ODBMS. Bento files function as user-defined data types which can run on multiple operation systems; Bento files was designed to be a portable component store. However, getting all these technologies to interact requires a great deal of inter-vendor cooperation. (Orfali et al. 1996).

2.9.4 Trends in Web technology

The Web will be recreated on top of a distributed object bus; the two main contenders for the bus are CORBA and DCOM. HTML will glue onto a compound document container; the two contenders are OpecDoc's Bento and

compound files of OLE. The three contending camps for live component are CORBA-enabled Java components, CORBA-enabled OpenDoc parts, and OLE ActiveXs. As for the tool market area, the competition is between Symantec's Java-based Cafe and Microsoft's OLE-based Internet Studio.

Intranets will extend the scope of corporate networks. Corporates are moving from private networks to Intranets. The common Internet technology links up suppliers, customers, and business partners. Corporations will increase their use of the Internet's wide-are security mechanisms (including SSL, S-HTTP, and public keys).

The Web will create a demand for small Internet PCs. In addition, there will be a substantial demand for lightweight Java machines. Components and shippable places will become the primary software distribution mechanisms in the Web PCs. Java machines have the potential to outdo Microsoft Windows in many new environments.

Mobile BLOBs, applets and components run within "containers". Scalable ODBMSs are used to track relationships between objects and their containers. Mobile containers or structured files will be used to ship groups of related objects and BLOBs to any destination on the WWW. (Orlafi et al. 1996).

The demand for mobile agents will in all likelihood increase rapidly, owing to their ability to decrease network load, encapsulate protocols, and perform asynchronous and autonomous execution. In addition, mobile agents offer automatic re-allocation and tolerate changes in the environment with dynamic flexibility. Furthermore, mobile agents are heterogeneous, robust and fault-tolerant services which are used by networked applications. There are several applications which will benefit from the services offered by mobile agents, for example telecommunication, work flow process control, groupware, electronic payments, and secure information brokering.

2.10 Integrating legacy systems

Wrapping techniques provide a way of integrating legacy systems with each other and with new software. Object wrappers re-encapsulate the system in a controllable form and isolate complex subsystems from each other. This technique creates systems that are more adaptable. Wrappers can employ multiple integration mechanisms such as files, sockets, remote procedure calls, and scripts.

Object wrapping can be divided into the following categories (Mowbray & Zahavi 1995):

- Layering
- Data migration

- Reengineering applications
- Middleware
- Wrappers for architecture implementation, and
- Wrappers for mediators and brokers.

Layering is the most basic level of wrapping. A layer is a mapping from one form of application program interface (API) to another. The functionality of the interface provided by layering depends on the APIs to the legacy system.. The functionality of the legacy systems can be extended by adding wrapping code to the layer. Layering also can be used to aggregate multiple legacy systems. (Mowbray & Zahavi 1995).

Data migration is used when systems have anomalies such as overlapping database tables in their schemes. Migration involves moving the data to another data mode. In this case, the wrapper consists of a layering code.

When an application is wrapped, it can be reengineered one piece at a time, if necessary. Wrapping enables replacing the old system components with object-oriented components. (Mowbray & Zahavi 1995).

The term middleware is used to describe a wide range of software for commercial system integration. Distributed processing middleware is shifting towards CORBA. Database and user interface middleware are upper middleware including software. The software mediates between various database products and creates a common access mechanism. In a CORBA-based software architecture, database middleware should be wrapped in a similar manner as in other legacy systems. (Mowbray & Zahavi 1995).

Encapsulation is the most general form of object wrapping. It is a black abstraction in which the interface hides the details of the underlying implementation and separates the interface from implementation.

At the architecture level, a wrapper is required to implement the architecture design in all aspects. It should provide interoperability between architecture, legacy subsystem, value-added functions, information (such as metadata, data conversions) and other architecture features.

The diversity of processing services introduces a requirement for a mediator or brokerage service capable of aggregating various type of functions, such as the following (Mowbray & Zahavi 1995):

- Access to disparate information sources and value-added services.
- Sophisticated search and data presentation algorithms to reduce client complexity and to simplify the user interface.
- Data conversion between incompatible formats.

Implementation trade-offs are often made at the expense of either performance or flexibility. For instance, a simple conversion server provides rather poor configuration flexibility, while requiring only one ORB hop. If the conversion is not direct, and more formats are involved, this form is the most efficient one (Figure 8.a). If each conversion server is separate, they can be placed on different systems and this is as efficient as would be with only one server (Figure 8. b). The broker provides additional flexibility since the service selection reasoning is separated from both the conversion servers and the clients (Figure 8.c). This is the most flexible solution since it enables placing conversion servers on separate systems. The difference in reference to earlier solutions is extra network overhead, while the cost is several hundred milliseconds per call.

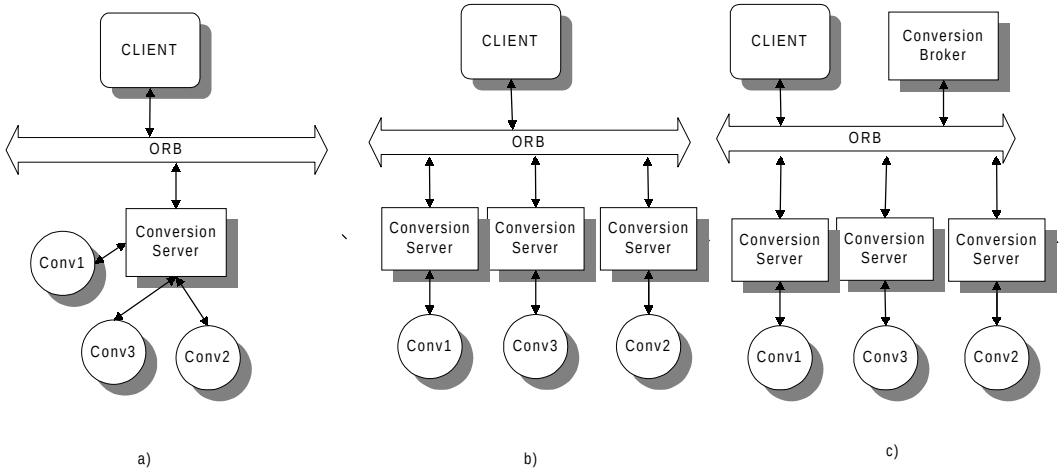

Figure 8. Alternatives for implementing object wrappers.

3. Commercial implementations for distributed objects

3.1 Iona's Orbix

Orbix by IONA Technologies a full implementation of the Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA). It combines distributed systems with object orientation in order to provide a distributed programming environment. It enables software interfaces to be defined in a standard language, and renders them accessible from any location in a distributed system. The communication mechanism is provided by the environment; the role of the programmer is to design and implement one or more server objects and a client which invokes the server (Buckner et al. 1997).

Orbix is compliant with CORBA 2.0 standard and supports the standard IDL to C++ mapping, which is defined by OMG as well as extended with a number of added value facilities (University of Paderborn 1997).

Orbix reduces substantially the costs of developing distributed applications and integrating existing components. It enables a system to be constructed as a set of interacting objects. Each of these objects is equipped with a well-defined interface, and Orbix allows these objects to communicate easily with each other, irrespective of details such as (University of Paderborn 1997):

- The hosts that these objects run on.
- The operating system that these hosts run.
- The programming language in which the objects are implemented.

The latest Orbix release also implements a programming model tailored for creating multithreaded applications. According to the officials, it supports also the Internet Callback Mechanism (ICM) and Secure Sockets Layer (SSL) security (IDG-Net 1998).

3.2 Visigenic's Visibroker

Unlike other CORBA ORBs (such as Orbix) which require that a software daemon is installed and configured on every node, VisiBroker is a fully dynamic system. VisiBroker ORB eliminates the necessity for maintaining or updating configuration files, therefore reducing the overhead requirement for installing and managing applications (Visigenic 1998).

The Location Service, which is a unique VisiBroker design, allows the dynamic discovery of all available objects and Smart Agents on a network. Smart Agents

provide a dynamic directory of all objects on a network. The Location Service can query Smart Agents for either all object instances of a specific type or for a particular instance of an object. Results can be returned as either object references or as complete descriptions of the instance, including the name, host name and the state of the object (Visigenic 1998).

The Naming Service enables associating meaningful names with objects, while the Event Service allows decoupling object communication. In addition, it enables asynchronous data distribution to multiple objects (Visigenic 1998).

VisiBroker supports IIOP over the industry-standard Secure Socket Layer (SSL) protocol, and provides secure communication between clients and servers (Visigenic 1998).

Since VisiBroker fully implements the IIOP protocol, C++ client programs can invoke methods on CORBA objects written in Java, and vice versa (Visigenic 1998).

VisiBroker provides native support for single and multi-threaded applications. It offers two multi-threading models that ensure a high level of scalability on the server: the thread-per-session model and the thread pooling model (Visigenic 1998).

Efficient communication is ensured by minimising the number of client connections to the server. All requests from the same client are multiplexed over the same connection, even if they originate from different threads. Client connections are recycled for subsequent reconnects to the same server. As a result, the need to incur overhead by creating new connections to replace released ones is eliminated (Visigenic 1998).

VisiBroker ensures that seldom-used objects are made available only when needed. When a request is received, an Object Activation Daemon (OAD) is used for dynamic and automatic launching of a server process. Additionally, VisiBroker enables the creation of local object references (which is useful for objects that are only valid for the lifetime of the process and do not need to be registered with a Smart Agent). Since these objects are not registered, overhead for such object references that do not require dynamic discovery is reduced (Visigenic 1998).

VisiBroker ORBs are fully CORBA 2.0-compliant and have native implementations of the Internet Inter-ORB Protocol (IIOP). IIOP is the industry communication standard for the Internet and intranets. This ensures high performance and interoperability for distributed applications, including interoperability with Web-based applications. VisiBroker's native implementation of IIOP is efficient since it eliminates the necessity for a bridge to a proprietary protocol (Visigenic 1998).

VisiBroker is compliant with OMG-specified IDL mappings. The C++ compiler complies with the 1.1 IDL to C++ mapping, and the Java compiler with the 1.0 IDL to Java mapping (Visigenic 1998).

3.3 Expersoft's CORBAplus

CORBAplus is an interworking solution for CORBA and DCOM (including bi-directionality and support for Automation and COM). In the following, a description of CORBAplus product line is presented (Expersoft 1998):

- CORBAplus for C++ is a CORBA 2.0-compliant Object Request Broker (ORB). It is used for developing enterprise-class distributed object applications.

- CORBAplus JavaTM Edition combines Java and CORBA, enabling creating Internet and Intranet applications which span the enterprise.

- CORBAplus ActiveX Bridge creates a bridge from CORBA to COM by providing Windows desktop clients with unprecedented levels of interoperability and flexibility.

- CORBAplus Transaction Service is an object-oriented transaction processing system that delivers such guaranteed data consistency as required for building enterprise-class distributed object application environments.

- CORBAplus Enterprise Edition is the Net integration server - the industry's first technology integration between CORBA-based ORB and message-oriented middleware.

3.4 Software AG's EntireX

EntireX is an open, enterprise-wide component server platform that functions by speeding up development cycles and establishing flexible, modular application landscapes. EntireX is a combination between Microsoft's Distributed Component Model technology (DCOM), powerful multi-platform integration and message brokering capabilities.

The EntireX provides the following services:

- DCOM for integration with the desktop.

- Broker for message-oriented middleware (MOM).

- Broker Services for the following enhancements: APPC Adapter for integration of CICS and IMS environments via LU6.2, MQSeries Adapter as an interface to IBM's message-oriented middleware, and Attach Service for dynamic replication and launch of servers.

- Software Developer's Kit for mix-and-match development tools and interfaces for all environments.
- Security interface to SAF for central access control.
- Runtime execution environment for applications created or equipped with EntireX technology.

Platforms: Servers with MVS, VSE, BS2000, UNIX and Windows NT. Clients with Windows 3/95/NT, Macintosh, OS/2, OS/400, UNIX, OpenVMS, VSE, MVS and BS2000.

Interface description language is most likely MIDL with DCOM wrappers.

Interfaces: ACI (Advanced Communication Interface) between the Broker and the Broker Services, which implies that EntireX ActiveXControl, Java classes, and DCOM Wrapper are available with the SDK. Communication supports MOM via ACI stubs and RPC via RPC stubs (Figure 9). (SoftwareAG 1998).

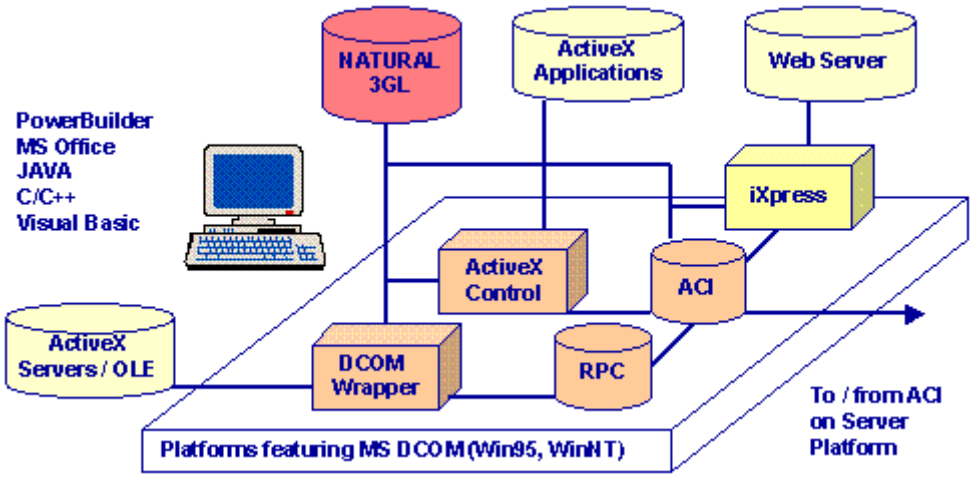

Figure 9. EntireX's components on typical client platforms.

3.5 Comparison between commercial middleware products

Features of commercial middleware are listed in table 8.

Table 8. Available features of commercial middleware.

Feature	**CORBAplus**	**VisiBroker 3.0**	**OrbixWeb 3**
Java			
C++			
Native IIOP implementation			
Firewall-enabled IIOP callback			
Static and dynamic invocation interfaces			
Interface and implementation repositories			
"In-process" server object instantiation			
Full support for HTTP-tunnelling			
IDL to C++ compiler			
IDL to Java compiler			
Java to IDL compiler			
Java to IIOP compiler			
Extended IDL data types			
Full server/callback support			
Local/remote administration of implementation repository			
Local/transient and global/persistent object references			
Support for single and multi-threading			
Optimized object binding and communication			
Central repository for all configuration information			
No configuration files			
Support for ORB extensions			
Secure object communication with SSL			
Sophisticated discovery of available objects			
Runtime can be transparently updated on the fly			
Naming service			
Event service			
Available for Windows 95 & NT			

Requirement	CORBAplus	VisiBroker 3.0	OrbixWeb 3
Java Virtual Machine	JDK 1.0.2 || JDK 1.1.X	JDK 1.1.X	JDK 1.0.2 || JDK 1.1.X

4. Experiences concerning the use of commercial middleware

4.1 Orbix AND VisiBroker

Orbix has been applied in an earlier case study by VTT Electronics concerning flexible manufacturing systems (Holappa 1998; Niemelä & Holappa 1998) . Visibroker was only briefly evaluated and used also with Orbix.

Installing and using VisiBroker was easy. There were no problems in establishing connections between client and server when both ends were using the same ORB. Connection between Visibroker on the server side and Orbix on the client side was also attempted, but the attempt was unsuccessful. The problem appears to involve different kinds of implementations in the CORBA Naming service. In practice, the IIOP between two commercial CORBA implementations does not function. Visibroker's bank account example was used as an application.

4.3 CORBAplus

The evaluation was carried out by a simple two-tier architecture. The application had a client-part and a server-part. The server was written in C++ and the client in Java. The C++ part was written with Visual C++ 5.0, and the Java part with a text editor and JDK1.1.5. The Web server used was OmniHTTPd v2.0a8 (Omnicron 1998).

A distributed phone book of company employees was used as an application example. The application uses the following IDL :

```
struct Employee
{
  string Name;
  string Abbreviation;
  string Phone;
};
typedef sequence<Employee> EmployeeList ;
interface PhbServer
{
  boolean AddEmployee(in string name, in string Abbr,
  in string phone);
  boolean GetEmployeeList(out EmployeeList mylist);
  boolean RemoveEmployee(in string Abre);
  boolean QueryEmployee(in string querystring,
  out EmployeeList mylist);
  string Hello ();
};
```

The server part uses Objectivity/DB for storing employee instances and creates simple queries from given predicates. The result of the query is returned to the client as a sequence.

The client is a Java applet running on a Web Browser, either Netscape Navigator 4.05 or Internet Explorer 4.0. The client connects the server when a button requesting an action is clicked. In Figure 10, a screen shot of the Java client GUI running on Netscape Navigator is shown.

Figure 10. Screen shot of the Java client.

Java ORB initialisation was performed in the following code :

```
public void InitializeOrb() {
  try {
    if(bFirstTime == true) {
// Netscape's default ORB override
       java.util.Properties p = new java.util.Properties();
       p.put("org.omg.CORBA.ORBClass",
       "com.expersoft.CORBA.ORB");
// Initialize the Orb.
       orb = org.omg.CORBA.ORB.init(this, p);
// This is the url we will connect to:
       String url = "iiop://ele302:10000/PhbServer1";
       orbObject = ((com.expersoft.CORBA.ORB) orb).
       string_to_object(url);
// narrow
       phbsRef = PhbServerHelper.narrow(orbObject);
       bFirstTime = false;
    }
    else {
// connect
       orb.connect(orbObject);
```

```
    }
  }
  catch (Exception exc) {
    exc.printStackTrace();
  }
}
```

4.3.1 Problems Encountered

The evaluation was carried out by using the CORBAplus demo pack version 2.2.0. Some of the problems may have been caused by the demo pack version.

4.3.1.1 C++

- Sequence problems. No thorough examples were available. The product support was very useful.
- Stub code generation. Unlike Orbix, C++ does not generate skeleton code.
- No debug libraries.
- Inadequate documentation.

4.3.1.2 Java

- Inadequate documentation. No documentation for ORB classes.
- Problems with different versions of Web browsers. In some versions, the default ORB had to be overridden. Older versions did not function at all.
- Problems with different versions of JDK's (although these depend on JDK, not on CORBAplus).

4.3.2 Merits

C++

- Adequate support. The response time of the product support was short.
- Gives a "mature impression". The product seems complete.
- Works simultaneously with Orbix daemon. Interoperability was not tested.
- Naming and Event services are included.

Java

- Possibility to override default ORB in Web browser either in applet code or in the HTML file.

5. Embedded middleware services

Middleware, which is used as a generic software platform for embedded client systems, requires communication services for remote procedure calls, events and messaging (Figure 11). The services are applied to the CORBA service categories (Mowbray 1997). Naming and security services are also needed if the embedded systems are connected to Internet or to other open networks. These services, in addition to the generic interfaces which isolate the middleware from the operating system and protocols, have to be provided as component-based software. Since applications are required to be different types of components (i.e. Java beans or applets, ActiveX components and CORBA components), three application interfaces should be supplied.

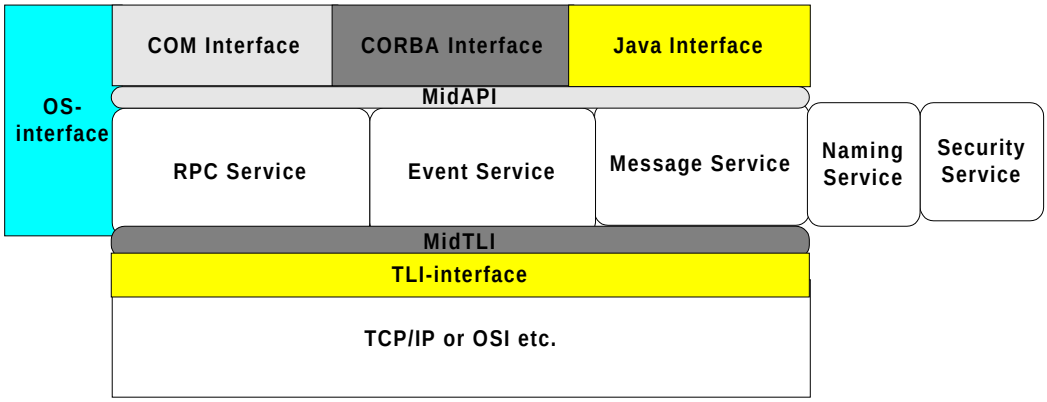

Figure 11. Overview of the basic services of embedded middleware.

5.1 Interfaces

The embedded middleware is designed for use in different types of execution platforms. Therefore, the software portability has to be supported by developing a generic transport layer interface which can be used with Internet and OSI suite protocols. If the TLI interface is designed as a generic interface, changes in the communication media or protocols beneath the session layer should not have any effect on other components in the embedded middleware.

The independence from the operating systems can be provided by using a bridge design pattern which, through ExOSInterface class, decouples the abstract operating system class (i.e. MidOSInterface) from the operating system and its interface (Figure 12). The method assures that the middleware services (described here as RPCInterface, EventInterface and MessagingInterface), and the operating system services, can vary independently (Figure 12). The same technique can be applied in order to increase the reuse of software in developing interfaces (i.e. COMInterface, CORBAInterface, and JavaInterface) for TLI and application

components. However, only one interface specification, i.e. the generic MidAPI component, is required to describe the services offered for applications. Similarly, the services required from the transport protocol layer can be described by one single interface specification, the MidTLI component.

The wrapping technique can also be utilised, but it requires two layers: one for middleware clients, and another for operating services. With respect to memory size and performance, the bridge pattern would appear to be more suitable for embedded systems.

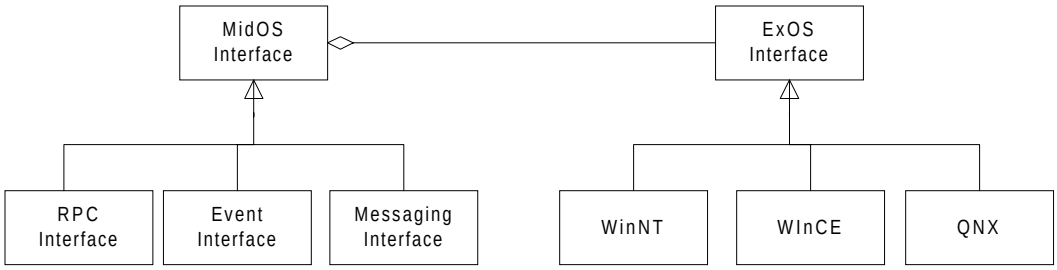

Figure 12. Isolating the impact of the execution environment.

5.2 Basic services

The basic services of embedded middleware support software flexibility. However, applications set certain requirements for the quality of the services (QoS), and these requirements must also be fulfilled. The following QoS requirements are used in applications that are time critical or real-time:

- Performance. The number of transactions or messages per a second, i.e. ability to respond in real-time.
- Reliability. Mean time to failure and mean time to repair (MTTR) which should be defined for the users of the embedded middleware services.
- Scalability. Number of systems, objects or processes that can be manipulated by the system before its performance specifications become impossible to fulfil.
- Resource consumption. The amount of memory and disk space required to support a service.

5.2.1 Communication services

Remote Procedure Call

The remote procedure calls are the basic communication methods used by application components in the client/server architecture. CORBA is based on ONC and DCOM uses its own RPC which is based on the DCE RPC. Since

CORBA producers seem to be moving from ONC to OSF DCE RPC (which includes also the security services), it is a suitable basis for embedded middleware

Event Service

Several embedded systems act as sensors or observers which notify a server or servers by sending events. Therefore, the event service is actually required as a basic service in the embedded middleware. It also decouples the producer and consumer applications. Furthermore, it provides the opportunity to develop loosely connected applications which can be used in different product variants or with add-ons of the product family.

The minimum requirement is the push mode. However, when events are triggered at known timing intervals, the pull mode is also useful.

The event service must be guaranteed to deliver the events to receivers. Consumer and event service protection means that if a consumer drops off-line, on-line events will be put on queues and delivered when the consumer returns. If the event service fails, event producers will still be able to queue events. In this case, the events are stored until the event service returns and delivered after the return. The last part of the QoS requirement is optional.

Message Passing

The message passing service is required in embedded systems which have to send large amounts of data without blocking: the service can either log off the receiver or handle the messages when it is unoccupied. The message passing mechanism is always useful for loosening connections between applications; in addition, it controls the requirements concerning the quality of the services.

Clients and servers can jointly determine the QoS which both sides demand in their requests and replies.

5.2.2 Naming and trader service

Networked embedded systems require basic services for finding servers and their services from the network. The naming service is required as a basic middleware component. Commercial implementations appear to consume excessive amounts of memory and do not provide a suitable match for embedded systems. Therefore, some other implementation technique has to be found; one solution would be to combine it with an interface repository and a trader service, thereby providing one optimised component to meet all maintenance or configuration requirements.

5.2.3 Security service

If embedded middleware is used in Web applications, the security services must be supported. This requirement would be adequately fulfilled by using commercial software such as SSL, S-HTTP, or the security service included in the DCE RPC.

5.3 Optional services

Relationship service

The relationship service is required when the event service is used in implementing the publisher and subscriber mechanism. It is a necessity for configuration and maintenance services which provide support for managing large, networked systems or systems which are updated frequently.

This service is needed in systems which are designed to be adaptive.

Concurrency service

Most of the embedded systems have also real-time requirements, for which the responses are delivered as concurrent processes. Embedded systems at real-time level do not therefore necessarily require the concurrency service.

Transaction service

The transaction service is required as an optional service if the embedded systems act as servers. The transaction service is normally used by a database. Therefore, since the service is an optional feature, the most convenient way to provide it is via embedded database vendors.

6. Summary

CORBA provides a standardised way for developing interoperable distributed objects with transparent communications. When embedded systems are used to compose Internet networked systems, DCOM will similarly be a competitive technique However, the commercial products based on CORBA are still quite immature, and some services are completely absent. DCOM based OPC would appear to be a suitable solution for control systems that are based on data acquisition, monitoring and presentation, but this entails support from equipment vendors. Among industrial developers of embedded systems, these technologies do not satisfy the increasing demand for reliable embedded middleware.

Enterprises have a demand for an embedded middleware framework. Since embedded networked systems benefit from incremental building, embedded middleware has to support software integration and adaptation. Customisation should be allowed if the demands of the application domain or product families so require. If middleware software is based on components which can be added into the systems as optional features, without side-effects on the old applications, then the interoperability between distributed embedded applications, and the extendibility of the systems could be assured. As a generic (component-based software) platform, embedded middleware enables enterprises to use new operation models such as remote testing, updating and marketing.

CORBA and DCOM are based on remote procedure calls, and only a few products offer the required software packages that are needed for example in event and messaging services. These services, in addition to security and concurrency control, are essential in developing networked systems that are either Internet-based or real-time.

Software portability should be supported for applications and the embedded middleware itself. The interface technique which supports all the commonly existing component models (i.e. JavaBeans, ActiveX and CORBA), provides the support for adaptive systems. Isolation between middleware and execution environment (i.e. distinctness between operating systems and communication protocols) is a practicable approach for increasing the reuse of software. However, this would require developing a generic interface for operating systems, transport communication layers, as well as for Wireless Internet. A generic interface such as this could be used in embedded applications that set critical restrictions for reliability, memory time and response times. The first step is defining a solution which provides an optimised and balanced method for fulfilling these requirements.

References

Agranat. 1998. in http://www.agranat.com. April 1998.

Alcaltel, Hewlett-Packard Co, Lucent Technologies, Inc., Object-Oriented Concepts, Inc., Sun Microsystems, Inc. & Tri-Pacific. 1998. Realtime CORBA. Version 1.0. Initial RFP Submission. 109 p.

Allegro. 1998. in http://www.allegrosoft.com/rpproduct.html. April 1998.

Apple. 1997. in http://devworld.apple.com/mkt/informed /appledirections/jan97/stratmosaic.html. April 1998.

BEA Systems, Borland International, Expersoft Corporation, International Business Machines, International Computers, IONA Technologies, Northern Telecom, Novell, Oracle, PeerLogic, TIBCO. 1998. Joint Revised Submission for CORBA Messaging.150 p.

Blaszczak, M. 1997. Professional MFC with Visual C++5. Birmingham: Wrox Press Ltd. ISBN 1-861000-14-6.

Box, D. 1996. Introducing Distributed COM and the New OLE Features in Windows NT 4.0. Microsoft Systems Journal, Vol. 11, no. 5, pp. 19 - 38.

Braddock. 1998. in http://www.jagunet.com/~braddock/opendoccomp.html. April 1998.

Buckner, I.C.A. Penny, I.A. 1997. UKC Orbix Survival Guide. UKC Computing. May 1997.

Dibble, P. 1997. Java in Embedded Systems. Proceedings of Embedded Systems Conference. San Jose, CA, Sept. 29 - Oct.2, 1997. Pp. 55 - 65.

Digital. 1998. in http://www.digital.com/info/SP5632. April 1998.

DiLascia, P. 1998. Licence to code our man from Jakarta. http://www.microsoft.com/mind/0596/jakarta/jakarta.html. 12 p.

Expersoft. 1998. http://www.expersoft.com.

Grimes, R. 1997. Professional DCOM Programming. Birmingham: Wrox Press Ltd. ISBN 1-861000-60-X.

Highlander Communications, Visigenic Software. 1998. Realtime CORBA. Joint initial submission. 44 p.

Hildebrandt Dan. 1997. Adapting PC Technology for Internet Appliances, Embedded Systems Conference Proceedings, 416 p.

Holappa, M. 1998. CORBA:n soveltaminen joustavan valmistusjärjestelmän perusohjelmistoon. VTT Tiedotteita 1911, VTT Offsetpaino: Espoo. 87 p.

Icesoft. 1998. in http://www.icesoft.no/ICEBrowser. April 1998.

IDG-Net 1998. http://www.idg.net/idg_frames/english/content.cgi?vc=docid_0-75915.html. June 1998.

Infobus. 1998. in http://esuite.lotus.com/esuite/esuite_site.nsf /Linking+View/The+eSuite+Infobus+Defined. March 1998.

ITU. 1998. in http://www.arts.su.edu.au/Arts/departs/itu/ats/OpenDoc.html. April 1998.

JavaAPI. 1998. JavaBeans API Specification.

JavaApp. 1998. in http://www.javasoft.com/applets/index.html, March 1998.

JavaBean. 1998. in http://www.javasoft.com/beans/. March 1998.

JavaEmb. 1997. in http://www.computer-design.com/Editorial/1997/03 /Embedded/397drbeanssb.html.

JavaFAQ. 1998. in http://chatsubo.javasoft.com/current/faq.html#CORBA. April 1998.

JavaProd. 1998. in http://www.java.sun.com/products/hotjava/1.1.2./. April 1998.

Johns, P. 1996. The ACBs of MFC ActiveX Controls. MSDN CDROM July 97.

Krieger, D. & Adler, R. M. 1998. The Emergence of Distributed Component Platforms. IEEE Computer. March 1998. pp. 43-53.

Lockheed Martin Federal Systems, Inc. 1998. Realtime CORBA. Response to OMG RFP for Realtime CORBA extensions. 35 p.

Magma. 1998. in http://www.magmainfo.com/. April 1998.

Microsoft. 1995. The Component Object Model Specification. 1995. Draft Version 0.9, October 24. Microsoft Corporation and Digital Equipment Corporation.

MOMA. 1998. Message Oriented middleware Association. http://www.moma-inc.org. March 1998.

Morrison et al. 1997. Java Unleashed. Second Edition. Indianapolis, USA: Sams.net Publishing. 1164 p. ISBN 1-57521-197-1.

Mowbray, T. & Zahavi, R. 1995. The Essential CORBA. Systems Integration Using Distributed Objects. New York: John Wiley & Sons. 316 p.

Mowbray, T. 1997. Inside CORBA. Massachusetts: Addison-Wesley. 374 p. ISBN 0-201-89540-4.

Murphy, K. 1997. Windows CE in Embedded Applications. Proceedings of Embedded Systems Conference. San Jose, CA, Sept. 29 - Oct.2, 1997. pp. 987-1001.

Niemelä E. & Holappa, M. 1998. Experiences with the Use of CORBA. Proceedings of the 24th EUROMICRO Conference, Los Alamitos, CA:IEEE Comp. Soc.,Vol. II, p. 989-996, ISBN 0-8186-8646-4.

Northern Telecom & Iona Technologies. 1998. Realtime CORBA Extensions. Joint initial submission. 56 p.

Nucleus. 1998. in http://www.atinucleus.com/webserv.htm. April 1998.

Objective Interface Systems, Inc. 1998. Realtime CORBA. Initial submission. 35 p.

Omnicron. 1998. http://www.omnicron.ab.ca/httpd. March 1998.

Open Group, 1997. DCE 1.1: Remote Procedure Call. Document Number: C706. http://www.opengroup.org.

OPC Taskforce, 1996. OLE for Process Control 1.0. August 1996.

OpenDoc. 1998. in http://www.cs.umbc.edu/kqml/OpenDoc.html. April 1998.

Orfali, R, Harkey, D. & Edwards, J. 1996. The Essential Client/Server Survival Guide. Second Edition. John Wiley & Sons: New York. 676 p. ISBN 0-471-15325-7.

Orfali, R. Harkey, D. & Edwards, J. 1997. Instant CORBA. John Wiley & Sons: New York. . 292 p. ISBN 0-471-18333-4

Parker, T. 1998. Mobile Wireless Internet Technology Faces Hurdles. IEEE Computer, March 1998. pp.12-14.

Pcweek. 1998. in http://www8.zdnet.com/pcweek/
reviews/0203/03open.html./

Quiotix. 1998. in http://www.quiotix.com/wshome.html. April 1998.

Rapid. 1998. in http://www.rapidlogic.com/welcome2.html. April 1998.

RMI. 1998. in http://chatsubo.javasoft.com/current/doc/rmi-spec. April 1998.

Sheinin, Y. Emelianov, M. Ignatiev, M. Embedded Real Time Web Servers. Real-Time Magazine, March 1998. pp.84-89.

Siegel, J. 1996. CORBA Fundamentals and Programming. New York: John Wiley & Sons. 693 p. ISBN 0-471-12148-7.

SoftwareAG. 1998. http://www.softwareag.com. March 1998.

Spyglass. 1998. in http://www.spyglass.com. April 1998.

Sun 1998. http://java.sun.com. Mach 1998.

Syrodowics, S. 1997. The Remote Procedure Calls Architectures and Embedded Systems Development. Proceedings of Embedded Systems Conference. San Jose, CA, Sept. 29 - Oct. 2, 1997, pp. 1405-1420.

University of Paderborn. 1998. http://www.uni-paderborn.de/software/orbix/doc/prguide/part2/chapter1/pgintro1.html. June 1998.

Visigenic. 1998. http://www.visigenic.com/prod/vbrok/vb30DS.html. March 1998.

Wingard Steve. 1997. Embedding HTTP Functionality for Web-Based Configuration and Management of Devices, Embedded Systems Conference Proceedings, 340 p.

Yasko, C. 1997. Embedded O/S Platforms for Wireless Systems. Proceedings of Embedded Systems Conference, San Jose CA, Sept. 29 - Oct. 2, pp. 824-865.

Appendix A. OPERATING SYSTEMS FOR EMBEDDED SYSTEMS

	FLEX Client	**PalmOS**	**Windows CE 2.0**	**Inferno**	**Newton OS**	**Java OS 1.1**	**EPOC32**	**GEOS**
Type	HPC	PIM	HPC/PDA	HPC/PDA	HPC/PDA	HPC/PDA	PDA/HPC	
Provider	Motorola	U.S. Robotics	Microsoft	Lucent Technologies	Apple Computer Inc.	JavaSoft	PSion Software PLC	Geoworks Operating System
Communication	Message-based		Wireless and wired LAN, socket services, TCP/IP, PPP, SLIP, IrDA	Styx, a set of API		SLTP, PPP, SSL, TCP/IP, UDP/IP, SNMP, DNS, NIS, ICMP	full network capabilities	TCP/IP, SLIP/PPP, IrDA
OS/other support	Virtual machine/32 bit kernel	pre-emptive, multitasking,	32 bit Windows CE, multithreaded, multitasking, pocket Internet, explorer	DIS virtual machine, GUI support, security, file systems, a common network interface	Newton OS, pre-emptive, multi-tasking, OO, persistent object strore and appl. framework in ROM	Virtual machine, HotJava, JavaAPI	pre-emptive 32 bit multitasking, pen based UI	multitasking, UI: imaging, geometry. Input: keypad, pen, digital ink, handwriting recognition.
Memory	512K ROM, 256K RÀM	512K ROM, 32K RAM	1.2MB ROM, 400K RAM for CE-OS	1 MB		4MB ROM, ¤ MB RAM	4 MB ROM	

	FLEX Client	PalmOS	Windows CE 2.0	Inferno	Newton OS	Java OS 1.1	EPOC32	GEOS
CPU	68000	688328	RISC CPUs, Hitachi SH-3, MIPS (NEC, Philips), Future: PowerPC, ARM, x86	x86, MIPS, ARM	Digital StrongARM, Cirrus logic ARM	SPARC, x86, StrongARM	Cirrus 7100, Digital Strong ARM	
Language	Script Language like C++		C++, additionals: ActiveX/COM/OLE, Java for Windows CE	C-based Limbo, multithreaded, garbage collection	Newton Script language	Java	C++, OPL	C++
Development	Windows/IDE	Windows, Mac/SDK	Visual Studio IDE, Visual C++, a subset of MFC and Win32 API, Embedded Toolkit for VisualC++	UNIX, WIN95, NT		JDK 1.1.4	PC/Windows, VisualC++	Windows NT/Borland C++
Applications	Motorola PageWriter 2000	PalmPilot with 2500 addresses, handwriting	Several HPCs from Casio, Compaq, NEC, Philips, Hewlett-Packard, Hitachi		Apple's MessagePad 2000 (LCD, 8MB ROM 5 MB RAM)	Products from Mitsubishi, Telxon, Blazie	Sharp Zaurus	HP's OmniGo 200

HPC = Handheld Personal Computer (web browser, spreadsheet, word processor)
PDA = Personal Digital Assistant (email, messaging)
PIM = Personal Information Manager (calendar, contacts, to-do lists)
(Yasko, 1997; Murphy 1997).

Appendix B. Third Party Java Virtual Machines / JDKs

Operating System	CPU	Description	Company / Organization
AIX	PowerPC	JVM	Kaffe.Org
AmigaOS	M68K	JVM	Kaffe.Org
BeOS		JDK	
BSDI	i386	JVM	Kaffe.Org
DG/UX	i386	JVM	Kaffe.Org
DIGITAL OpenVMS	Alpha	JDK	Digital Equipment Corporation
DIGITAL Unix	Alpha	JDK	Digital Equipment Corporation
DIGITAL Unix	Alpha	JVM	Vienna University of Technology
EPOC 16	Psion Series 3	JVM	
FreeBSD	i386	JDK	Bluegum Software Specialists Inc., The FreeBSD Project
FreeBSD	i386	JVM	Kaffe.Org, The Hungry Programmers
HP-UX		JDK	Hewlett-Packard
HP-UX		JVM	Hewlett-Packard
IRIX	MIPS	JVM	Kaffe.Org
Linux	MkLinux, PowerMac, PowerPC	JDK	
Linux	PowerPC, Alpha, i386, Sparc	JVM	Kaffe.Org

Operating System	CPU	Description	Company / Organization
Linux	Alpha	JVM	Vienna University of Technology
Linux	i386	JVM	The Hungry Programmers
MachTen	PowerPC	JVM	Kaffe.Org
MacOS	Macintosh	JDK	Apple
NetBSD	i386, Sparc, M68K, MIPS	JVM	Kaffe.Org
NetBSD	i386	JDK	Quick.com.au
NetWare		JVM	Novell
NeXTStep	i386, Sparc, M68K	JVM	Kaffe.Org
OpenBSD	i386	JVM	Kaffe.Org
OS/2	i386	JDK	IBM
OSF/1	Alpha	JVM	Kaffe.Org
Reliant Unix	MIPS	JDK	Siemens Nixdorf)Informationssysteme AG
RiscOS		JVM	
SCO	i386	JVM	Kaffe.Org
Solaris	Sparc, i386	JDK	Sun Microsystems
Solaris	Sparc, i386	JVM	Kaffe.Org
SunOS	Sparc	JDK	
SunOS	Sparc, M68K	JVM	Kaffe.Org
UnixWare	i386	JVM	Kaffe.Org
UXP/DS	Sparc	JVM	Kaffe.Org
VxWorks		JVM	Wind River Systems
Windows 95/NT	i386	JVM	Kaffe.Org, Mach J Col
Windows CE	H/PC	SDK for Java	Microsoft

Appendix C. HPC PRODUCTS WITH WINDOWS CE 2.0

Feature	Casio Computer Co.	Compaq Computer Co.	Hitachi Ltd.	Ericsson Mobile Computing	Hewlett-Packard Co.	LG Electronics	NEC Computer Systems Division	Philips Electronics
Model	Cassiopeia E-10	2010C	HPW-200EC	MC 12	HP 360LX	Phenom	Mobile Pro 750C	VELO 500 (16/24 MB)
Processor Speed	NEC VR4111 MIPS RISC processor	75 MHz, RISC processor	100 MHz, SH3 32-bit RISC chip		Hitachi	100 MHz Hitachi SH3 RISC chip	80MHz NEC VR4111 processor	75 MHz 32-bit MIPS-based Philips PR31700 RISC processor
Display	240x320 LCD screen , backlight	Resolution: 640x240 Type: STN LCD	640x240 LCD, 256 colours	640x240 touch screen with backlight	640x240, 16-grayscale (colours in 620LX; 1998))	256 colours, 640x240 backlight screen, 1/2 VGA display	256 colours	grayscale LDC, 640x240, backlight
Memory (RAM/ROM)	4 MB/	20 MB/ 16 MB	16 or 32 MB/12 MB	4 MB/5 MB, 2MB Compact Flash	8 MB/10 MB	16 MB (expandable)/12 MB	16 MB/16 MB	16 MB (expandable)/16 MB , 8MB DRAM miniature card in 24 MB Velo 500

Feature	Casio Computer Co.	Compaq Computer Co.	Hitachi Ltd.	Ericsson Mobile Computing	Hewlett-Packard Co.	LG Electronics	NEC Computer Systems Division	Philips Electronics
Interfaces	Compact flash card slot, serial port, IrDA port, built-in microphone and speaker, automatic data synchronisation with a desktop PC (up-to-date)., Options: modem adapter, compact flash card, AC adapter	33.6 Kbps integrated modem, integrated RJ11, serial port 115KB/s, PC card slot, Infrared port 115Kb/s, speaker	33.6 Kbps modem, Type II PCMCIA slot, serial port, Compact flash card slot, built-in microphone, speaker, and voice recorder	IrDA port, PC-card slot type II, drag&drop between PC &MC 12	Compact flash card slot, PC card slot for modem	33.6 Kbps modem, built-in VGA port, PCMCIA type II slot, compact flash slot	Compact flash and PC type II card slots, VGA port, built-in modem, microphone, speaker, infrared support	61 keys, 10 quick start-keys., built-in microphone, speaker, 28.8 Kbps V.34 data/fax modem. IrDA (115 Kbps), RS-232 (230 Kbps)
Power Supply	2 AAA-size alkaline batteries, back-up battery CR2016	NiMH rechargeable batteries, Lithium backup battery, A/C adapter	rechargeable Li-ions Battery Pack			rechargeable Li-ions /10 hours		NiMH rechargeable battery pack

Feature	Casio Computer Co.	Compaq Computer Co.	Hitachi Ltd.	Ericsson Mobile Computing	Hewlett-Packard Co.	LG Electronics	NEC Computer Systems Division	Philips Electronics
Windows CE Applications	Microsoft Outlook with ActiveSync, Note taker, voice recorder, e-mail, inbox, Internet explorer 4.0	Windows CE services with ActiveSync, Pocket Outlook, Pocket Internet Explorer, Pocket Word 2.0, Pocket PowerPoint 2.0, Visual Basic 5.o Runtime, Visual C++ runtimes, Pocket Street 2.0	Pocket Outlook, Pocket Word, Pocket Excel, Pocket PowerPoint, Pocket Internet Explorer	Internet Explorer, Inbox, Pocket Word, Excel; Calendar, Contacts and Tasks (autosync. with M. Schedule & Outlook 97)	PowerPoint	HPC: Pocket Word, PowerPoint Viewer, Excel, Internet Explorer, Outlook, PC: CE services	Sync. Schedule+ 7.0/Exchange, Lotus Organizer, ACT! (Calendar, contacts, tasks, inbox, calculator, world clock), Pocket Word, Excel, Power Point, Internet Explorer	Pocket Word, Excel, PowerPoint Viewer, Internet Explorer, Microsoft Pocket Outlook , e-mail, sync. CE service with ActiveSync.