

Technology Briefing Report

Middleware

RENAISSANCE Esprit Project 22010

Identifier	D5.1b Middleware
Type	Internal result
Activity	WP5.1b
Date	31 st July 1996
Status	Final
Partner	SINTEF
Availability	Restricted

*To help solve customer's heterogeneity and distribution problems, and thereby enable the implementation of an information utility, vendors are offering distributed system services that have standard programming interfaces and protocols. These services are called **middleware services**, because they sit «in the middle», in a layer above the OS and networking software and below industry-specific applications.*

(Philip A. Bernstein, Communication of the ACM, February 1996 «Middleware: A Model for Distributed System Services»)

1. Overview

(Description of the technology in general terms.)

Middleware is, to some extent, found in every client/server environment. Whenever a client sends a request to a server or an application to download data from a database, some form of middleware is involved: mediating the client/server link and smoothing out the potential incompatibilities between communications protocols, database query languages, application logic, and hardware operating systems.

I think we can divide existing middleware into seven different categories of services provided:

1. **Data Management Services** (i.e. Database and file system middleware).
2. **Communication Services** (i.e. **RPC** (Remote Procedure Call) and **messaging** middleware).
3. **Distribution Services** (i.e. location, time and security services).
4. **Object Management Services** (i.e. by using Object Request Brokers (**ORBs**)).
5. **Application Co-operation Services** (i.e. Transaction-Processing (**TP**) monitors, e-mail, etc.).
6. **Presentation Services** (i.e. User Interfaces, printing and multi-media middleware).
7. **System Management Services** (i.e. Configuration-, change-, operations-, problem-, and performance-management services).

1.1 Data Management Services

These services provide access to files and databases distributed over the network and basic management functions to maintain their operational integrity and application availability. **File** middleware should provide users transparent query and update access to one or more file systems that are located in a client/server system. **Database** middleware, as the name implies, has a restricted application. Most client/server systems have an application on the client that accesses data in the server's database. If the application is designed to use only one database type, and is likely to remain that way, then a database middleware strategy may be a good idea. See also the *Technology Briefing Report on Databases*.

1.2 Communication Services

These services all rely on the same basic, low level transport protocols. We have two different ways of routing messages in a communication based system. One is peer-to-peer messaging which is used by remote procedure call (RPC). The second is routing by using a Broadcast Message Server (BMS). Synchronous **messaging** delivers a message from its source to its destination(s) in real-time. Asynchronous **messaging** (using queues) enable distributed applications to send and receive messages using a relatively simple set of APIs. **RPC** based middleware allows applications to access data and functions on remote servers much the same way as applications accessing a local database or function. Messaging and message queuing require the application developer to deal with the message data, while RPCs handle the data automatically.

1.3 Distribution Services

These services enable, in conjunction with communication services, all networked applications and services to be viewed as a single system. **Location** services consist of a directory and a naming service. The directory service maintains physical and static information about the characteristics of network resources (e.g. servers, files, disks, applications etc.) such as its name, network address, and creation date. The name service enables the network resource to be referenced dynamically, at run-time, even if names or locations change.

1.4 Object Management Services

These services should provide a first order of co-ordination between objects. Objects may use a mechanism called an Object Request Broker (ORB) to transparently make requests to, and receive responses from, other objects located locally or remotely. While the data management services co-ordinate data, object management services co-ordinate objects. An object contains, in addition to data, also methods that operate on the data. This emphasises distributed computations.

1.5 Application Co-operation Services

These services provides high-level application oriented functions significantly reducing the amount of intelligence that must be built into the distributed applications and services. Application co-operation services are built on communication services. **Transaction-Processing Monitors** provides an API for development of transaction applications and support for effective execution. The main function of TP monitors are to co-ordinate the flow of requests between terminals (or other devices) and application programs that can process these requests. **E-mail** can be used not only for exchanging text messages, but also to send graphics, images, voice, unstructured data and files (i.e. binary code), etc.

1.6 Presentation Services

These services provide visual facilities for end-users. **User Interface** services support the communication session management between the user and one or more of the distributed applications and services, as well as the visual user interface for character, graphics or/and image displays. **Print** services facilitate print job submission and status etc. **Multi-media** services allow the display, manipulation, control and retrieval of distributed graphical, audio, video and animation data.

1.7 System Management Services

System management services enable the continuous monitoring of application and system functions to ensure that a corporation's business requirements can realise an optimum quality of service from the distributed application environment.

Not all available middleware fall into only one of these categories, i.e. some middleware may be a combination of several middleware services.

2. Role in evolution process

(The part played by that technology in the general evolution process.)

The middleware technologies plays an important role in the re-engineering/evolution process. Migrating from legacy systems often involves a careful step by step process ensuring the performance and reliability of the resulting system. Middleware helps to make this possible by providing gateways between new and old software components, User Interfaces (UIs), and between new applications and old databases.

The selection of middleware technologies are generally performed after obtaining a satisfying system model of the revised application. But you have to ensure, during the design phase, that the supporting middleware tools exists and are applicable. Re-design can be very time consuming and expensive.

The middleware services that needs to be supported may be discovered directly from the system and architecture models. However, this is not always the case. Some middleware services also depends on other services to perform their tasks (i.e. application co-operation services relies on communication services).

The middleware selection process has to be done early in the implementation phase to avoid surprises regarding tool and platform supports. One way of selecting middleware is to follow a seven step decision process (proposed by the Patricia Seybold Group):

1. Selecting the transport protocol to be used.
2. Selecting a pilot application representing the most important class of applications the organisation will need for the near future.
3. Selecting an enabling service - the server-based code that supports clients with shared applications.
4. Selecting a category of middleware by determining its primary purpose, characteristics, and practical uses.
5. Selecting the middleware products(s).
6. Selecting application development tools (ORBs, RPCs, etc.) based on the usual quality measures.
7. Selecting middleware management tools to monitor, control, and administer the environment.

3. Relevance to Renaissance

(The specific relevance of the technology to Renaissance and how it might be part of the Renaissance method.)

Middleware technologies will ease the step-by-step process of migrating from a centralised legacy system to a client/server system by offering tools to help an incremental evolution process (incremental rewriting). Using middleware, small parts of the system can be re-engineered and rewritten without affecting the rest of the system.

Middleware offers the possibility to develop/implement the client application(s) in a client/server environment independent from the server application(s). This way one can avoid critical downtime of systems during the development of new client applications. Applying middleware helps making loosely coupled applications.

Using middleware allows client and server applications to be sited on different hardware and software platforms. This means that old hardware and software can be reused and combined with new technologies.

4. Available support

(A discussion of available tool support with examples if appropriate.)

There are many middleware tools supporting the different services mentioned in the **overview** chapter.

4.1 Data Management Services

Database middleware is software designed to cope with the challenge of providing universal uniform data access to all clients in a client/server environment. The available tool support for database middleware are wide. All the large database providers provide database middleware connecting their own, and sometimes other databases, to a range of platforms. But also many third part vendors make database middleware. Most database middleware provides several connection protocols, SQL translation, and a structured development language (for centralising tasks on server, such as updatings, complex data-retrievals, etc.). Many also supports some file services like file transfers between different platforms, reading remote files, etc.

One **example** of a middleware supporting these features are **CI-Link** from **Cornut Informatique** in France. CI-Link is a middleware that connects MS-Windows and Macintosh applications to SQL databases under UNIX. It comes with a high-level built-in development language which lets you centralise tasks on the server. CI-Link is compatible with the several relational DBMSs like Oracle, Informix, Ingres and Progress.

Another database middleware candidate is **Accessworks** from **Oracle Corporation**, in co-operation with **Digital**. Accessworks provides a desktop API via ODBC, SQL services, and read/write access to a wide area of databases, etc. Accessworks is available for the most common client desktops such as MS-DOS, MS-Windows, OS/2, Macintosh, Open VMS, and UNIX systems.

Most existing data management middleware supports high level programming languages and desktop utilities to help the development process.

4.2 Communication Services

Middleware supporting the **communication** services can be divided into two different groups: **RPC** and **messaging** systems. **RPCs** provide facilities for synchronous procedure calls on remote systems as if they were local. The tools supporting **RPCs** includes an Interface Definition Language (IDL), a compiler, a Universal Unique Identifier (UUID) generator, and a **RPC** runtime library. **RPCs** supports connectionless and connection-oriented transports. The IDL and its compiler describes the **RPC's** interface. The compiled code (the object code) is in two main parts - one for the client side, and one for the server side. The **RPC** runtime library consists of a set of routines, linked with both the client and server sides of an application, which implements the communications between them.

The most famous **RPC** toolset is **DCE** (Distributed Computing Environment) from **OSF**, but also available from most UNIX hardware vendors. **DCE** is a set of services that support the development, use and maintenance of distributed applications. **DCE** allows diverse systems to work together co-operatively and masks the technical complexities of the network. Because it is independent of the operating system and network, it is compatible with many diverse environments currently in place by users.

Messaging middleware enables distributed applications to send and receive messages asynchronously using a set of APIs. The messaging middleware relieves the applications of message routing, queuing, and session establishment and termination. A good messaging middleware also is independent of platforms and can operate between applications on different platforms. One **example** of a such a middleware API is **Communications Integrator** from **Covia Technologies**.

4.3 Distribution Services

The **distribution** services can be divided into three categories: location services, security services, and time services. Tools supporting **location** services provide a directory and naming service. The directory service allows a client to access a remote file, a remote table, or a remote process without having to know where that object is physically located in the network. The tools supporting location services eases the connection to servers by hiding network details and providing naming services. We don't have to walk around remembering server addresses and paths. Using the Internet and/or an intranet infers a security problem. **Security** middleware are aimed at protecting the internal systems. The security middleware is installed between the server, the application and remote users, and provides secure transactions independent of client and server applications.

4.4 Object Management Services

Tools supporting **object management** services use Object Request Brokers (ORBs) to provide transparent communications between objects sited on different locations in a network. **OMG's** **CORBA** is an example of this technology (further discussed in the *Technology Briefing Report on Distributed Object Technology*). **IONA's** **ORBIX** is the most famous implementation of **CORBA**. **OLE/COM** is another example that is implemented by **Microsoft**. In addition to provide communication between objects, **CORBA** middleware also provide communication with the **WWW**, **RDBMs** and **ODBMs**.

4.5 Application Co-operation Services

Tools supporting **application co-operation** services are providing support for a large number of concurrent users that access transactions programs and services (i.e. databases, security, workflows), local and distributed load balancing to optimise performance, and efficiently synchronising data updates to multiple databases during a single transaction using standard protocols. **Tuxedo** from **Novell** is an **example** of a TP monitor providing these features plus several others. The advantage by using TP monitors are their ability to manage several thousands of users, concurrent database accesses, and large volumes of data.

We also have workflow service which allows the developer to control scripts to manage the execution of sequences of operations that relate to particular business processes. The workflow service makes distributed applications serve the business process which avoids modifying the application if the business process changes as is seen with managing document workflow. Workflow services using transaction monitor services, is called **transaction workflow**.

4.6 Presentation Services

Tools supporting **presentation** services provide features for mapping GUIs and transparent printing. These services are often included in other tools. But there exists dedicated tools for mapping character based UIs into GUIs. **X Windows** is an example of a toolset providing presentation services.

4.7 System Management Services

Tools supporting **system management** services should provide configuration management, change management, operations management, problem management, and performance management. These services, although not fully available, enable applications and system functions to be continuously monitored to ensure that a corporation's business requirements can realise an optimum quality of service from the distributed application environment.

As mentioned, all existing middleware do not fall into these seven categories of middleware. Most tools provides two or more kinds of services. For example: tools supporting data management services often also supports security services, location services, and time services.

5. Maturity assessment

(An assessment of the maturity of the technology.)

The maturity level of the middleware varies accordingly to the type of services they are meant to support.

Services	Maturity
Data Management	Medium
Communication	Medium/High
Distribution	Medium/High
Object Management	Low/Medium
Application Co-operation	Medium/High
Presentation	Medium
System Management	Low

Table 1 Maturity assessment.

We also have combined architectures that assemble data management, communication, application co-operation, distribution, and object management services, but these are not yet very mature.

6. Inapplicability

(Areas where the technology should NOT be applied (e.g. 4GLs for real-time systems))

Middleware should be used carefully in systems that are demanding real-time responses, as heterogeneous platforms connected via middleware may slow down the system.

Also immature middleware should not be used in systems where the data security is of high importance. There have been found very serious security holes in new technologies such as Java. Middleware should be used with care in systems connected to the Internet and that contains confidential data!

Commercial available middleware may also be inapplicable on legacy systems using their own database formats (like indexed files). In such cases one need to write dedicated middleware that translates the legacy formats into a format known by existing middleware. Newer middleware such as different CORBA implementations may also be incompatible (or at least very hard to use) with legacy database systems like for instance IBM's IMS.

7. Future development

(An assessment of possible future developments of the technology.)

The future development of middleware technologies will probably mostly be focusing on object management, database middleware, and combined architectures. Object management and combined architectures are both very new technologies and not very mature yet. But as more and more systems go from non-object-oriented (i.e. centralised legacy systems) to object-oriented architectures (i.e. client/server systems), we need supporting middleware. CORBA and OLE/COM are examples of the distributed object management technology.

New opportunities have also appeared with World Wide Web (WWW) technologies. Combining existing multi-platform browsers/viewers with legacy systems can be an easy and quick way of migrating to client/server architectures. Middleware such as Java from Sun can be used to connect the client application with the server. This topic is further discussed in the briefing reports on Internet and distributed object technologies.

There is also a trend towards more Intelligent Middleware (IM). IM doesn't replace traditional middleware; it simply hides the complexity of middleware by providing a protective layer between the application programs and the underlying infrastructure technology. IM consists of a set of tools and runtime services which increases the robustness of the operating environment. IM allows you to develop applications independent from the actual underlying middleware. By eliminating details and enforcing standard notation, this will make the development process faster and more reliable.

8. Other comments

(Other comments regarding middleware).

Although middleware interfaces generally are aimed at building new client/server applications, they play an important role in connecting existing applications to new ones - like when migrating from a legacy system to a client/server architecture. Traditionally, organisations have used a simple point-to-point client/server approach to implement middleware connections. But as the number of clients and servers increase, the number of connections can increase exponentially. This can lead to performance bottlenecks and complicated systems that are a nightmare to administrate. From the IS professional's point of view, middleware has therefore had a reputation for being complicated, hard-to-use, and arduous to maintain. This is some of the things we want to get away from when migrating from a legacy system - middleware can (if used wrong) work against it's purpose.

This report has so far only described the positive effects of using middleware. The disadvantages/uncertainties can be listed shortly as:

- The technology is rapidly changing.
- There are few people with experience in the marketplace.
- There exists relatively few satisfying standards.
- The tools are not good enough.
- Too many platforms to be covered.
- Middleware often threatens the real-time performance of a system.
- Middleware products are not very mature.

The conclusion has to be that middleware technologies are very important to succeed with an incremental migration of legacy systems into client/server systems, but that it is hard to find the correct tools and people to accomplish the work. It is a need for better processes.