Editorial Thoughtscapes
Words Impact Business Success
www.et-writer.com

B2B Enterprise Technology Story

Web Summary:

Software in the Middle

by Leon A. Enriquez

Reading Time:
28 minutes

Reader Benefit:
♦Gain insights about "software in the middle" or middleware;
♦Understanding the fundamental ideas surrounding software known generically as middleware;
♦Knowing how middleware enables different software applcations and platforms to co-exist and work together.

Middleware has emerged as a critical second level of the enterprise IT infrastructure. The need for middleware stems from growth in the number of applications, in the customisations within those applications and in the number of locations in our computing environments.

Middleware is a layer of software "glue" that interacts between the network and the applications. This software acts as a plumbing and provides services such as identification, authentication, authorisation, directories, and security.

In today's Internet context, applications usually have to provide these services themselves and this situation leads to competing and incompatible standards. For instance, middleware that promotes standardisation and interoperability will make advanced network applications much easier to use and deploy.

These factors now require a set of core data and services to be moved from multiple instances into a centralised organisational offering. This central provision of services eases application development, increases robustness, assists data management, and provides overall operating efficiencies.

Software in the Middle

by Leon A. Enriquez

Middleware has emerged as a critical second level of the enterprise IT infrastructure. The need for middleware stems from growth in the number of applications, in the customisations within those applications and in the number of locations in our computing environments.

These factors now require a set of core data and services to be moved from multiple instances into a centralised organisational offering. This central provision of services eases application development, increases robustness, assists data management, and provides overall operating efficiencies.

Middleware is a layer of software “glue” that interacts between the network and the applications. This software acts as a plumbing and provides services such as identification, authentication, authorisation, directories, and security. In today’s Internet context, applications usually have to provide these services themselves and this situation leads to competing and incompatible standards. For instance, middleware that promotes standardisation and interoperability will make advanced network applications much easier to use and deploy.

Thus, middleware is rightly the distributed systems and mobile systems infrastructure that make concepts such as global mobility, ubiquitous personalisation and adaptable computing technical realities. Basically, middleware bridges the gap between modern communications-enabled operating systems and the thousands of third-party applications.

The term middleware is used to describe separate products that serve as the glue between two applications. It is, therefore, distinct from import and export features that may be built into one of the applications. Middleware is sometimes called plumbing because it connects two sides of an application and passes data between them.

Consider the case of developing middleware platforms, deployment frameworks and test-bed of distributed and mobile applications. Such middleware projects make the world come alive around you where technology is no longer your enemy. As you move from location to location, your computing world can be mapped automatically onto the facilities available.

For instance, doctors going round a hospital can always monitor a critical patient’s vital signs on the nearest terminal without the need to log on, identify themselves, and start up applications. This is because the middleware knows where they are, what they need help with, and configures the world around them. Or consider the case of a lecturer who can always see the locations of students within a campus, thanks to intelligent middleware.

Middleware Functions
Put another way, middleware is the systems software that bridges the gulf between modern communications-enabled operating systems and crucial user applications. Middleware functions can been divided into the following areas:

1. *Platform:* The technology to construct, distribute and mobilise components. Examples of such construction and support platforms are: distributed object platforms such as CORBA, DCOM or RMI. These provide facilities for building object-oriented distributed systems.
2. *Mobile agent systems:* Which provide facilities to build agents that can be transmitted from site to site. Such agents can act on the user’s behalf with direct intervention and can get information, broker deals, negotiate based on set guidelines, etc.
3. *The Web:* Some favour the Web as a structure for building applications around. Such Web-enabled applications allow users to gain access to information without the contraints of being location- or desk-bound in the computing environment.
4. *Frameworks:* Using a platform to construct a framework of components, designed to support a particular range of applications. Examples include platforms to support: multimedia applications frameworks of objects, representing components of multimedia applications such as cameras and video windows can be plugged together to create on-the-fly applications.
5. *Groupware Applications:* Designed to support generically group activities, such as joining and leaving a meeting, and coordinating tools such as shared drawing e.g., executed in AutoDesk with file-sharing capabilities.
6. *Services:* Focussing in on specific services needed in a middleware system. Examples include database and transaction processing systems, and systems for distributed systems management.

Middleware function as connectivity software that consists of a set of enabling services – which allow multiple processes to run on one or more computers to interact across a network. Middleware is essential to migrating mainframe applications to client/server applications, or even Web-enabled applications and provides for communication across heterogeneous platforms.

Such middleware technologies have evolved during the 1990s to provide for interoperability in support of the move to client/server architectures. The most widely-publicised middleware initiatives are:
♦ Open Software Foundation’s Distributed Computing Environment (DCE);
♦ Object Management Group’s Common Object Request Broker Architecture (CORBA); and
♦ Microsoft’s COM/DCOM (Component Object Model or COM, and Distributed Component Object Model or DCOM).

Middleware also acts as a software that connects two otherwise separate applications. For example, there are a number of middleware products that link a database system to a Web server. This allows users to request data from the database using forms displayed on a Web browser, and it enables the Web server to return dynamic Web pages based on the user’s requests and profile.

Core Middleware
Core middleware services are those that all other middleware services depend on. The challenges in providing these services are as much political as they are technical. Many of the hardest issues involve the ownership and management of data in the complex world of business.

As a whole, there are five main services that are central to middleware, namely:

1. *Identifiers:* A set of computer-readable codes that uniquely specify a subject.
2. *Authentication:* The process of a subject electronically establishing that it is, in fact, the subject associated with a particular identity.
3. *Directories:* Central repositories that hold information and data associated with identities. These repositories are accessed by people and by applications to, e.g., get information, customise generic environments to individual preferences, and route mail and documents.
4. *Authorisation:* Those permissions and workflow engines that drive transaction handling, administrative applications and automation of business processes.
5. *Certificates and public key infrastructures (PKI):* Certificates and PKI are related to the previous four core middleware services in several important ways.

Some services, like authentication and directories, are in all categorisations. Others, such as co-scheduling of networked resources, secure multicast, and object brokering and messaging, are the major middleware interests of particular communities, but attract little interest outside of those particular communities.

Identifiers. An identifier is a character string that connects a real-world subject to a set of computerised data. Identifiers were simple when each person had exactly one. Now people generally have several identifiers, and identifiers apply not only to people, but also to group of people, or to objects (or groups of objects) such as printers and applications. Thus the relationships among a subject’s identifiers, and policies associated with the assignment of identifiers, become important issues.

Authentication. Given the breadth of interactions that are now computer-assisted, establishing that a particular request is associated with a specific real-world subject becomes critical. The traditional approach of login and clear text password is far too insecure and inflexible for the variety of ways that clients need to authenticate to servers.

Directories. Much of the information about real-world subjects needs to be contained in a general-purpose, high-performance server that can respond to application requests for information. There are substantial technical and political issues in the development and operation of a directory service. Technically, determination of the elements of the directory (the schema), the ways of addressing the elements (the namespace), and operational issues such as replication and partitioning need to be addressed. Applications must be reengineered to use the directory. Policy issues include ownership of data, feeds into and out of the directory, and setting permissions to read and write data.

Authorisation. An important subset of the information about a real world subject is what it is permitted to do. Authorisation can range from allowing access to refined controls of a remote electron microscope to permissions to place purchase orders below a specified level on an institutional account. Defining these rules, including means to delegate or reassign authority on a temporary basis, as well as delivering this information to applications, are some of the challenges in this newly emergent area.

Certificates and PKI. The software, protocols and legal agreements that are necessary to effectively use certificates combine to form a Public Key Infrastructure (PKI). A PKI has several components. A Certificate Authority (CA), that manages and signs certificates for an institution Registration Authorities, operating under the auspices of the CA, that validate users as having been issued certificates. PKI management tools, including software to manage revocations, validations and renewals directories to store certificates, public keys, and certificate management information databases and key-management software to store escrowed and archived keys.

Applications can make use of certificates and can seek validation of others' certificates. Trust models that extend the realm of secure communications beyond the original CA policies that identify how an institution manages certificates, including legal liabilities and limitations, standards on contents of certificates, and actual campus practices.

Among the potential uses for certificates are individual authentication, email encryption, digital signatures, and access controls. Each of these uses can place different requirements on the PKI components. For example, private keys for encryption may be escrowed, while private keys for signatures may not be.

Below the core middleware services, at the boundary of the network layer, lie a number of services that can be classified as middleware-based networking or networking-oriented middleware. These services include:

- *Secure multicast.* This is multicast extended to permit, at the network layer, secure access to join a multicast session.
- *Bandwidth brokering.* This is a service that securely allocates quality of service (QoS) to various applications and users within an institution or organisation.

Typically these services require core middleware services, such as identifiers, authentication and directories, in order to operate.

“Upperware”
Above the core middleware services are a number of types of application-oriented middleware, or “upperware.” A rough grouping of such middleware known as upperware would include:

- *Services for ubiquitous computing.* Organisations need a variety of open protocols and implementations that allow students to access their bookmarks and aliases from any location, as well as institutional and multi-organisational file systems to enable sharing and support collaboration tools.
- *Support for research computing.* Efforts are underway to transform scattered national computational resources into a coherent grid, providing researchers consistent access across a variety of architectures, permitting co-scheduling of resources, coupling data, networking and computing together.
- *Support for administrative computing.* The new generations of business systems have loosely-coupled components that depend on a common applications infrastructure, which provides services such as object brokering for component requests, message handling between components, and monitoring of transactions.

Again, these services depend on core middleware components in order to operate. In turn, as these areas continue to evolve rapidly over the next few years, new utilities may be developed within the core to support them.

There are numerous services that applications would like to have provided for them, rather than having to perform these functions themselves. These “upperware” services can be grouped into three categories:

1. *Business application middleware:* Typically businesses are seeking extended tool sets with which to construct modular accounting or transactional applications. Four services are frequently mentioned in this category: object resource brokering (to find business data in distributed environments), message handling (both asynchronous and synchronously between processes), transaction monitoring (to perform audit functions), and application gateways (such as X.500 to SMTP). These tools are being developed by major computer companies and will find their way into administrative systems in higher education. Good references to business application middleware abound on the network.

2. *Research application middleware:* Complex networked computing environments are being developed to support the acquisition, processing and management of scientific data in situations unique, high-end resources are required. Supercomputers, advanced networks, network storage systems, and specialised scientific instruments are being connected together into fabrics to address particular research agendas. To manage this environment requires a number of advanced services such as coscheduling of networked resources, bandwidth brokers, library synchronisation, and database discovery.

3. *Ubiquitous Computing Tools:* To achieve “anywhere, anytime, anyhow” computing, users will need to be presented with a consistent, customised interface while employing a variety of devices from a variety of locations. While such mobility, portability and ubiquity will depend heavily on authentication and directory services, additional protocols and APIs need to be established. For example, standard sets of data will need to be moved frequently and securely between devices and centralised directories.

Mechanisms are also needed to change attribute preferences depending on the characteristics of the device being used.

Middleware Performance
There is a need to note some performance guidelines for selecting middleware. It seems simple. If middleware is nothing more than a layer above the existing network, than all middleware should perform the same way. This however is not the case because of the complexities involved.

To begin with, middleware technology has grown in many different directions. The results are obviously the many trade-offs when considering middleware performance.

Typically, software development project leaders select middleware without considering performance. This kind of myopia in selection criteria can lead to disaster, including failed projects, upset users, or both.

Selecting the right middleware for your distributed application has a major drawback. It’s difficult to determine performance until the application is up and running. By that time, you have to live with the consequences as the damage has been done.

However, some performance guidelines for selecting middleware can be provided. In short, what’s best is not always fastest, and what’s fastest is not always best. The trick is to find the best and the fastest.

Middleware selection is all about design. Application architects should rightly assess and select a middleware layer based on matching the right solution to the right problem. You must consider the application requirements. This ensures that the right middleware architecture is selected to meet the optimal performance requirements. You need to be diligent here.

Middleware Services
Middleware services are sets of distributed software that exist between the application and the operating system and network services on a system node in the network.

Middleware services provide a more functional set of Application Programming Interfaces (API) than the operating system and network services – to allow an application to locate transparently across the network, providing interaction with another application or service. Such services need to be independent from network services, be reliable and available, and scale up in capacity without losing function.

Common middleware categories can take on the following different forms which include:
- ♦ TP monitors;
- ♦ DCE environments;
- ♦ RPC systems;
- ♦ Object Request Brokers (ORBs);
- ♦ Database access systems; and
- ♦ Message Passing.

Transaction processing (TP) monitors, which provide tools and an environment for developing and deploying distributed applications.

Remote Procedure Call (RPCs), which enable the logic of an application to be distributed across the network. Program logic on remote systems can be executed as simply as calling a local routine.

Message-Oriented Middleware (MOM), which provides program-to-program data exchange, enabling the creation of distributed applications. MOM is analogous to email in the sense it is asynchronous and requires the recipients of messages to interpret their meaning and to take appropriate action.

Object Request Brokers (ORBs), which enable the objects that comprise an application to be distributed and shared across heterogeneous networks.

The main purpose of middleware services is to help solve many application connectivity and interoperability problems. However, middleware services are not a miracle cure if there is a gap between principles and practice. For example, many popular middleware services use proprietary implementations thus making applications dependent on a single vendor's product.

Again, the sheer number of middleware services causes confusion and may be a barrier to using them. So, to keep their computing environment manageably simple, developers have to select a small number of services that meet their needs for functionality and platform coverage.

While middleware services raise the level of abstraction of programming distributed applications, they still leave the application developer with hard design choices. For example, the developer must still decide what functionality to put on the client and the server sides of a distributed application.

The key to overcoming these problems is to fully understand both the application problem and the value of middleware services that can enable the distributed application.

To determine the types of middleware services required, the developer must identify the functions required, which fall into one of three classes:

1. *Distributed system services,* which include critical communications, program-to-program, and data management services. This type of service includes RPCs, MOMs and ORBs.

2. *Application enabling services,* which give applications access to distributed services and the underlying network. This type of services includes transaction monitors and database services such as Structured Query Language (SQL).

3. *Middleware management services,* which enable applications and system functions to be continuously monitored to ensure optimum performance of the distributed environment.

A significant number of middleware services and vendors already exist. It is likely that middleware applications will continue to proliferate with the installation of more heterogeneous networks.

The costs of using middleware technology, e.g., license fees, in system development are entirely dependent on the required operating systems and the types of platforms. Middleware product implementations are unique to the vendor. This results in a dependence on the vendor for maintenance support and future enhancements. This reliance may have a negative effect on a system's flexibility and maintainability.

However, when evaluated against the cost of developing a unique middleware solution, the system developer and maintainer may view the potential negative effect as acceptable in view of the situational constraints such as budget, deployment time, and other factors.

Middleware Pros and Cons
Middleware has many advantages, including the following:

1. Hiding platform software and hardware details from end users;
2. Making application distribution transparent to users;
3. Providing portability, interoperability, flexibility, and scalability;
4. Enabling new applications that use its services;
5. Identifying several important components that can be used and shared across many environments; and
6. Helping handle legacy applications.

However, middleware also presents unique problems which include:

- *Maturity:* The technology is not mature enough to use in many mission-critical projects;
- *Standardisation:* Many services use proprietary protocols and application programming interfaces (APIs) and are not based on standards;
- *Cost:* Middleware adds to the cost of the environment;
- *Performance:* Middleware may cause a performance degradation in bandwidth and latency; and
- *Complexity:* Many services are complex and not well understood.

Middleware technology has become quite important in applications representing a wide range of industries, including: aerospace and defence, banking and finance, telecommunication, and education.

It is expected that middleware will also play an important role in many new applications – such as digital libraries, multimedia, virtual manufacturing, air-traffic control simulation, etc. These applications need infrastructure support in order for data to move freely among different systems.

Middleware Communication Services
Early middleware mainly provided communication services. Today’s middleware provides many services that can be categorised into three broad areas:

Communication services handle low-level communication between the client and the server. For instance, the Remote Procedure Call (RPC) mechanism management support services are used by many applications, such as name management, security, failure handling, and memory management application-specific services provide assistance for classes of applications, such as SQL (structured query language) database access, transaction processing, and data-replication services.

Here, we focus on middleware technologies that provide most of these services: DCE, CORBA, DCOM, and Java. All of these technologies are considered to be general-purpose, unlike some technologies that target specific markets, such as database-specific middleware.

Distributed Computing Environment
The Distributed Computing Environment (DCE) is a software infrastructure for developing distributed systems. It consists of a set of application programming interfaces and a set of run-time services, which together provide the fundamental functionality required for distributed applications. DCE is based on a simple but flexible Remote Procedure Call (RPC) paradigm. In the middleware arena, DCE is generally considered low level in that it implements primitive fundamental services.

DCE is produced by the Open Group (Cambridge, MA), an integrator specialising in distributed systems and open systems standardisation. The Open Group provides a reference DCE implementation and a set of conformance tests. Resellers can purchase the reference implementation and port it to their platforms or develop their own DCE product.

One of DCE’s strengths is the number of interoperable implementations available. Vendors that provide DCE products include Fujitsu, Hewlett-Packard/Compaq, Hitachi, IBM, NEC, Siemens, Sony and Silicon Graphics.

DCE consists of a set of software packages layered on top of an operating system. In addition to its RPC-based communication services, it provides a set of basic or core services that includes: a naming service, Cell Directory Service (CDS); a security and authentication service; and a time synchronisation service, Distributed Time Service.

In addition, there are a few application-level services associated with DCE. Two well-known services are Distributed File Service (DFS) and Encina, a transaction processing support package. Security and CDS are strong components of DCE. DFS provides many attractive features as a distributed filesystem. Encina is an attractive transaction processing monitor that takes advantage of DCE’s underlying core services.

DCE uses an Interface Definition Language (IDL) to specify procedure interfaces. DCE IDL is similar to other interface languages, including CORBA IDL, but it is unique. Note that there is no standard for IDLs. Currently, only the C language is directly supported by DCE IDL.

Despite DCE’s positive features, it has many problems, including complexity and lack of object support in its architecture.

Common Object Request Broker Architecture
The Common Object Request Broker Architecture (CORBA) is a standard for transparent communication between application objects. The CORBA specification is developed by the Object Management Group (OMG), a non-profit industry consortium of more than 800 software vendors and users involved in the development of object technology for distributed computing systems. The OMG does not produce any software; rather, it provides specifications that come from OMG members who respond to requests for proposals.

In 1990, the OMG introduced a reference architecture for object-oriented applications called the Object Management Architecture (OMA) with many subsequent revisions. The OMA is mainly composed of an object model and a reference model. The OMA object model defines how objects are specified in a distributed environment. In this client/server model, the servers are objects that provide services to clients, and the clients obtain services by invoking operations on server objects.

The OMA reference model defines the OMA components, their interfaces, and the interactions between components and interfaces. These components are: Object Request Broker (ORB), Object Services, Common Facilities, Domain Interfaces, and Application Interfaces.

The ORB enables clients and objects to communicate in a heterogeneous distributed environment. It provides the communication service within the middleware. The Object Services provide core services for using and implementing objects. The OMG has specified the following services: Naming, Event, Life Cycle, Persistent Object, Transaction, Concurrency Control, Relationship, Externalisation, Query, Licensing, Property, Time, Security, and Object Trader.

The Common Facilities are interfaces for horizontal end-user-oriented applications that can be used by many application domains, such as user interface and information, system, and task management. Domain interfaces are aimed at specific application domains such as manufacturing, finance, and health. The application interfaces are non-standard interfaces specific to a certain application.

The CORBA specification, first introduced in 1991, describes the interfaces and services that ORBs must have. In effect, CORBA is basically the technology adopted for ORBs. CORBA provides a “clean model” – that is, the interface of an object and its underlying implementation are separated. Clients do not need to know how or where servers are implemented. Server objects are visible only through interfaces and object references. The specification includes an IDL for describing interfaces. IDL mappings to several languages (such as C, C++, Java, and COBOL) have been specified and provided by many vendors.

Currently there are at least a dozen ORBs from different vendors in the commercial market. All provide similar basic services but differ in the number of platforms supported and services provided. Most of them are available on Windows 95, Windows NT, and many Unix platforms. Examples include: Orbix, IONA Technologies; VisiBroker, Visigenic; ObjectBroker, BEA Systems; ORBplus, HP; and Component Broker Connector, IBM.

CORBA is used in many applications representing different industries, e.g., aerospace and defence, banking and finance, manufacturing, health care, telecommunications, petroleum, transportation, insurance, and education. These applications range from small prototypes and experiments to mission-critical projects.

Distributed Component Object Model

Distributed COM is an extension of the Component Object Model (COM), developed by Microsoft Corp. as an object-based programming model for developing and deploying software components. DCOM (previously called Network OLE) adds distributed support to COM to work across a network. The addition includes communication with remote objects, location transparency, and an interface to distributed security services.

COM is the basis of many Microsoft object-based technologies such as ActiveX and Object Linking and Embedding (OLE). ActiveX is now being promoted by Microsoft as a complete environment for components, distributed objects, and Web-related technologies. In essence, ActiveX is Microsoft’s label for a wide range of COM-based technologies.

COM-based technologies are directly supported by Microsoft on the Windows operating system. Other software vendors like Software AG, are currently porting some of these technologies to other platforms such as Unix. COM has been part of Windows for some time. It separates object interfaces from their implementations, similarly to CORBA. Each COM object is an instance of a COM class. One or more COM classes are housed in a COM server, which provides the necessary structure around an object to make it available to clients.

There are three kinds of COM servers:
1. In-process servers execute in the same process as their clients;
2. Local servers execute in a separate process from their clients but on the same machine; and
3. Remote servers execute in a separate process on a different machine and use DCOM to communicate with remote servers.

COM defines a binary call standard for interfaces. It also defines a language for specifying interfaces called Microsoft Interface Definition Language (MIDL). An MIDL compiler is provided to generate client proxies and server stubs in C or C++ from an interface definition, similar to client stubs and server skeletons in DCE and CORBA.

The DCOM extension enables COM processes to run on different machines. DCOM uses an extension of the DCE RPC for interactions between DCOM objects called Object RPC, or ORPC. The main difference between these two is that when the client asks the registry for the server location, the DCOM registry points to an IP address, and COM points to a location on the local machine.

Both COM and DCOM provide facilities that are similar to CORBA services – but are not called services because they are built into the COM library. These facilities provide functions such as naming, security, transaction, persistence, life cycle, versioning, event, and data access services.

DCOM is integrated into Windows 2000, and NT 4.0 and can be downloaded free of charge for Windows 95/98. For instance, Software AG has ported DCOM to many Unix platforms, such as Sun Solaris 2.5 and Digital Unix 4.0, and will porting it to other platforms as well.

Java Technologies
Java, developed by Sun Microsystems, is a label for a broad range of object-based technologies for the Web and distributed applications. It is an object-oriented programming language as well as a computing platform for developing and running distributed applications.

The Java platform consists mainly of the Java Virtual Machine (JVM) and Java API, which includes Java core classes. The Java platform sits on top of many other platforms and compiles to byte codes – machine-independent instructions for the JVM.

The Java Platform is currently embedded in browsers such as Netscape Navigator, and runs on many other platforms such as Windows, Unix, IBM MVS, and network operating systems.

The Java API, a multi-platform standard interface, specifies a set of interfaces for a wide range of applications. There are two types of Java API: the Java Base API, which defines the core features for developing and deploying Java programs, and the Java Standard Extension API, which extends the capabilities of the core API.

Java programs are categorised as applets, applications, or JavaBeans. Applets are small or modular programs that require a Java-compatible browser to run. Java applications, on the other hand, are standalone programs that do not require a browser. And JavaBeans are reusable software components that are platform independent, and allow dynamic interactions with other components. Unlike an applet, a JavaBean can interact with other components over the network and can run in containers other than a browser.

Distributed objects, such as JavaBeans, can interact with other applications across a network using the Java Remote Method Invocation (RMI), which enables objects in one JVM to invoke methods on objects in a remote JVM. The invocation methods on both local and remote objects use the same syntax.

An RMI compiler (RMIC) generates stubs and skeletons, as in CORBA and DCOM. Recently, Sun announced that Java RMI will support CORBA IIOP (Internet InterORB Protocol) as a transport protocol in addition to its native transport protocol.

Java provides many services – though most are not well defined – that are similar to CORBA services. Some have been developed along with other Java technologies such as JavaBeans and Java Platform. Among these services are security, Java DataBase Connectivity, Java Naming and Directory Interface, Reflection and Introspection, Event, Persistence, and Java Transaction Service.

Similarities and Differences
The four middleware technologies described provide somewhat similar functionality. However, there are some key differences including maturity, ownership, standardisation, platform support, and services provided.

Although DCE and CORBA have been around for several years, in general DCE is more mature and has a more complete set of core services than the others. But DCE suffers from the lack of object support in its architecture.

Although CORBA has a richer set of services than the others, many of those have been approved recently, and will take time to be implemented. In addition, many CORBA facilities and some services are still being reviewed by the OMG. It is expected to take some time for CORBA to have a complete, mature, and fully implemented set of services. That should not dissuade software developers from using CORBA now, as it is the only general-purpose middleware based on object-oriented technology.

Both DCOM and Java are owned, specified, and developed primarily by their vendors, Microsoft and Sun, respectively. The advantages and disadvantages of vendor-specified products, compared to standards committee specifications are well known and understood in the industry.

Among the four technologies, only DCOM is not widely supported on multiple platforms. It runs mainly on Windows and on a few Unix systems through third-party vendors. In contrast, DCE, CORBA, and Java are widely supported on many platforms.

Progress Report
Middleware will become increasingly sophisticated because it allows users to access remote services as if they were local.

Vendors and users are increasingly dependent on standards, as they want to ensure that their investment in developing and integrating middleware services pays off. Many committees are working on introducing standards into industrial products, and many vendors are pushing to make their products de facto standards.

Because of the constant changes in middleware technology – which are hard to stay on top of without vigilance – this market may take some time to stabilise. In the meantime, each organisation should choose its middleware based on current and future needs. Yet, it is important to remember that middleware is an infrastructure – where an initial error in choice will not be so easy to correct!

Furthermore, selection of the communications mechanism is a key issue when designing and building a distributed application. If things were simple, we would communicate application to application using a direct synchronous paradigm. We would call the remote application or resource, and the remote application would return status or data. However, the real world is not that simple.

There are many different communications paradigms these days. For our purposes in this discussion, we can limit these as follows:

1. Synchronous;
2. Asynchronous;
3. Connection-oriented;
4. Connectionless;
5. Direct; and
6. Queued.

Take note that some middleware products may use one, two, or all of these communications paradigms. As such, grouping these concepts in terms of middleware categories is not relevant.

Synchronous: In synchronous communications, the calling program sends a request to a remote program and waits for the response. A remote procedure call (RPC), such as the one that exists within products like the Open Software Foundation's Distributed Computing Environment (DCE), is the best example of a synchronous middleware layer.

Synchronous communications means that the calling application must stop processing or is blocked from proceeding until the remote procedure produces a response.

Asynchronous: Asynchronous communications are unblocked or do not block the program from proceeding. The program can make the request and continue processing before a response occurs.

Most message-oriented middleware layers support asynchronous communications through the point-to-point-messaging or message queue models.

Connection-oriented: Connection-oriented communications means that two parties first connect, exchange messages, and then disconnect.

Typically this is a synchronous process, but it can be asynchronous.

Connectionless: Connectionless communications means that the calling program does not enter into a connection with the target process.

The receiving application simply acts on the request, responding if required.

Direct: In direct communications, the middleware layer accepts the message from the calling program and passes it directly to the remote program.

You can use either direct or queued communications with synchronous processing; however, direct is usually synchronous in nature and queued is usually asynchronous.

Queued: When using queued communications, the calling process, typically a queue manager, places a message in a queue. The remote application retrieves the message either shortly after it's been sent or at any time in the future except for time-out restrictions. If the calling application requires a response, such as a verification message or data, the information flows back through the queuing mechanism.

The advantage of the queuing model over direct communications is that the remote program does not need to be active for the calling program to send a message to it. What's more, queuing communications middleware typically does not block either the calling or the remote programs from proceeding with processing.

Which paradigm should you use? The answer depends on both the functional and performance requirements of the application. But let's look at the trade-offs of each category selected.

In many respects middleware is the glue that holds client/server applications together, as well as a productivity tool that saves developers from having to hassle with the details. It's a layer that exists between the application and the underlying complexities of the network, the host operating system, and any resource servers such as database servers.

Middleware makes vastly different platforms appear the same to an application and, if used correctly, saves time and money in the development of client/server and Internet/intranet systems.

Yet, middleware is changing. Where RPCs and native database middleware once ruled the land, the next generation of middleware products now ties distributed processing together and ties applications to resource servers.

Where we once built applications that communicated intra-system using inter-process communications, we now use middleware to link the same objects across many different systems.

What's more, with the maturation of the multi-tiered client/server architectures, middleware is providing a location for application processing, placing the business logic on a centralised layer.

Finally, the Web is bringing a new paradigm and new products to middleware. The new generation of Web-enabled intranet or Internet applications needs glue for database and application server connections.

Clearly, middleware is becoming more complex. The number of new product introductions is not exploding, although the variety of middleware is increasing, as are the capabilities. So how do you get a grip on this moving target?

First, you need to understand the standards and forces at work. Second, you need to identify the categories of middleware available and what purposes each category serves. Finally, you need to understand the products and which product fits into which category.

Box Story 1:

Middleware Goals

The goal of a middleware initiative is to assist in the creation of interoperable middleware infrastructures among the related communities. This goal is guided by several principles:

1. *Make it happen:* In order to deploy the advanced applications that are at the core of the initiative, interoperable middleware infrastructures are needed. The challenges lie as much in policy and practice as in technology. The need is immediate, and broad participation and significant resources will be required.

2. *Be an honest broker:* Interoperable middleware infrastructures are at the intersection of the interests of a large number of communities. Organisations are all in urgent need of middleware services. We need to work with these groups and help them to work with one another.

3. *Integrate across applications:* Middleware is a true infrastructure, and as such it should be able to work with the widest possible variety of applications. A single infrastructure should support the goals of organisations, and should be tightly coupled to key administrative systems.

4. *Interoperate between networks:* In line with the mission, middleware will need to interoperate and do so at both functional and policy levels.

Box Story 2:

Software Principles

In many user scenarios, there is work devoted to the identification, development and deployment of middleware that supports organisational needs. Basically, the design of the necessary protocols and software should be guided by the following principles:

1. *The software should be loosely coupled.* Given uncertainties such as the volatility of the technologies involved, it is likely that middleware will go through a rapid evolution in the next few years. Organisations will want to replace and enhance components without having to redo the entire infrastructure.

2. *Software deployments should demonstrate early wins.* Given the political aspects of middleware deployment, it will be very useful to show immediate benefits of early components. This will help motivate the significant organisational investments that will be required. Individual components should have value in themselves as well as in concert.

3. *Make software as economically and technically cheap as possible.* Financial stresses and employee retention issues suggest keeping software and expertise costs low.

4. *The software systems should accommodate the distinctive aspects of the organisation.* For example, the commercial IT environment has a number of special characteristics, such as the migratory workstation habits of employees, as well as freedom and privacy. Middleware solutions must accommodate these characteristics.

5. *The software should be easy to use.* End users prefer natural naming and intuitive tools. Users may not be able to handle complexity in management of middleware components or personal data.

Box Story 3:

Defining Middleware

Simply put, middleware places an easy-to-use API between the application and the resource that the application needs.

or example, if you are building a Java applet and would like to gather data as part of the operation of the applet, you can use Java Database Connectivity (JDBC) classes to access information on any number of databases. The JDBC classes remove the developer from the complexities of the target database and allow the applets to use any number of databases without having to understand the native feature of each database.

Use a common API, and the middleware does all of the work. The same concept applies to Open Database Connectivity (ODBC) for Windows-based client/server applications and even to proprietary middleware layers such as Borland's Database Engine (BDE).

Middleware goes well beyond simple database access operations. It's also able to provide transparent API-level access to other systems and system services without requiring that the developer know where the systems are.

The middleware layer can find the system using some sort of naming service, invoke a remote process, and return a response to the calling process.

Products such as the Open Software Foundation Research Institute's (OSF) Distributed Computing Environment (DCE), distributed object technology such as those based on Common Object Request Broker Architecture (CORBA), and most message-oriented middleware (MOM) products are examples of such middleware.

Base of Standards
One of the trends in today's middleware products is the movement to standards, including ad hoc standards driven by powerful vendors (as opposed to committee standards such as CORBA).

In the past, middleware was a proprietary proposition without the possibility for interoperability. Today, however, we are learning to use standards such as CORBA and Distributed Component Object Model (DCOM) as a base-level model and thus the wire for middleware products.

DCOM serves as a standard base for homogenous Windows environments. DCOM allows COM-enabled applications such as those built using Visual Basic, Delphi, and PowerBuilder to link to and invoke services of other COM-enabled application across a network using a remote procedure call (RPC) mechanism.

Thus, DCOM serves as a primitive middleware layer itself or as the infrastructure for other middleware products. Middleware products that use DCOM include Microsoft's Transaction Server TP monitor and Microsoft's Message Queue Server (MSMQ). Transaction Server is a TP monitor that defines transactions using ActiveX controls, and it communicates with applications and resource servers using DCOM.

But DCOM is not the only ORB, nor was it the first. CORBA also provides an infrastructure for middleware products, using CORBA's natural ability to communicate with other ORBs running on similar or dissimilar systems through a common interface and wire-level protocol.

The idea here is to tie systems together using the plumbing of CORBA and thus provide a virtual system for application developers. Products that use CORBA for their infrastructure include VisiBroker from Visigenic Software and Orbix from Iona Technologies Inc. Both of these ORBs, by the way, provide bindings to Java, also making them Web-enabled middleware.

Internet InterORB Protocol (IIOP) is the wire-level protocol that these CORBA ORBs use together. IIOP is also considered by Internet vendors such as Netscape Communications to be the next wave in Web connectivity technology.

Types of Middleware
Middleware comes in a few flavours. For our purposes, let's divide middleware into database-oriented, virtual system, middle-tier, gateways, and Web-enabled. Of course, the problem with categorising middleware is that it's difficult to do so.

For instance, some database-oriented middleware may use a MOM infrastructure; and TP monitors can use almost any sort of middleware even though TP monitors are themselves middleware.

Database-Oriented: Database-oriented middleware, as you may have guessed, includes those middleware products that enable applications to communicate with local or remote databases. The idea is to provide some sort of API access to a database using the middleware layer to traverse the operating system and network layers on behalf of the client.

In many cases, even the API is hidden from the developer behind the features of the development tool. For instance, in the world of client/server development, database-oriented middleware is built in. PowerBuilder, for example, comes with native links to most popular databases; or you can use ODBC. Borland provides its BDE with almost all the company's development tools. The BDE lets developers communicate with any number of databases using native or ODBC calls.

JDBC is the most significant new database middleware standard. JDBC defines a call-level interface (CLI) for Java development and is not a part of the latest Java Development Kid (JDK) from the JavaSoft division of Sun Microsystems Inc. JDBC is really just a set of Java classes for specific databases and borrows heavily from the ODBC architecture. to see JDBC as part of Java development tools such as JBuilder from Borland.

OLE-DB provides a single point of access for multiple databases. While moving slowly, it is adding value to the aging ODBC standard. With OLE-DB, the goal is to provide OLE automation access to any number of databases or add a COM layer between the application and the database.

There are tool-independent database-oriented middleware products as well. Rogue Wave Software's DBTools.h++, for example, connects most C++ applications to databases. DBTools.h++ is able to make relational tables and attributes appear as native C++ objects. If Java is more your speed, Rogue Wave also has a Java-enabled version known as JDBTools, which provides database connectivity directly from Java applets and applications. A similar product includes Persistence from Persistence Software – where Persistence wraps relational databases to make them appear as objects for easy manipulation by an object-oriented application development tool.

If it's legacy data you're after – or the ability to access data on multiple machines – then you should look at high-end database-oriented middleware layers such as EDA/SQL from Information Builders (IBI). EDA/SQL's approach to database middleware is to support as many operating systems, networks, and databases as possible.

For example, you can access information on a DB2 database running on an mainframe, using a simple ODBC driver from a client. Such a high-end product is helpful to organisations looking to migrate to client/server without having to turn off their business-critical legacy systems.

Virtual System: Virtual system middleware enables developers to deal with many different systems as if they were a single system, using a common middleware layer and API set. This is true plumbing. You install native versions of the virtual system middleware layer on each system, configure a directory service, and link to applications, thus exposing heterogeneous system-level services to other applications on the network.

By invoking a series of APIs, you may be launching a process on a mainframe, updating a database, and invoking an application service on a Unix system, all from the same client application.

DCE is a good example of virtual system middleware, providing RPC-based access over a variety of systems that DCE supports. DCE provides its own security service and directory service, among other things, and is able to communicate with any number of resource servers using gateways and interfaces.

However, DCE, is not able to meet the performance expectations of client/server developers requiring a rather heavy-duty server and network connection. What's more, the synchronous RPCs required that all participating systems be active in order for an operation to take place. Typically, RPCs must complete their operations before the application can resume operation. In other words, they block the application.

MOM is able to get around some of the limitations of RPC-based middleware like DCE by providing an asynchronous communication mechanism using messages. This means that the target server need not be up and running, and the MOM API call does not block the application.

What's more, MOM does not require a huge amount of network bandwidth and is able to track transactions using persistence or storing the transmission to disk, to recover from system and network failures.

Examples of MOM products include IBM's MQSeries, and Microsoft's MSMQ.

Box Story 4:

Middleware FAQ

What is middleware?
The term middleware is used to describe a broad array of tools and data that help applications use networked resources and services. Some tools, such as authentication and directories, are in all categorisations.

Other services, such as co-scheduling of networked resources, secure multicast, and object brokering and messaging, are the major middleware interests of particular communities, such as scientific researchers or business systems vendors.

One definition that reflects this breadth of meaning is “Middleware is the intersection of the stuff that network engineers don’t want to do with the stuff that applications developers don’t want to do.”

Why is middleware important?
Middleware has emerged as a critical second level of the enterprise IT infrastructure, between the network and application levels. The need for middleware stems from the increasing growth in the number of applications, in the customisations within those applications, and in the number of locations in our environments.

These and other factors now require that a set of core data and services be moved from their multiple instances into a centralised institutional offering.

This central provision of service eases application development, increases robustness, assists data management, and provides overall operating efficiencies.

Why is middleware urgent?
There are several drivers bringing middleware to organisations. For example, consider the requirements for middleware services such as authentication and directories. Large projects will be extending across a broader community.

Or the preparing requirements for digital signatures for, say, loan forms. New versions of software, such as Windows 2000, come with the tools to build ad hoc middleware components. It is urgent that campuses build a coherent infrastructure to respond to these drivers.

What makes the need for middleware distinctive?
Many companies and other communities of interest are coming to understand the importance of middleware to their missions, and are proceeding with development. Here they face unique technical and policy issues in its deployment.

Technical issues include the employee mobility, the diversity of equipment, and the requirements of advanced applications. Policy issues include ownership of data, and extended collaborative relationships.

Together these considerations make middleware deployment within higher education significantly harder than deployment outside of it.

When middleware becomes part of the IT environment, how critical will a robust infrastructure be?
The middleware components of the future IT environment will be every bit as critical as the underlying network infrastructure, requiring 24x7 service, high performance, and appropriate redundancy.

Directory services will receive millions of hits per day; identifiers will have explicit control mechanisms; attribute services will be invoked by almost every application. In addition, there are strict operational constraints on security services.

Is middleware a centralised or a distributed issue on campus?
It is both. Like network services, there is a need for a consistent infrastructure across campus that is best provisioned centrally. At the same time, many parts of the contents of this infrastructure are best maintained by the individuals themselves, and by their departments.

The trick is to create a centrally coordinated service that provides tools and authority for distributed management of the contents.

Aren't we going to get middleware from the commercial marketplace?
It is certainly the case that many basic middleware products deployed will be commercial products. These products will come both from diversified software companies.

Depending on the situation, a number of distinctive characteristics of the community create design considerations that require complex implementations.

Finally, the collaborative nature of work will raise interoperability issues that must be addressed within the community.

What kind of investments will organisations need to make?
Like networking, middleware will require considerable commitments of time and money. However, the types of costs are different.

Networking has required large sums of capital (for fibre, routers, switches, etc.) and considerable operating costs (for external access, maintenance, etc.) Personnel costs have been relatively modest.

For middleware, the hardware costs (servers, readers, etc.) are likely to be relatively low. Software costs are unclear now, but there are clearly considerable expenses in building bridges to legacy systems and to evolving middleware-enabled applications.

Unlike networking, middleware has a second major cost component: process time. An organisation must develop consensus and support for the deployment of middleware, clarify data ownership and management issues, specify relationships among individuals, groups and information technology objects, establish legal agreements, and change the way that information is managed on the campus.

How does the initiative intend to address these needs?
Efforts will focus on advancing the level of middleware within higher education. A set of related activities will include fostering technical standards, aggregating and disseminating technical design and implementation strategies, fostering opportunities for vendors to shape and deploy products, and integrating efforts with specific scientific and research communities.

What should campuses be doing now?
It is not too early for organisations to begin the processes that address the policy side of the challenge, building awareness about the need for middleware, identifying key constituencies that will be involved in the process, and taking basic inventories of the data and management relationships on campus.

At the same time, experimentation in the core technologies, most notably in directory services, should be undertaken.

About the Author
Leon A. Enriquez is managing editor and business manager of Editorial Thoughtscapes – a professional writing firm. Leon can be reached at leonenriquez@et-writer.com.

<ENDS>