Middleware: The Glue for Modern Applications

Management Summary

Middleware is the software “glue” that helps programs and databases that may be on different computers work together. Middleware is crucial to the success of modern applications because most computing is now distributed computing. Most application systems, including all client/server and Internet applications, divide processing and/or data across two or more computers connected by a network. More than 95 percent of new applications use some type of off-the-shelf middleware to facilitate communication among their component parts.

Most early client/server applications (circa 1986-1994) were aimed at low-end or moderately demanding business applications. They generally used two-tier architectures, and middleware was not much of an issue because it was embedded unnoticed in the database management system (DBMS) or in a network operating system. The most widespread forms of middleware still are remote file systems, such as those in Novell’s NetWare, Sun Microsystems’ NFS and Microsoft Windows; and remote data access middleware that is bundled into DBMSs such as DB2, Oracle and SQL Server. However, the more-demanding, mission-critical enterprise applications have long required multitier architectures and more powerful program-to-program middleware such as transaction-processing monitors (TPMs), e.g., Customer Information Control System (CICS) and Tuxedo. Moreover, a variety of business pressures and technological advances are driving major changes in middleware use. The newer generations of distributed applications, particularly Web-based systems and integrated systems that combine multiple applications from different sources, are bringing the need for friendlier, richer and more flexible middleware. In response to escalating requirements, the traditional file-oriented, data-oriented and TPM middleware products are evolving, and other forms of middleware are emerging. Enterprises are accelerating their use of middleware forms such as:

- Component Object Model (COM), Common Object Request Broker Architecture (CORBA) and Java-based component-oriented middleware, including application servers, object request brokers (ORBs) and object transaction monitors (OTMs);
- Message-oriented middleware (MOM), including queuing systems and publish-and-subscribe services;
- Internet-inspired technology, including browsers, Web servers, Extensible Markup Language (XML) tools and related E-commerce services; and
- Application integration services such as gateways, integration brokers, transformation engines and business process managers.
This *Strategic Analysis Report* introduces the fundamental middleware technical concepts in plain language. It presents a taxonomy of middleware and an overview of the historical evolution of the different kinds of middleware. It provides examples of each type of middleware and describes their basic characteristics and uses to help managers, IS architects and developers understand how to get started using middleware.

This report explores the following Key Issues:

- What is middleware and why will it matter?
- How will users make middleware and architecture choices that work despite the ongoing industry transition?
- How will ORBs, TPMs, OTMs and Web application servers evolve to support the changing requirements of modern enterprise applications?
- Which technical and business trends will have the most impact on IS architecture and middleware strategies during the next five years?
- What will be the most successful ways of integrating new, purchased and legacy applications?

The report presents analysis related to the following Strategic Planning Assumptions:

- Ninety percent of new Web and other client/server applications will be multitier (i.e., three or more tiers) in 2001, up from 55 percent in 1998 (0.8 probability).
- More than 80 percent of large enterprises will use COM and some aspect of CORBA or Internet Inter-ORB Protocol (IIOP) somewhere in their enterprise applications by 2001 (0.8 probability).
- The middleware for architectural-level, intra-application communication will continue to differ from the “city planning”-level integration middleware used between independently designed applications through 2004 (0.9 probability).
- By 2000, use of OTM-based components and frameworks will enable development of three-tier applications that are as intuitive, easy to use and effective as the current two-tier technologies, making OTMs the mainstream technology (0.8 probability).
- By year-end 1999, 75 percent of middleware vendors will have deployed or at least announced XML support, including compliance for at least XML and DOM (0.9 probability).
CONTENTS

1.0 Overview ........................................................................................................................................ 7
1.1 Introduction ...................................................................................................................................... 7
1.2 Defining Middleware .......................................................................................................................... 7
1.3 Why Have Middleware? ................................................................................................................... 8

2.0 Middleware Taxonomy ....................................................................................................................... 9
2.1 Basic Middleware ............................................................................................................................. 9
2.1.1 Data Management Middleware .................................................................................................. 10
2.1.2 Communication Middleware .................................................................................................... 11
2.1.3 Platform Middleware ............................................................................................................. 12
2.2 Middleware for Application Integration ......................................................................................... 13
2.2.1 Gateways ................................................................................................................................. 15
2.2.2 Superservices............................................................................................................................ 16
2.2.3 Integration Brokers .................................................................................................................. 17
2.2.4 Business Process Managers .................................................................................................... 19

3.0 Evolution of Application Platforms ................................................................................................. 19
3.1 DBMSs as Application Platforms .................................................................................................... 19
3.2 Platform Middleware Market Requirements ................................................................................ 21
3.2.1 Systematic Market Requirements ............................................................................................ 22
3.2.2 Opportunistic Market Requirements ....................................................................................... 23
3.3 The Impact of Component Software .............................................................................................. 24

4.0 The Role of Communication Middleware ......................................................................................... 26
4.1 Characteristics of Communication .................................................................................................. 26
4.1.1 Connection-Oriented and Connectionless Relationships .......................................................... 26
4.1.2 Synchronous and Asynchronous ............................................................................................... 27
4.1.3 Communication Patterns ......................................................................................................... 27
4.2 What Happened to RPCs? .............................................................................................................. 30
4.2.1 Examples of MOM and Other Communication Middleware .................................................. 31

5.0 Integration Middleware ................................................................................................................. 31
5.1 Supporting Extended and Composite Applications ......................................................................... 32
5.2 Screen Scraping and Integration Servers ....................................................................................... 34
5.3 Superservices.................................................................................................................................. 35
5.4 Wrappers ....................................................................................................................................... 36
5.5 Integration Brokers ......................................................................................................................... 37
5.6 Adapters ....................................................................................................................................... 40
5.7 XML Integration Servers ............................................................................................................... 41
6.0 Data-Oriented Integration Technology ................................................................. 41
6.1 Database Gateways ............................................................................................... 42
6.2 Replication and Propagation ............................................................................... 43
6.3 Transformation Engines ..................................................................................... 44
7.0 Conclusions ......................................................................................................... 45
Appendix A: Acronym Key .......................................................................................... 47
FIGURES

Figure 1. Distributed Application With No Middleware ................................................................. 9
Figure 2. Basic Middleware ............................................................................................................ 10
Figure 3. Remote Data Access Middleware .................................................................................... 10
Figure 4. Communication Middleware .......................................................................................... 11
Figure 5. Middleware in the OSI Stack .......................................................................................... 12
Figure 6. Platform Middleware ..................................................................................................... 12
Figure 7. Intra-Application and Interapplication Middleware .......................................................... 14
Figure 8. Full Middleware Taxonomy ............................................................................................ 15
Figure 9. Alternative Topologies for Middleware Gateways ............................................................ 16
Figure 10. Examples of Superservice Layers ................................................................................ 17
Figure 11. Separate Integration Broker ......................................................................................... 18
Figure 12. Local Integration Broker ............................................................................................. 18
Figure 13. Variations of Platform Middleware .............................................................................. 20
Figure 14. Two Markets for Application Servers .......................................................................... 22
Figure 15. Communication Models .............................................................................................. 28
Figure 16. Heterogeneous Applications Using Basic Middleware ................................................. 32
Figure 17. Simple Use of Integration Middleware ......................................................................... 33
Figure 18. Example of a Composite Application .......................................................................... 33
Figure 19. Wrappers ......................................................................................................................... 36
Figure 20. Alternative Roles for a Broker ...................................................................................... 38
Figure 21. Compound Broker Topologies ..................................................................................... 39
Figure 22. Transformation Engines ............................................................................................... 44
1.0 Overview

1.1 Introduction

Middleware is the essential foundation for distributed computing. Its most basic function is to enable communication among application programs or DBMSs, whether within a single application system or across multiple application systems. Middleware is used in every common form of modern business or governmental computing, including client/server and Web-based applications; custom applications and purchased packages; procedural and object-oriented programs; intraenterprise or interenterprise business processes, including E-commerce; and large-enterprise and small-business systems.

The rapid evolution of application architecture in recent years, from centralized, monolithic, dumb-terminal architectures to distributed, multitier, component-based systems, has been enabled by the equally rapid evolution of the underlying middleware. This report describes all of the major categories of middleware with the premise that the communication between programs and databases that are within the same application system has fundamentally different requirements than the communication between programs and databases that are in separate systems. In our view, the most important criterion for determining whether something is part of the same “application system” is whether it is designed by the same group or whether it is independently designed. Programs and databases that are designed by one group can use a common set of data models, object models, business models and compatible software and hardware technology. By contrast, programs and databases that are designed by different teams at different times, in different departments or different companies, and in different physical locations, will almost inevitably be inconsistent in all of these. This has important implications with respect to the choice of middleware and how it is used.

Each of the kinds of middleware reviewed in this Strategic Analysis Report has its areas of strength and its limitations. A large enterprise is likely to use at least one product from every category of middleware somewhere in its application portfolio. The goal of this report is to make it easier to find the right tool for each particular job.

1.2 Defining Middleware

The term “middleware” is used in many ways. In this report, middleware is the software “glue” that helps programs and databases that may be on different computers work together. More formally, we define middleware as “runtime system software that directly enables application-level interactions among programs in a distributed computing environment.” Here, system software means software that is positioned between an application program and lower-level operating-system, data management and networking services. A computing environment is physically distributed when its programs and/or databases are spread across two or more computers. Middleware is also useful when logically distributed components happen to run on the same computer. Note that a DBMS is a set of programs, so the wording of this formal definition covers distributed data as well as distributed application code. Application-level interactions are those that transfer business data and/or information about its semantics or context (not just technical “housekeeping” data) to or from an application program.

Examples of middleware include application servers, TPMs, MOM, ORBs, OTMs, remote procedure call (RPC) services, integration brokers (including message brokers), business process managers (including workflow services) and database gateways. Development tools that have no runtime aspect and most system management utilities are not middleware because their impact is indirect; i.e., they are not directly involved in the transfer of application information between programs.
Our inclusion of runtime platforms such as application servers, TPMs, ORBs and OTMs makes our definition of middleware broader than some other popular definitions. Elsewhere, the term “middleware” is sometimes limited to program-to-program communication services such as MOM and RPCs (what we would call “communication middleware”). We believe that a definition that only includes communication middleware is too narrow. There are many other kinds of software that “sit in the middle” (between the application, the operating system and the network) acting as “glue.” It is good to have one general term (“middleware”) rather than multiple terms for each form of such software. Note also that application servers, TPMs, ORBs and OTMs (what we would call “platform middleware”) do the same thing that MOM and RPCs do, i.e., they send messages between programs, although they also do other things like managing system resources. Furthermore, most people would agree that database gateways are a kind of middleware, so MOM and RPCs together do not constitute a full list of middleware products even in the commonly used narrow definitions.

There is also no industry consensus on whether integration brokers, integration servers and business process automation managers (such as workflow systems) are middleware. We consider them to be a form of high-level middleware because they facilitate the interactions of application programs and DBMSs. They include program-to-program data transfer among their functions, although their role goes far beyond mere communication (and they may leverage other forms of middleware to actually move the bytes around). Brokers, and, to an even more pronounced extent, business process automation managers, do work that would otherwise have been implemented in an application program (or been left unautomated). Just as communication middleware (e.g., MOM) assumes or hides some of the functions of the communication stack, and platform middleware (e.g., application servers) assumes or hides some of the functions of an operating system, integration middleware (e.g., brokers and process managers) assumes or hides some of the functions of the application program. But, as the name implies, all of these are in the middle so we call them all middleware.

1.3 Why Have Middleware?

Most computing is now distributed computing; i.e., most application systems divide their processing and/or data across two or more computers connected by a network. The most common distribution occurs across the network link between the desktops and a server, but in many cases, data and messages also are sent among multiple servers. Communication may span a wide-area network (WAN), a local area-network (LAN) or both.

All forms of middleware have as one of their primary goals the task of facilitating communication among the components of an application system or among multiple application systems. Computer communication is inherently complicated. It brings the need to find resources wherever they are located on a network; establish communication links; package (“marshal”) data for transmission; dispatch the message; transport the bits across some physical network; combine packets on the receiving end; validate that the whole message was received; check for errors, and report and correct errors where possible; unpack (“unmarshal”) the data; deliver the message to the receiving program; and “tear down” the connection to return control of memory and other resources to the operating system. The purpose of device drivers and other layers (e.g., TCP and IP) in a communication software “stack” is to offload the most mundane, yet difficult aspects of this work from the application program. Middleware — or, in the absence of middleware, code in the application program — does the remaining work.

Developers can get one program to talk to another program by writing calls to the basic network software that is found on virtually every computer (see Figure 1). The most common direct interface between a program and the networking software is a socket, a low-level application programming interface (API) for TCP/IP. In Systems Network Architecture (SNA) networks, the common counterparts to sockets are the
Common Programming Interface for Communications (CPI-C), Advanced Program to Program Communication (APPC) or High Level Language API (HLLAPI). (These technically are not transport layer interfaces; rather, they are application-layer interfaces that let most of the transport-level issues show through.) Other APIs are available for both IP and SNA networks, and other networks such as NetWare’s SPX/IPX and Compaq’s DECnet also have their respective API counterparts.

### Figure 1. Distributed Application With No Middleware

When applications interface directly to networking software, no middleware is involved. This puts the burden of handling all of the upper-level communication-related chores (generally everything above the transport layer, as described in the OSI communication model) on the developer. The vast majority of distributed applications are no longer written this way, however. More than 95 percent of all distributed processing uses some form of off-the-shelf middleware to offload some or all of this work.

#### 2.0 Middleware Taxonomy

#### 2.1 Basic Middleware

The programs and databases that are part of a distributed application system (e.g., application “A” in Figure 2) may use one or more of the three different kinds of basic middleware:

- **Data management middleware** may help programs read from and write to distributed databases or files (e.g., program A’ to DB’ in Figure 2).

- **Communication middleware** may help programs talk to other programs (e.g., program A’ to program A”).

- **Platform middleware** may help programs talk to other programs (e.g., again, program A’ to program A”) and will also provide programs A’ and A” with various environmental services.
2.1.1 Data Management Middleware

Remote data access middleware and remote file access middleware are among the most widely used of all middleware. These data management services are generally not separate products; rather they are embedded in DBMSs, operating systems or similar software environments. All modern DBMSs include a remote-access capability so that the DBMS engine can (optionally) be called from a client application located elsewhere. For example, a desktop application may use an SQL language API to communicate with a set of Open Database Connectivity (ODBC) libraries which in turn leverage a TCP/IP network layer to get to a DBMS engine on another computer (see Figure 3). In some cases, a full DBMS resides on the same computer as the application, and some of the data is managed locally. In other cases, only a thin subset of DBMS processing is executed on the client side, with most of the DBMS logic running on the server. Microsoft provides ODBC and OLE DB client interface layers and also the drivers for its own DBMS and file management products. All of the other major DBMS vendors support ODBC and Java Database Connectivity (JDBC) access, as do some third-party (non-DBMS) vendors such as Information Builders, International Software Group and Merant.

Source: GartnerGroup

Figure 2. Basic Middleware

Figure 3. Remote Data Access Middleware
In addition to the basic (read-write) remote data access middleware, more-powerful forms of data middleware, e.g., database gateways, are also available from DBMS vendors and independent vendors. Database gateways enable connectivity to heterogeneous DBMS engines, sometimes including nonrelational databases, using a common API (usually SQL). These products are actually a type of integration middleware (discussed in Section 5.1) but their topology may be the same as the basic remote data access middleware shown in Figure 3.

Remote file systems, such as in NetWare, NFS, Windows and the Open Group’s Distributed Computing Environment (DCE) Distributed File System, are a little different in their flavor, although their topology is similar. A remote file system has some kind of redirector software on the client (requesting) side and a corresponding file management system on the server side. Such systems enable the sharing of small or large objects (e.g., program code and engineering drawings) or they may be used as a virtual software distribution mechanism so that each client fetches a copy of the application at runtime. Unlike databases, most remote file systems do not manage concurrent access or transactions on a record-by-record or row-by-row basis (although some record-oriented indexed file systems do).

2.1.2 Communication Middleware

The purpose of communication middleware (see Figure 4) is to facilitate program-to-program communication. Communication middleware provides an API that shields (abstracts) the application developer from having to deal with lower-level communication chores and it offers a set of services that extend the power of the communication model in various ways. Communication middleware includes MOM, RPC services and some other similar products.

![Communication Middleware Diagram](image)

Source: GartnerGroup

**Figure 4. Communication Middleware**

The services provided by middleware can be generally categorized into the session, presentation and some application layer aspects of the OSI stack (see Figure 5). The exact functions vary from product to product and from product category to product category. However, the common theme is to make development and deployment of distributed processing faster and easier. In most cases, it is uneconomical to develop an application directly (without middleware) to the network software (e.g., CPI-C, HLLAPI or Sockets) today. However, sometimes it can still be practical to avoid middleware, specifically when the task is very complex or very simple, e.g., when it only involves two parties at known network addresses and does not require DBMS access, queuing, publish-and-subscribe, load balancing, re-routing, or elaborate retries and error recovery. Note that in relationship to the network software, communication middleware is the “application.”
2.1.3 Platform Middleware

Platform middleware is a combination of communication middleware, as described above, and a variety of additional runtime services that provide an execution environment for application logic. Platform middleware includes a layer of functionality that partially or fully insulates the application program from the operating system in the same way that communication middleware insulates the application program from the network software (see Figure 6).

In addition to communication-related functions, platform middleware performs one or more resource management functions such as:

- Managing memory and operating-system processes (address spaces),
- Loading programs from disk,
- Starting and stopping programs,
• Passing messages to and from programs (within the same process, among processes within the same computer, and to programs on other computers),
• Trapping and responding to errors (sometimes including failover),
• Load balancing among programs, and/or
• Managing transactions.

Examples of platform middleware include application servers, TPMs, DBMS stored procedure services, object-oriented database management systems (OODBMSs), ORBs and OTMs. Note that the distinction between communication middleware and platform middleware is a matter of degree rather than being binary. We categorize RPCs as communication middleware, but subsystems that include RPCs may include some platform middleware services and even data middleware services. For example, the Open Group’s DCE includes not only RPCs, but also security, naming, threading, timing and distributed file services.

On the other hand, we put ORBs in the platform middleware category, although ORBs began as variations of RPCs and were used, especially in the past, simply as communication middleware (i.e., the operating system and programming language directly provided most of the resource management services). However, most ORBs could start programs even from their inception, a feature which differentiated them from most RPCs. Today’s ORBs and OTMs are much more similar to TPMs and other application servers than they are to basic communication middleware. Distributed TPMs such as BEA Systems’ Tuxedo and TopEnd were also often used primarily as messaging systems — i.e., as communication middleware — although they optionally provide many platform-type services, such as application management and/or transaction management.

2.2 Middleware for Application Integration

Most of today’s enterprise-scale business problems require that multiple independently designed application systems work with each other in some way. For example, a solution may require that new application code interact with purchased application packages and with legacy application programs and databases. This introduces additional requirements with respect to middleware usage.

An application system (e.g., application A in Figure 7) is designed by one development team and will therefore presumably be internally consistent in its choice of technology and design. All of the programs (e.g., A’ and A”) in application A will generally use the same intra-application middleware, or at least compatible middleware. These programs are also likely to use the same DBMS and operating system, and maybe even the same development language. Moreover, the programs will generally use the same data model, object model and semantics, again, because one team designs them all.
The situation changes when integrating two or more independently designed applications (as illustrated in the right side of Figure 7). When a program B’ in application system B needs to talk to a program C’ in application system C, there are almost certain to be disparities in the data formats and semantics. There are also likely to be differences in technologies, such as different operating systems, DBMSs, programming languages and middleware. Therefore, the middleware that program B’ will use to talk to program C’ will likely have some different characteristics than the middleware that it uses to talk to B” (another program in system B). Both intra- and interapplication communication paths will involve some basic middleware: platform middleware, DBMS middleware and/or communication middleware. But the interapplication (B to C) paths may also involve some “integration middleware.” The middleware for architectural-level, intra-application communication will continue to differ from the “city planning”-level integration middleware used between independently designed applications through 2004 (0.9 probability).

Integration middleware is high-level middleware that provides some function or set of functions that is specifically related to connecting application programs or middleware products that differ from each other. Integration middleware complements, and sometimes even includes, basic communication middleware, platform middleware or data management middleware. Integration middleware performs one or more of the following four roles (see Figure 8): gateway, superservice, integration broker or business process manager. However, commercially available integration middleware does not fall into neat categories. Some products are just gateways; others are gateways and superservices; others are superservices and integration brokers but have few or no gateways; and so forth. This area of technology is immature and dynamic. Vendors and users are still experimenting to find better ways of packaging and delivering the middleware functions needed for integration. It is much harder to classify and sort integration products than it is to classify basic middleware products, although even basic middleware has some internal areas of overlap.


2.2.1 Gateways

A middleware gateway is software that translates among two or more protocols. It intercepts the communication after it has been sent and converts it into a form that is understandable by a dissimilar recipient. Gateways are available for all three kinds of basic middleware:

- Database gateways are offered by all of the major DBMS vendors and also by specialist, non-DBMS vendors such as Cross Access, Information Builders and Neon Systems (not to be confused with New Era of Networks, also called “NEON”).

- MOM gateways are widely available, particularly for the better-known MOM products. For example, there are gateways between IBM’s MQSeries and just about every other MOM product on the market.

- Platform middleware gateways connect all of the major application server protocols such as CICS, COM, CORBA and Java Remote Method Invocation (RMI). For example, there are COM-to-CICS, CORBA-to-CICS, COM-to-CORBA and Tuxedo-to-CICS gateways. This kind of gateway is usually accessed through wrappers or adapters rather than directly from an API (see Section 5.0). Some of these gateways are written by the platform middleware vendors themselves. Others are written by independent companies, e.g., Insession’s Transfuse (www.insession.com).

Some gateways reside on the same computer as one of the applications (see the left side of Figure 9). Others reside on a separate computer, so the first segment of a message’s journey is conducted using a different protocol than the second segment (see the right side of Figure 9). These are just two of the possible ways to package a gateway.
Figure 9. Alternative Topologies for Middleware Gateways

Some vendors use the term “bridge” to describe a protocol-oriented (i.e., session, presentation and some application level features) gateway (e.g., a COM-CORBA bridge). They may then use the term “gateway” to apply to some higher-level translation service, e.g., a wrapper that deals with some business application semantics. However, in communications, the term “bridge” typically describes a connection way down the OSI stack (e.g., at the data link or network level), and typically implies that the upper-level protocol is compatible while the lower-level protocols may or may not be the same. In the spirit of communications vocabulary, then, we prefer the general term “gateway” for all translation between heterogeneous middleware because it is all at upper OSI layers (i.e., application, presentation and session layers).

2.2.2 Superservices

A superservice is “super” in the sense that it goes above or beyond one or more lower-level services in the underlying software. A superservice presents to the application program its own super-API, effectively masking or superseding the API(s) exposed by other software layers. A superservice may provide some common services, such as directory, transaction management or security, across two or more operating systems, ORBs, TPMs, DBMSs, application servers and/or networking layers.

For example, a superservice can enable an application to use multiple underlying MOM products through a common API (see Figure 10, left side). The superservice can “normalize” the communication semantics among different products by hiding some features found in only one product or by filling in the gaps that are missing in the other product.

A superservice may support one or more platform or data management functions. A superservice can protect against change, so that, taken to an extreme, the application could theoretically be ported to different middleware or even to a different operating system without changing the application source code (see the right side of Figure 10). The use of a superservice is explored in more depth in Section 5.3.
2.2.3 Integration Brokers

An integration broker is a third-party intermediary (hence the term “broker”) that facilitates interactions among application systems (see Figure 11). A broker is not communication middleware, nor a gateway or superservice, although most broker-based products have one or more of those bundled in. Some brokers require that you use a particular MOM (e.g., IBM MQSeries), while others have interfaces to a wide variety of software environments.

By definition, the broker itself provides two value-added application-layer functions:

- **Transformation** — translates message or file contents, including both syntactic “conversion” and some degree of (greater or lesser) semantic “transformation”; and
- **Flow automation** (or flow control) — some form of smart addressing, such as content-based routing and/or publish-and-subscribe.

Some brokers also provide other services such as:

- Business process management (e.g., workflow) to manage each instance of a multistep procedure (see Section 2.2.4);
- Message warehousing, i.e., storing messages to be retransmitted or analyzed at a later time; or
- Application-specific or technical adapters, along with some related development tools, gateways and templates for connecting to application systems.

To enable these services, a broker will usually have some (strong or weak) form of:

- Repository to hold the metadata descriptions of the input and output message formats (i.e., a message dictionary), the transformation rules and the routing rules; and
- Administration and monitoring facilities to manage the broker configuration.
Integration brokers may be packaged as a separate software facility (as in Figure 11) or they may be collocated with an application server (as in Figure 12). Brokers may run directly on the operating system or they even may be hosted by platform middleware such as a TPM or ORB. However, brokers themselves are not platform middleware, i.e., are not fully functional as a runtime environment for business logic, although they maybe tied into a general-purpose platform middleware product. Applications can connect to a broker invasively through a super-API or noninvasively through wrappers. Figure 11 and Figure 12 are examples of broker configurations, but there are many other variations not shown here.

An integration broker is not an ORB, although both use the word “broker.” An ORB is platform middleware that supports runtime application programs, whereas an integration broker by itself is not a good general-purpose business application server. On the other hand, traditional ORBs do not have the transformation services or sophisticated flow automation services of an integration broker (i.e., ORBs typically use relatively simple request/reply communication semantics).

Many variations of broker have emerged during the past 10 years using labels such as “message broker,” “integration engine,” “interface engine,” “data broker” and “integration hub.” In earlier GartnerGroup research, we used the term “message broker” to describe any broker that supported program-to-program communication. However, this proved confusing to some clients who assumed that message brokers only interfaced to MOM. Another terminology problem arose with “data brokers,” a term that is sometimes applied to brokers that read and write files and/or databases. Now that many data brokers talk to MOMs...
and virtually all “message brokers” talk to DBMSs and files, they are all “integration brokers.” However, brokers still vary in their capabilities, with some better suited for a particular problem than others. For example, some brokers are optimized for handling individual messages whereas others are optimized for processing batches of transactions and database tables.

2.2.4 Business Process Managers

Business process managers change the notion of application integration from one of managing interactions (e.g., making routing decisions) to a more sophisticated perspective that involves coordinating an entire multistep business process. They track and direct each instance of a business process, such as each individual order or medical insurance claim, through a life cycle that may consume seconds, minutes, hours, days or weeks. Unlike simpler forms of flow automation, a business process automator “remembers” (i.e., maintains context information in a persistent file or database) for the duration of a process instance that spans many individual activities.

Process managers come in several forms and go by many labels. Traditional “workflow” and “enterprise work management” products focus primarily on human-executed work steps, using features such as work lists, organization models, role management and document and image handling. Other, more recent, “business process automation” or “businessware” products focus mostly on programmed (computerized) task steps and lack features for direct human contact. Some of the new, richer “workflow broker” products are getting good at handling both human and automated steps.

Vendors such as Action, Compaq (Digital), Eastman, FileNet, Forte, Fuego, Hewlett-Packard, IBM, ICL, InConcert, PegaSystems, Plexus, Staffware, Universal Systems and Vitria have offered various process managers for some time. Many other vendors, such as Neon, are now bringing similar capabilities to market in their integration brokers.

The topology of a business process manager is comparable to that of the integration brokers in Figure 11 and Figure 12. Most of these products would even qualify as a special form of integration broker because they support transformation and intelligent routing (the two definitional qualities of an integration broker). However, integration middleware is complicated because there are so many different ways to combine and package the functions described in the preceding four sections (gateways, superservices, integration brokers and business process managers). The configuration and use of brokers is explored further in Section 5.5.

3.0 Evolution of Application Platforms

This section reviews three separate issues related to the evolution of platform middleware: the changing role of the DBMS as platform middleware, the distinction between opportunistic and systematic application platforms, and the increasing use of components.

3.1 DBMSs as Application Platforms

The first popular generation of client/server applications (1986 to 1994) was largely enabled by DBMS middleware. Originally, these applications only used remote data access topologies, i.e., all of the application logic was on the desktop and the server simply ran the database engine (see Figure 3 in Section 2.1.1). However, DBMS vendors, beginning with Sybase, then introduced stored procedures as a way of hosting some of the application logic on the server.

DBMS stored procedures are a combination of platform middleware and DBMS middleware. On the client side, they somewhat resemble remote data access middleware (as in Figure 3) because they do not provide platform services to the client program. The API to the client program is semantically an RPC
(i.e., program-to-program and request/reply) rather than a traditional data manipulation language (DML) verb, although the DBMS uses the same transport protocol for a stored procedure as it uses for DML (i.e., remote data access SQL statements). On the server side, stored procedures are platform middleware (see the left side of Figure 13), hosting user-written application logic in a manner similar to that of any application server (e.g., ORB, TPM, OTM or Web application server).

**Figure 13. Variations of Platform Middleware**

Stored procedures are widely used on Unix, NT and, to some extent, the AS/400. On the mainframe (e.g., in IBM's DB2), a particularly robust form of stored procedure is available, although it has not been widely used as an alternative to CICS because CICS had such a large head start in applications, tools and installed base. Until recently, most stored procedures were limited in the choice of programming language that was supported, i.e., they only supported a DBMS vendor’s proprietary 4GL (some now support Java as an alternative).

DBMS-enabled, two-tier architectures, with or without stored procedures, are most often applied to applications with relatively modest business requirements such as:

- Fewer than 100 users
- A single, relational database
- Availability limited to five or six days a week and less than 24 hours per day
- Less than 50 application programs
- Connectivity to other systems mostly in the form of nightly batch file transfer

During the past five years (1994 to 1999), multitier (three or more tiers) architectures began to supersede two-tier architectures in an increasing number of client/server applications. GartnerGroup has observed the uptake of multitier architectures mostly among larger, more demanding distributed applications where two-tier architectures do not perform well, and among Internet or intranet applications. More than 90 percent of Internet and intranet applications use three or more tiers, although it is technically possible to run them with a two-tier topology. Overall, 90 percent of new Web and other client/server applications will be multitier (three or more tiers) in 2001, up from 55 percent in 1998 (0.8 probability).

In a three-tier architecture, some or all of the server business logic is hosted in a platform environment outside of the DBMS (e.g., in an application server, ORB, OTM or TPM) although a DBMS is always still
part of the solution, to provide data management services. In some cases, DBMS stored procedures are still used to host some of the logic, complementing an external application server in a multitier architecture. Furthermore, DBMS stored-procedure facilities are also gradually acquiring some of the characteristics of OTMs such as support for components (see Section 3.3).

There are also two other related forms of DBMS-based application servers: OODBMSs and object-relational platforms. Both of these provide transparent management of object state and automatic persistence of that state. This is a more fully object-oriented style of development and execution. It has been enabled by OODBMSs for years (see the left side of Figure 13). A variation on this is delivered by some modern OTMs that do transparent object-relational mapping between the in-memory object state to and from a relational DBMS (see Figure 13, right side). As in a pure OODBMS, the task of issuing DML statements to read and write data is removed from the application code because it is managed by the middleware environment.

Examples of object-related application platforms include:

- Gemstone Systems’ GemStone/J (www.gemstone.com)
- Lockheed Martin’s HARDPack (www.owego.com/hardpack)
- Persistence Software’s PowerTier (www.persistence.com)
- Template Software’s Enterprise Integration Template (www.template.com)

Furthermore, the application servers from IBM, Oracle, Sybase and others (notably including Microsoft’s upcoming Windows 2000 COM+) also are becoming object-oriented hybrids. Their internal designs are focused on optimizing the interaction of the application server and its respective DBMS engine. In a Java environment, “entity beans,” i.e., those with container-managed persistence, would fit in this category.

3.2 Platform Middleware Market Requirements

Platform middleware is a combination of communication middleware and a variety of resource management services that support an application program at runtime. Application servers are a type of platform middleware that interacts with intelligent requesters — e.g., Web browsers, client/server desktop programs or peer server programs. In contrast, the early TPMs were a type of platform middleware meant to interact with nonprogrammable devices (dumb terminals). Today, all TPMs — indeed, all platform middleware — can support intelligent requesters and thus all platform middleware is a form of application server (although TPMs such as CICS often still run a mix of dumb terminals and intelligent requesters).

The role of an application server is to provide access to units of server-based application business logic (i.e., programs). The less-powerful application servers simply assure availability of the runtime “executable” that performs the required business function. Their primary function is to facilitate the start and stop — and the restart after failure — of the runtime application executables. More powerful application servers provide access to business functions on the server with better quality of service; i.e., they keep going despite hardware or software failures, despite increasing numbers of simultaneous requests for the same function, despite a distributed or heterogeneous nature of the transaction, and despite security and privacy restrictions. The most powerful application servers intercept all interaction between requesters and servers and present to the application a high-level programming model that is designed to facilitate a productive business application environment.

The application server market is composed of two segments: systematic application servers and opportunistic application servers (see Figure 14).
3.2.1 Systematic Market Requirements

The systematic market is characterized by application projects that are considered strategic by the central IS organization. These projects are expected to have a long-term impact on the enterprise. These projects are usually well-funded and are preceded by extensive research, product selection and design. After development, the systematic applications normally become subject to a systematic maintenance process, including quality assurance, managed rollout, change management, and ongoing support by full-time or part-time dedicated technical staff.

In the systematic market, the long-term viability of the technology and the vendor (read: low risk) are key selection criteria. Equally important are mainframe-grade “quality of service” characteristics, such as availability, performance, manageability and integrity. A successful technology vendor in the systematic market sells a relationship, not just a software product. This includes worldwide support and professional services, as well as ongoing involvement of the vendor with the enterprise’s IT initiatives. Users in the systematic market are expected to pay extra for the safety and breadth of the relationship with the technology vendor, and thus the prices of the systematic products are usually relatively high. A typical contract in the systematic market is in the range of $300,000, usually including software, support and some professional services upfront.

Prior to the introduction of component software (see Section 3.3), the systematic market was addressed primarily by TPMs including:

- BEA Systems’ Tuxedo (www.beasys.com)
- Hitachi Computer Products’ Open TP1 (www.hitachi.com)
- IBM’s CICS family, Encina, Information Management System/Transaction Monitor (IMS/TM), Transaction Processing Facility (TPF) and TxSeries (www.ibm.com)
- Siemens AG’s openUTM (www.siemens.com)
- Unikix Technologies’ UniKix (www.unikix.com)

The systematic application server market is dominated by IBM’s CICS (on the mainframe). On other platforms, BEA’s Tuxedo is the leading player.
3.2.2 Opportunistic Market Requirements

The opportunistic application server market, by contrast, consists of products that emphasize fast learning (for developers), rapid results in application development, reduced technical-skill requirements for users, and reduced preliminary design work or post-development administration. These products also include support for modern technology features and, last but not least, a low cost of entry for business users. Many technology contracts in the opportunistic market start with initial costs in the range of $30,000, including the development, the runtime tools and some technical support.

Opportunistic application projects are driven by the business requirement to urgently take advantage of a new opportunity or to respond to some competitive pressures. Often, these projects are not funded centrally, and generally the lifetime of the application is not clear when the application is conceived and deployed. While few would admit that scalability, availability and integrity are not important to them, the opportunistic projects usually settle for moderate scalability and availability to get the productivity. Some of the opportunistic applications become mission-critical over time, but they still remain opportunistically built.

Due to the nature of the opportunistic market, it is characterized by a large number of vendors, from recent start-ups to some more mature vendors and products. Many features and differentiating capabilities are offered, and the life span of an independent vendor in this market is often less than five years before the product is acquired, repositioned or simply discontinued. In such an environment, vendor viability is limited; mergers, acquisitions and new vendor start-ups are an ongoing process in the opportunistic market. For that reason, the spectrum of maturity and focus of opportunistic products varies much more than in the systematic market.

Stored procedures are a type of opportunistic application server, as are the object-oriented DBMS platforms described in the previous section. Other non-DBMS, opportunistic application servers can be separated by their initial vision and core technical competence into two groups:

- The first group emerged to mitigate the problems of the Web CGI protocol. These are products whose core competence is focused on integration with the Web client style and support for HTTP/URL/HTML and other Web-standard protocols. These Web application servers are the majority of the current opportunistic market (not counting DBMS stored procedures).

- The second group is that of distributed Java platforms. These products are primarily Java ORBs and only secondarily Web application servers. Their core model assumes that the requester-server communication always occurs between Java programs (true Web application servers are built for direct connection from a browser into the application program). As Java is a highly technical development environment, requiring far more technical skill than most 4GL and scripting environments, the Java ORBs tend to have a systematic flair and appeal to more technically sophisticated projects.

Examples of non-DBMS-based, opportunistic application servers include:

- BEA Systems’ WebLogic Server (www.beasys.com)
- Bluestone Software’s Sapphire (www.bluestone.com)
- Haht Software’s HAHTsite System (www.haht.com)
- Metaserver’s Metaserver (www.mserver.com)
Middleware: The Glue for Modern Applications

- Novera Software’s Application Server (www.novera.com) — although this is currently positioned more for integration purposes than for hosting new application logic
- ObjectSpace’s VoyagerPro (www.objectspace.com)
- Oracle’s Application Server (www.oracle.com)
- SilverStream Software’s SilverStream Application Server (www.silverstream.com)

The opportunistic market cannot be dominated by any one product for long, given the continuous innovation and retooling that it requires. It is important to note that some of these products have evolved over time to offer notable systematic features, and while most have not yet established the required credibility and enterprise production record, technically they qualify to be placed in the systematic technology market. These include application servers from Sun-Netscape Alliance (NetDynamics and the former Kiva product), BEA/WebLogic, Oracle, Bluestone, Inprise and Forte. Some of these products are capable of supporting concurrent users numbering in the high hundreds or even thousands, though in most cases these numbers are achieved only when using the particular application style for which the product is optimized. This differs from the core systematic products that support similar numbers of concurrent users across a variety of architectural styles for the application.

The distinction between the systematic and the opportunistic markets is not a temporary phenomenon. As new technologies and architectures penetrate the core enterprise computing systems, some fail to qualify and disappear and others become part of core enterprise computing. Effectively, the conservative systematic market acts as the barrier, protecting the core enterprise resources from market hype and failed technology directions. Meanwhile, new technologies emerge and, driven by vendors’ innovation and business competition, replace the previous generation of the opportunistic leading edge. XML-based application technologies, as well as application integration and dynamic networking technologies, are beginning to directly compete against the maturing Web application server products.

Most users will find that no single application server can meet the conflicting requirements of their enterprise’s computing portfolio. Increasingly, users recognize the need to combine a systematic and an opportunistic technology to provide integrity for their core systems and dynamic vigor for their competitive use of technology. While some users will find a single platform suitable for both application styles, most will not and all should consider the systematic and opportunistic projects separately when making decisions for their enabling technologies.

3.3 The Impact of Component Software

As software gets bigger and more complicated, developers have an increasing need to break big problems into multiple smaller problems that can be solved separately. The core insight of component software is the importance of modularity. Reuse is an important corollary to modularity. The “plug and play” nature of components enables some aspects of applications to be assembled from interchangeable parts. Encapsulation (data hiding) is another corollary: components interact through program-to-program communication, not by sharing data. Component middleware (i.e., ORBs and OTMs) is naturally well-suited to hosting applications that have many “moving parts” because they document program-to-program interface definitions and manage the connections (object references).

The three major component models today are Microsoft’s COM, the Object Management Group’s CORBA and Sun’s Java RMI. Virtually all new platform middleware products, including ORBs, Web application
servers, OTMs and, increasingly, even DBMS stored procedures, support one or more of these component models. *More than 80 percent of large enterprises will use COM and some aspect of CORBA or IIOP somewhere in their enterprise applications by 2001 (0.8 probability).*

In 1996, GartnerGroup projected: *OTMs, a new class of middleware, with the combined functionality of TPMs and ORBs, will be functionally complete by 1998 and mature by 2001 (0.8 probability).* GartnerGroup’s view continues to be that middleware vendors will converge the previously distinct TPM and ORB technologies into OTMs. This started unfolding in earnest in 1998 and is continuing in 1999. TPMs are acquiring component-style interfaces; TPMs and ORBs are gaining improved Web support; and ORBs and Web-oriented application servers are getting transaction management and other mission-critical “quality of service” features, such as load balancing and failover. In other words, they are all transmuting toward full-function OTMs.

Note that component technology is being applied to both opportunistic and systematic products, although its initial use has been broader in the less-demanding, opportunistic arena. TPMs are likely to do well, at least in the near term. TPMs represent a blend of the new and the old. They enable the modern, distributed multitier architectures, but they have long been proven in mission-critical applications. Although some TPMs are adding limited support for components (particularly Enterprise JavaBeans), TPM applications generally still use a procedural programming style, in contrast to the component interfaces of the ORBs and OTMs. TPMs will likely be concentrated in higher-end business applications with demanding requirements such as many concurrent end users, very high up-time, and broad distribution over many locations. We expect that TPMs will lose market share for new applications during the next five years as OTMs become better and more proven, working their way up from less-demanding applications. *By 2000, OTM-based components and frameworks will enable development of three-tier applications that are as intuitive, easy to use and effective as the current two-tier technologies, making OTMs the mainstream technology (0.8 probability).*

The two biggest obstacles to enterprises adopting ORBs and OTMs have historically been concerns about manageability and security. We believe that enterprises must expect to continue to have some difficulties in these areas during the next several years, particularly when dealing with the less mature products, but ORBs and OTMs are evolving quickly. TPMs generally tend to be more mature and more complete in their management facilities than those other forms of middleware, although management and security will not be trivial in any large distributed application regardless of the middleware products that are used.

Some examples of ORBs and OTMs are:

- BEA Systems’ WebLogic Enterprise and Object Broker (www.beasys.com)
- Hitachi Computer Products’ TP Broker (www.hitachisoftware.com/tpbroker)
- Information Builders’ Enterprise Component Broker (www.ibi.com)
- Inprise’s Visibroker family (www.inprise.com)
- Iona Technologies’ Orbix (www.iona.com)
- Microsoft’s COM/DCOM/MTS/COM+ (www.microsoft.com)
- PeerLogic’s (ICL’s) DAIS (www.peerlogic.com)
4.0 The Role of Communication Middleware

Communication middleware is used in situations where the program-to-program communication facilities that are embedded within a program’s runtime environment (i.e., its platform middleware and/or operating system) are not sufficient.

Separate (unbundled) communication middleware is usually not needed for intra-application purposes if platform middleware is present. Platform middleware, including application servers, TPMs, ORBs, OTMs and DBMSs (with their stored procedures and other middleware services) support good program-to-program communication within a single computer and for limited intercomputer exchanges. For example, these products have good ways to connect desktop PC clients to servers, even if the servers are running a different operating system. They are also fine for connecting programs on one server to programs on another server, as long as all programs are using the same platform middleware product (which they usually are when all programs are within the same application system).

However, separate communication middleware may be helpful for:

- **Intra-application program-to-program communication** when there is no platform middleware (i.e., when the programs run directly on the operating system). Although operating systems have interprocess communication services, these are unfriendly even for local interchanges and are particularly poor at communication between programs on different computers. RPC communication middleware was invented specifically to address this need.

- **“Extended” systems** where components of the application system run on multiple different operating systems and the developer does not want to use the same platform middleware on every system. For example, some application suites began life on a mainframe and were extended to desktop PC clients and Unix and NT system clients and servers. The developer may prefer to use CICS on OS/390 and different platform middleware on NT and Unix. Communication middleware can be used as a common denominator across all environments rather than using a collection of gateways between the platform middleware products.

- **Interapplication (integration) scenarios** where the application systems are dissimilar in their technology (e.g., operating system, DBMS and platform middleware), their design (e.g., dissimilar data formats and semantics) and their management (e.g., managed by different departments or running at different times of the day). In these situations, communication middleware is used, often, but not always, supplemented by integration middleware.

There are three basic types of communication middleware: messaging systems, RPC services and multipurpose products that offer some messaging-related, conversational and/or RPC functions. To help understand when and why these are appropriate, Section 4.1 reviews the inherent characteristics of program-to-program communication.

4.1 Characteristics of Communication

4.1.1 Connection-Oriented and Connectionless Relationships

Some middleware is connection-oriented, meaning that software bound into the sender works directly with software that is bound into the receiver. Each is aware of the other, and they participate in mutual “handshaking.” By contrast, connectionless communication means that the sender and receiver have no
direct relationship with each other; rather, each connects to some intermediate piece of middleware software which knows the actual network addresses and does much of the communication-related housekeeping and message routing. Connectionless communication implies that there is a period of time in which the message is in the hands of an intermediary, rather than with the sender or the receiver. Note, however, that connections may exist at different layers of the communication stack. RPC middleware is connection-oriented with respect to the application programs and their local communication software, although RPCs can run over either a connection-oriented transport (e.g., TCP/IP) or a connectionless transport (e.g., UDP/IP). This contrasts to MOM in which the applications are connectionless with respect to each other, but which may also run over either a connection-oriented transport or a connectionless transport.

4.1.2 Synchronous and Asynchronous

Communication is often described as synchronous or asynchronous in nature, although these terms are problematic because they are used to mean different things. There are two major issues here:

- Are the sender and receiver active simultaneously? Communication between concurrently executing programs implies that the message is delivered immediately. If, on the other hand, the parties are not running at the same time, the middleware must support deferred delivery (i.e., queuing) so that the receiving program can get the message when it runs at a later time. For example, E-mail, messaging-and-queuing MOM, and most database replication middleware can defer delivery.

- Does the sender block or not? Blocking means that the requesting application sends its message and then stops processing while it waits for a reply in the form of data or an acknowledgment. Nonblocking means that the requesting application continues to do other tasks while waiting for a reply. Communication between programs that are not concurrently executing must always be nonblocking, because the sender cannot wait for a reply from a program that is not running (or is not immediately started).

Some people use the term “synchronous” to mean “blocking” while others use it to mean “concurrently executing.” This ambiguity is the source of considerable confusion. Since “blocking” always implies “concurrently executing,” the terminology conflict only arises for the case of nonblocking-but-concurrent execution. To resolve this ambiguity, we have adopted the following definitions:

- “Fully asynchronous” means nonblocking/nonconcurrent.
- “Partially asynchronous” means nonblocking but concurrently executing.
- “Synchronous” means blocking/concurrent.

4.1.3 Communication Patterns

Developers have five fundamental communication patterns (or “models”) from which to choose when writing applications that require interactions between programs (see Figure 15). The conversational and request/reply models are two-way communication, whereas the message-passing, messaging-and-queuing and publish-and-subscribe models are all forms of one-way messaging. In addition, there are some compound variations (e.g., publish-and-reply) which are not described here.
Conversational: In the conversational model, participating programs interact with one another in the form of a dialogue, much like a telephone conversation, with each program responding in turn to information received from the other program. Both parties (programs) must be available at the same time for the conversation to take place, so they are never fully asynchronous. The sender establishes contact with the receiver, sends and receives messages, and (usually) explicitly terminates the conversation. Conversational interactions are usually blocking, but they can be nonblocking. This communication model is usually implemented using low-level communication APIs such as APPC or CPI-C (i.e., APIs that belong to network software rather than to middleware). Conversational programming is used more often by system software (e.g., inside middleware itself) than by business application programs. However, a few middleware products, such as BEA’s Tuxedo, support conversations that are used infrequently, but nevertheless are occasionally important, e.g., for multistep transactions or for returning large result sets in batches from a query service.

Request/Reply: The request/reply (or “request/response”) model is also two-way and interactive, but it differs from a conversation because it is based on a single request and a single return. A program sends a request message and a receiver (server program) performs the desired function and returns some result. The sender is usually blocked (it is usually fully synchronous). A partially or even fully asynchronous operation could arguably be considered a request/reply if there is one input and one matching response, but a fully asynchronous request/reply pattern (in the eyes of the application) would usually not be handled as a request/reply by the middleware. Rather, this would traditionally be implemented using what appears to the middleware to be two unrelated one-way operations (i.e., the application has the job of matching the reply to the request).

Message Passing (or “Messaging”): This is a simple, one-way transfer. The sender is generally unblocked — it does not need to wait because there will be no response (note, however, that the application program may receive some acknowledgement that the message has been successfully sent).
The majority of messaging is a one-to-one operation, but messaging can be a broadcast to all listeners or a multicast to a certain set of receivers.

**Messaging-and-Queuing:** The messaging-and-queuing, or “store-and-forward,” model is based on intermediate message storage. A queue is a database of messages in transit. The sending application sends the message to the messaging middleware, which places it in a queue that may reside on the client system, the final destination system or on another node in the network. Message delivery can be deferred; depending on the application requirements, it may occur within a second or not for weeks. The sender is generally unblocked. Queues may be in memory — in which case, they will be fast, but vulnerable to losing messages in the event of a power loss — or they may be persistent (written to disk), which is slower but more reliable.

**Publish-and-Subscribe:** This is a communication pattern in which information sources “publish” (i.e., send) information to a somewhat intelligent middleware infrastructure, and information consumers “subscribe” by specifying what kind of information they want to receive from the infrastructure. The middleware must be able to physically transport messages from one or more publishers to one or more subscribers. It must also be smart enough to find the proper destinations by matching each message to subscription criteria. This model naturally supports one-to-many or many-to-many communication in contrast to either message passing or message queuing, both of which mostly (but not entirely) aim at one-to-one communication. In addition, senders and receivers in the publish-and-subscribe model sometimes identify themselves dynamically by changing their subscription rules as they run.

Large enterprises are likely to use most if not all of the five communication models (or “patterns”) described here. No single model is suitable for all application types. Developers should understand what communication options are available and select appropriately, rather than continually relying on the model with which they are most familiar. Note that there are numerous variations on each of these five basic patterns.

The communication pattern is an issue separate from the middleware used to implement that pattern. This fact is a source of much confusion. The *pattern* describes the logical relationship between the communicating parties, not the technical details of how it is implemented. RPCs and ORBs are usually equated with the request/reply pattern, but they can support a request with no reply. That is logically message passing in the eyes of the sending application logic because it is a one-way operation (although the syntax of a one-way RPC message will be different than that of a MOM-enabled message). A one-way logical operation using a traditional RPC or ORB could be termed “pseudo-asynchronous” because its underlying RPC or ORB middleware is typically (although not exclusively) connection-oriented and fundamentally synchronous (i.e., the application logic does not require that the recipient be active simultaneously, but the underlying middleware does require this).

Similarly, MOM is sometimes equated with message passing, yet more than one-third of its use today is for two-way request/reply interactions. MOM manages the reply by using a separate send/receive message and automatically matching the returning message with the outgoing message, mimicking the apparent behavior of “RPC” software but with a different syntax. A request/response implemented on top of MOM may be termed “pseudo-synchronous” because the application sees a synchronous relationship (e.g., the sender may block while waiting for a return message), even though the underlying MOM may be considered connectionless and asynchronous in a different sense.

Note that publish-and-subscribe is a high-level concept that can be implemented over many different kinds of middleware products. Furthermore, most MOM that supports publish-and-subscribe messaging also supports plain, directed (one to one) message passing, queuing and request/reply. Ultimately, most
Middleware can enable most of the communication models, although they differ in how well they support each of them. In particular, the connectionless nature of MOM, some TPMs and some OTMs has different implications than the connection-oriented nature of most RPCs, most ORBs, some TPMs and certain other middleware. (For more information, see SSA Research Note SPA-401-229, 20 May 1997.)

4.2 What Happened to RPCs?

RPCs were the first significant form of program-to-program middleware on nonmainframe platforms (TPMs were the first major form of middleware on mainframes, emerging in the late 1960s). However, RPCs never dominated business applications and they did not endure as a major middleware product category in their own right. Few new applications are written directly to RPC middleware today. However, they are widely used as a layer of functionality embedded in other products. RPCs are also significant as the forerunner of ORBs and OTMs.

The original “RPC” was Sun’s Network File System (NFS) RPC, also called Open Network Computing (ONC) RPC or Transport-Independent (TI) RPC in its various incarnations. It originated in the mid-1980s and it is still available on most operating systems, along with NFS, as an extension of the TCP/IP stack, although its commercial use is gradually fading.

The other famous RPC was The Open Group’s DCE RPC. It is still available from IBM and other major vendors as an embedded system software layer utilized by a few high-level subsystems (such as IBM’s Unix- and NT-based WebSphere application server, formerly called Transaction System, and earlier called Encina). Microsoft’s MS-RPC, a proprietary, reverse-engineered version of the DCE RPC, serves as the foundation for many other layers of Microsoft middleware, including Distributed COM (DCOM). Inprise also still offers Entera, which is based on the DCE RPC. Versions of the DCE RPC are therefore surprisingly widespread, although it is largely invisible and rarely directly used by new business applications because of its complex, although powerful, API set.

Both the NFS RPC and the DCE RPC failed to keep up with the evolution of application requirements. Neither added native, component-based interfaces, as found in ORBs and OTMs; nor transaction management, as found in TPMs and OTMs; nor asynchronous messaging, queuing and publish-and-subscribe, as found in MOM. Such features were occasionally added on top of these RPCs, but the RPCs themselves generally stood still as the world passed them by.

Many other forms of middleware have implemented the equivalent of RPCs. Sybase no longer actively markets its highly functional, previously popular Open/Client-Open/Server RPC. Platform middleware, including application servers, TPMs (e.g., CICS and Tuxedo), DBMSs (e.g., stored procedures), ORBs and OTMs (e.g., Iona’s Orbix), bundle in their own request/reply communication middleware which, while not actually called “RPCs,” acts very much like them.

Microsoft’s Remote Automation facility is effectively a limited-purpose RPC. Designed as an extension to the Visual Basic (VB) language, it is simple to implement. In view of the fact that the majority of Microsoft client applications are written in VB, Remote Automation is, in practice, the most popular Microsoft middleware for distributed computing (outside of the HTTP applications).

GartnerGroup is not aware of any actively marketed independent (unembedded) RPC products aimed at new applications. The few remaining vendors of unembedded RPC technology now focus on security, PC-to-mainframe connectivity or RPC-related development tools. Extant RPC products include:

- Entera, a DCE-based middleware and development suite from Inprise (www.inprise.com)
- PC-DCE, a PC version of DCE from Gradient Technologies (www.gradient.com)
• Rogue Wave/NobleNet RPC, formerly EZ-RPC, offered by Rogue Wave Software (www.roguewave.com/noblenetcom.html)

• Transaccess, an RPC and development tool for Windows, OS/390 and Unix, from Proginet (www.proginet.com).

4.2.1 Examples of MOM and Other Communication Middleware

The popular standalone (unembedded) communication middleware products on the market today are either MOM or a combination of messaging and other forms of middleware. Unlike RPCs, these products do not compete directly against platform middleware for routine, synchronous, intra-application roles. Rather, they complement platform middleware by providing features that are normally missing from platform middleware. Their strengths are in connectionless communication, “store-and-forward” (queuing), guaranteed delivery, one-to-many distribution, very broad platform support (many operating systems), and, in some cases, content- or subject-based addressing (e.g., publish-and-subscribe). Such features are helpful for some intra-application and many interapplication roles.

Examples of MOM include:

• BEA’s MessageQ (www.beasys.com)
• IBM’s MQSeries (www.ibm.com)
• Iona’s OrbixTalk and OrbixNotification (www.iona.com)
• Level 8’s FalconMQ and X-IPC (www.level8.com)
• Microsoft Message Queue Services, or “MSMQ” (www.microsoft.com)
• Modulus’ InterAgent Toolkit (www.modulus.com)
• NetSys Technology Group’s (formerly Verimation’s) VCOM (www.netsys.net)
• Oracle’s Advanced Queuing (www.oracle.com)
• PeerLogic’s Pipes (www.peerlogic.com)
• Talarian’s SmartSockets (www.talarian.com)
• Tibco Software’s Rendezvous (www.tibco.com)

Multifunction communication middleware products are a blend of MOM and related services. They generally support synchronous and asynchronous communication, in some cases including queuing and in some cases including conversational styles. Examples of such commercially available products include:

• Precise Connectivity Solutions’ Comm/Booster (www.precise-conn.com)
• Software AG’s EntireX Message Broker and EntireX DCOM (www.softwareag.com)
• Vertex’s Netweave (www.vetx.com)

5.0 Integration Middleware

Components and other programs that are part of the same application system communicate with each other differently than the way they communicate with programs and databases that are in separate applications. Intra-application communication has consistent data formats, semantics and software
technology. Basic middleware, including application servers, OTMS, ORBs, TPMs and remote data management middleware, is optimized for this purpose. On the other hand, interapplication communication may need to accommodate diversity in data formats, semantics, protocols, programming languages, DBMSs and operating systems. Programs that are in independently designed applications relate to each other in an “arms-length” fashion, which may utilize integration middleware in addition to basic middleware. We recommend that enterprises have two separate middleware strategies: one for interapplication purposes and one for intra-application purposes.

Note that it is possible to integrate heterogeneous applications using basic middleware without any integration middleware. For example, in Figure 16, Program B’ is coded directly to two middleware products. B’ can “speak intra-application” to B” through its common platform middleware (e.g., Java RMI, if both are written in Java) and “speak interapplication” to a Unix or mainframe application C’ through communication middleware (e.g., IBM MQSeries). However, in many situations a direct, custom interface is difficult and time-consuming, because the developers must code all the functions that could be performed by gateways, superservices, brokers and process managers. The developer’s task may be shorter and simpler if wrappers or some form of integration middleware are used. These are described in this section.

![Figure 16. Heterogeneous Applications Using Basic Middleware](image)

**5.1 Supporting Extended and Composite Applications**

Plain platform middleware is primarily aimed at enabling new application systems. However, enterprises often seek to extend an existing application (see Figure 17) with a new user interface (e.g., Web browser or Windows client/server GUI), and/or some new application logic and possibly even a new database or file. An extended application could use a PC front end added onto a mainframe CICS application to provide a “user-friendly” graphical interface and some minor additional business functions. Although this may use a middleware gateway, we would not call this a true integration problem because the add-on
portion can be designed and written with knowledge of the original mainframe application’s semantics, thereby avoiding some of the problems that arise from independently designed “composite” applications.

Figure 17. Simple Use of Integration Middleware

In a composite application (see Figure 18), two or more independently designed application systems are “glued together” to create a new “virtual” application. When there are more than two application systems, or when there are many complicated, changing interfaces between the application systems, developers are increasingly using a multilayer design pattern that puts an “integration application” or broker of some sort in the middle. This could be called “three-tier” integration, but the term “three-tier architecture” has been used in another sense for years so the usage could cause confusion. A popular current example of this multilayer pattern puts an application on a Web application server to mediate between the Web clients (coming in through a Web server) and one or many back-end application systems. The same design pattern was previously used in some sophisticated traditional Windows client/server applications. Some middle-layer integration applications can even be accessed both from a browser and a PC client/server desktop application simultaneously.

Figure 18. Example of a Composite Application
Extended applications and composite applications are interactive (fully or partially synchronous). This multilevel integration pattern is most commonly implemented with some combination of an application server (i.e., the platform middleware for the integration application) and integration tools (e.g., a superservice and/or gateway). This implies that the developer will code the custom middle layer integration application using some third-generation language (3GL) or fourth-generation language (4GL). However, some form of graphically programmed integration broker without 3GL or even 4GL coding can solve some synchronous integration problems.

5.2 Screen Scraping and Integration Servers

Programmatic emulation of terminal interfaces, also known as screen scraping, is a valuable and widespread tactic, particularly for extended and composite applications. Admittedly, screen scraping is not elegant, and it creates maintenance problems because any change to a screen map will require adjustments in the screen-scraping tool and sometimes also in a client application. It is often better to use an alternative approach, such as a message, an ORB or an RPC interface, if one exists. Sometimes it is practical to modify the application to support such interfaces. Nevertheless, “scraping” a terminal data stream can be very useful because it is noninvasive, widely available and uses a highly structured protocol. Screen-oriented messages, including 3270 data streams and HTML or XML pages, are far easier to interpret and programmatically emulate than an APPC or sockets conversation.

Some products with screen-scraping capabilities are primarily intended to complement new midtier integration code that runs on other application servers. Examples of such products include:

- CST’s (www.cst.com) Jacada product family, for both mainframe and AS/400 applications
- ClientSoft’s (www.clientsoft.com) ClientBuilder suite
- Computer Associates International’s (www.ca.com) Opal, a popular, widely used part of its Harmony product family
- Computer Network Technology’s (www.cnt.com) Enterprise/Access, a development tool and runtime environment for re-engineering legacy applications to open networking environments
- EnterpriseLink Technology’s (www.enterpriselink.com) SmartTran, including SmartTran Loader, SmartTran Builder and SmartTran Server
- Mozart Systems’ (www.mozart.com) Mozart, including MozNet, MozNet3270, MozWin and MozAgent
- Computer Associates/Platinum’s InfoSession (www.platinum.com), a unique approach to mainframe access that combines mainframe-based 3270 data stream interpretation and session management services with a SQL-style access mechanism for enabling distributed access from Web and client/server GUI front-end applications
- Santa Cruz Operation’s (www.sco.com) Tarantella for Web-style application front ends
- Smart Technologies’ (recently acquired by i2 Technologies, www.i2.com) Smart DNA, a scalable Web server architecture that opens the data from existing systems to Web delivery
- Softouch Systems’ (www.softouch.com) CrossPlex for enabling Web access to 3270-based applications
Other products blend screen-scraping capabilities with other features, such as general-purpose application servers (platform middleware), gateways and/or other (non-screen-scraping) integration functions. The following are examples of such products:

- Gresham Computing’s Casablanca (www.gresh.com)
- InfoSpinner’s ForeSite (www.infospinner.com)
- Intelligent Environments’ Amazon Integrator (www.ieinc.com)
- Mitem’s MitemView (www.mitem.com)
- Simware’s Salvo Server (www.simware.com)

Note that any general-purpose application server can also be used to implement the middle layer in a composite application by using gateways to connect into heterogeneous back-end server applications.

### 5.3 Superservices

Integration solutions that are intended to encompass many application systems, and to last for many years, will usually benefit from the adoption of a systematic, strategic view of architecture. The long planning horizon for a strategic integration project will often lead to making upfront investments that make subsequent administration, additions and modifications easier to implement.

A superservice layer can be a helpful ingredient in a strategic integration solution. Directory and security services are two of the most common functions of a superservice. For example, a superservice can superimpose some coherence on a heterogeneous configuration by providing a superdirectory service that can span several environments (e.g., by resolving a business component name or logical destination name into the address of an actual data item, program, queue name or other resource). A superdirectory is one that can interact with other directories, and is often based on LDAP. Or, a superservice may attempt to provide cohesive security services that can span two or more software environments. Note that the underlying runtime software environments, i.e., platform middleware, DBMS, MOM or operating systems, may still be using their own directory and security services, so the superservice is indirect and high-level rather than actually replacing the native functions.

A superservice may also shadow or specialize functions other than directory or security. For example, it can do one or more of these things:

- Allow different middleware products to share a common API (see the left side of Figure 10 in Section 2.2.2). The superservice may "normalize" the service semantics among different products by hiding some features (the lowest-common-denominator strategy, bringing the potential drawback of hiding some useful feature found in only one product), or by filling in the gaps itself (a theoretically laudable strategy that brings the drawback of a maintenance burden to support some potentially complex basic middleware logic in the abstraction layer).

- Simplify and shield the developer from the complexity of the middleware’s native API (thus helping even if there is only one middleware product).

- Protect against change, so that the application could theoretically be ported to different middleware or even to a different operating system without changing the application source code (see Figure 10, right side).

- Help enforce programming standards (e.g., security or directory look-ups).
• Enrich the underlying middleware by adding complementary features.

A superservice typically functions as an abstraction layer, a design concept that is used many different ways in modern computing. In a sense, all middleware that exposes an API (directly to the application program) is a type of abstraction layer. For example, platform middleware is an abstraction layer above the operating system. Thus, CICS, DCE, Java Virtual Machine, Tuxedo and DBMS stored procedures are abstraction layers that can, with varying degrees of success, enable application portability among multiple operating systems. Another form of abstraction layer is a DBMS gateway. These abstract the DBMS APIs, generally using ODBC as a super-API to normalize SQL dialects (see Section 6.0).

Middleware abstraction layers are often custom code, but sometimes they are part of an off-the-shelf middleware product, such as Candle’s Roma. In either case, they are system software, not application software, because they offer general-purpose, reusable technical services rather than application-specific business functions.

5.4 Wrappers

Gateways and superservices help resolve technical differences among heterogeneous middleware products, but they do not reconcile application differences such as disparate data formats and semantics. To address these higher-level aspects of inconsistency, developers must either change the application program code (in the source or destination) or use a wrapper or broker. This section describes wrappers; brokers are covered in the following section.

A wrapper is a proxy, i.e., executable code, that makes a foreign application program appear compatible with a local program by translating a local call or message into a call or message understandable by the foreign application. A wrapper differs from a middleware gateway or abstraction layer because it is tailored to the semantics and formats of a particular application interface, i.e., it knows how to translate certain input and/or output parameters. In some cases, that is all the wrapper does, because the participating applications happen to use the same underlying middleware so protocol conversion is not required (see the left side of Figure 19).

However, in many cases, a wrapper attacks technical protocol differences in addition to the application-specific semantic and format differences (see the right side of Figure 19). When a wrapper speaks to middleware or a middleware gateway that is dissimilar to the local platform middleware environment, it is, in effect, acting as a value-added kind of gateway.
Microsoft’s COM-TI is a third variation on the notion of a wrapper. It is a general-purpose two-way protocol gateway from COM to CICS. COM-TI is supplemented by development facilities that generate type library entries so that it can behave as an application wrapper. A client program invokes a method on COM-TI which then conducts a dialogue with a remote CICS mainframe application in accordance with the information from the type library. The remote CICS transaction appears to be a local COM object.

Wrappers may be on the same computer as the client (requesting or sending) application, or they may be on the same computer as the wrapped application, or an intermediate system. Figure 19 shows only two of many possible wrapper configurations. Most wrappers support two-way, request/reply communication patterns, but they may support one-way messages. A “thin” wrapper understands one interface (entry point). A “fat” wrapper can deal with multiple interfaces, sometimes in multiple systems.

A wrapper is usually more application software than system software, so a wrapper is generally not exactly middleware (middleware is system software, referring to reusable runtime code that performs generic, non-application-specific functions). However, wrappers may contain middleware. Moreover, they sometimes act as a protocol gateway (whether in tailored application code or by embedding something that is recognizable as a reusable middleware component). Most wrappers are written in a 3GL, such as C, C++ or Java, although there are also high-level development tools that help generate wrappers (the wrapper development tools are not middleware either).

Screen-scraping tools may be used to wrap an application. They often include a user-friendly development tool that reverse-engineers an existing screen map (usually 3270 or 5250 terminal screens). They help an application developer write a new PC, Unix or Web front-end application that can interact with a legacy application without having to fully understand and code directly to, for example, the complexities of HLLAPI, a lower-level middleware API for CICS and IMS/TM.

The word “wrapper” is used most often when the calling application runs on an ORB or OTM, such as a COM, CORBA or Java-based platform middleware product. However, the term is also suitable for use in non-object-based environments. Sometimes the word “adapter” is used to describe a wrapper for nonobject environments, but the connotations are slightly different (see Section 5.6).

5.5 Integration Brokers

An integration broker acts as a multipurpose, reusable “fat” wrapper. Like a simple wrapper, a broker must be tailored to understand specific application interfaces and message or data formats. However, a broker differs from a common “thin” wrapper in two ways. First, brokers are fat wrappers, capable of dealing with multiple sources and consumers (M:N) instead of the point-to-point (1:1) coupling of most wrappers. Second, brokers have a core of reusable runtime code that performs generic functions, and thus are middleware (system software) rather than application software. They have built-in functions that reduce or eliminate the need for 3GL or 4GL hand-coding. Most brokers are tailored through a GUI development tool, although most have provision to exit to custom code where necessary. A broker could be used to wrap just one application interface (1:1), but it is capable of more.

A broker is based on program-to-program, database-to-database and/or program-to-database communication between disparate applications. A broker is not just communication middleware (e.g., MOM) because it provides value-added services (transformation, intelligent flow automation and sometimes other services) from the application layer of the OSI stack. Some brokers include their own communication middleware, while others require the use of a specific middleware product, and yet others have interfaces to speak with a wide range of lower-level middleware (this is a key variation among
broker products. Some brokers bundle in middleware gateways and/or superservices such as meta-directories. Virtually all are bundled with some development and administration tools.

All integration brokers support one-way (asynchronous) communication, whether individual messages, large files or database tables. This lets them address a broad range of integration scenarios such as 1) data reconciliation among two or more application systems, or 2) multistep business processes that involve a sequence of activities among multiple applications. Some also support synchronous, request/reply integration, including composite applications.

Brokers may be deployed in one or more of three basic roles:

- The broker may be a central hub, coordinating integration across multiple application systems which sit as nodes on spokes (see the left side of Figure 20). Application systems may be connected to the hub using a variety of protocols, sometimes including file transfer, TCP/IP sockets, MOM or even screen scraping. In the majority of circumstances, communication is one-way and asynchronous (in the eyes of the application systems). The hub reconciles both transport-level and application-level differences. Workflow systems, interface engines and data transformation products are usually configured as hubs. Broker products that specialize in the hub-and-spoke architecture often do not include their own communication middleware; rather, they have numerous gateways and other interface mechanisms.

- Alternatively, the broker may act as server node on a message bus or “pipe,” performing many of the same functions (e.g., transformation, content-based routing or process management) found in the hub of a hub-and-spoke architecture. However, a broker node does not act as a gateway among different transport-level communication systems or middleware (see the right side of Figure 20). All applications that connect into the message pipe communicate over the common communication middleware. Integration platforms such as Active Software’s ActiveWorks, Tibco’s ActiveEnterprise and Vitria’s BusinessWare bundle in both the messaging middleware and the broker. The applications interact through either synchronous or asynchronous communication patterns.

- Finally, broker middleware can also be positioned as a fat wrapper (or “connector”), acting as the interface between one application and the message bus (or multiple applications and the bus, effectively making the broker a hub). Most of the data transformation and routing features that are helpful in a broker node are also useful in a fat wrapper (see the far right of Figure 20).

![Hub and Spoke vs Message Bus](image-url)
In practice, broker topologies may be more complex than a simple hub or pipe. Sometimes multiple hubs are configured into a “snowflake” design in which each application is linked to one hub (see left side of Figure 21). Each hub is linked to one or more other hubs. Alternatively, a network of multiple hubs may be used where each application system may be able to interact directly with many different hubs (see the right side of Figure 21). Some broker products are capable of supporting several different configuration options, whereas others are optimized for one or two particular configurations.

Examples of integration middleware product suites with broker (and other) capabilities include:

- Active Software’s ActiveWorks Integration System (www.activesw.com)
- BEA’s e-link (www.beasys.com), a superservice and platform with optional brokers
- Skyva International’s Skyva (www.skyva.com)
- Candle’s Roma (www.candle.com), a superservice with optional brokers
- Cross Worlds’ Integration Solutions (www.crossworlds.com)
- Extricity Software’s Alliance (www.extricity.com)
- Frontec’s AMTrix (www.frontec.com/index2.html)
- Glotech Solutions’ Message Broker System (www.glotech.com)
- GEIS’ Enterprise System, EDI Application Integrator and InterLinx (www.geis.com)
- HIE’s Cloverleaf and OM3 (www.hie.com)
- Hewlett-Packard’s Changengine (www.hp.com)
- I2’s Rhythm Link (www.i2.com)
- IBM’s MQSeries Integrator and MQSeries Workflow (www.ibm.com)
- InConcert’s Inconcernt (www.inconcert.com)
Middleware: The Glue for Modern Applications

- Level 8’s Geneva (www.level8.com), a superservice platform with optional broker
- SunGard’s MINT (www.mintech.com)
- MSI’s Solutions (www.msi-solutions.com)
- Muscato’s ENGIN (www.muscato.com)
- New Era of Networks’ MQSeries Integrator, Neonet, (CAI’s) Impact and (Vie’s) Copernicus (www.neonsoft.com)
- Netik’s xNetik (www.netik.com)
- Oberon Software’s Prospero (www.oberonsoftware.com)
- ObjectSwitch’s ObjectSwitch (www.objectswitch.com)
- Oracle’s AQ (www.oracle.com)
- Precise Connectivity Solutions’ Q/Booster and Web/Booster (www.precise-conn.com)
- PRL’s I/O Exchange (www.priscotland.com)
- SAGA Software’s Sagavista (www.sagasoftware.com)
- Software AG’s EntireX Message Broker and EntireX DCOM (www.softwareag.com)
- Software Technologies’ DataGate (www.stc.com)
- Sopra’s Bus Applicatif and Regles du Jou (www.sopra.com)
- Tibco’s TIB/ActiveEnterprise (www.tibco.com)
- TSI Software’s Mercator and (Braid) Messenger (www.tsisoft.com)
- Vitria’s Businessware (www.vitria.com)

5.6 Adapters

Integration middleware vendors are increasingly offering off-the-shelf adapters, including development tools and industry-specific or application-specific integration templates of various forms. The term adapter or connector describes some set of technology for connecting an integration broker with an application system or an external middleware product. An adapter may include runtime middleware (e.g., gateway or abstraction-layer-type services), development tools and/or some form of templates (e.g., for transformation or flow automation). One adapter generally deals with a group of interfaces (entry points) into the same target application; one would not call each separate interface an “adapter.”

Adapters come in two basic forms:

- An application-specific adapter, i.e., one that deals with application formats and semantics, is a set of one or more wrappers or technology that helps generate such wrappers. For example, Visual Edge’s Madrid (www.visuledge.com) is an adapter toolset that connects into SAP’s R/3 using application-specific wrappers. It can span COM, CORBA and other middleware technologies, thus combining gateway functions with application-level functions.

- By contrast, a technical adapter can be a gateway, a superservice or both. A technical adapter does not understand the semantics or formats of any particular application, so the burden of dealing with
those issues falls on the developer. For example, many integration middleware products offer adapters into CICS, COM, CORBA and MQSeries.

Many vendors have adapters to popular packaged applications such as SAP R/3. Vendors that specialize in the finance industry, such as Neon/Vie, Netik, SunGard/Mint, Tibco and TSI (recently joined by HIE, Software Technologies and others), support finance-specific message sets such as SWIFT. Vendors that target healthcare support HL7, a healthcare-oriented message set. Some application-specific extensions are now found in almost all of the integration brokers, although virtually all of the them are also capable of general-purpose integration.

5.7 XML Integration Servers

A variety of new integration tools and integration servers have recently come to market to capitalize on the sudden emergence of XML. The XML specification defines a syntax and document organization for data, represented by human- and machine-readable tag/value pairs (e.g., <first_name>Ben</first_name>). XML itself, and other members of the XML “family” of standards, are produced by the World Wide Web Consortium, with domain-specific specifications now also being developed by many other consortia (e.g., the OMG’ XML Metadata Interchange Format).

XML-oriented integration middleware is structurally similar to other middleware that offers super-APIs, gateways, communication middleware, platform middleware and/or development tools. Early support for XML gives some products a temporary advantage in certain applications compared to traditional middleware. However, by year-end 1999, 75 percent of middleware vendors will have deployed or at least announced XML support, including compliance for at least XML and DOM (0.9 probability).

XML-based Web integration servers are a variation on previous Web integration servers. Both kinds of Web integration servers directly support the HTTP protocol, execute a proprietary, high-level 4GL or scripting language, and may have one or more database or application adapters. XML-based integration servers happen to use XML as their format for normalized data, and are primarily being used for Internet-based, business-to-business (BTB) application integration. Their combination of technologies is attractive for opportunistic BTB projects, particularly when doing business with diverse and rapidly changing business partners, because they simplify negotiating data formats and the basic communications infrastructure (i.e., the ISP, firewall and HTTP protocol are typically already available). Because this is an emerging methodology that is still unproven on a large scale, and because of factors such as the unpredictable throughput associated with Internet-based communications, we recommend that Internet-based application integration using technologies such as XML-based Web integration servers initially be restricted to opportunistic projects tolerant to variations in connection throughput.

Examples of the many XML-based middleware products include:

- Arkona’s Universal Update Server (www.arkona.com)
- Object Design’s eXcelon (www.objectdesign.com)
- Vertex Industries’ evolve (www.vetx.com)
- WebMethods’ B2B Integration Server (www.webmethods.com)

6.0 Data-Oriented Integration Technology

Data-centric approaches to integration reflect the obvious importance of data in data processing. The majority of today’s real-world integration solutions are data-centric in one way or another, although the
more-traditional forms of batch data integration are giving way to new forms of asynchronous, event-driven, data-centric integration, and to message-based and object-based integration for some situations.

Two different data-centric techniques are relevant to integration: direct data integration and file transfer. Direct data integration implies real-time read or write access to databases or files that are owned by other application systems, using the basic remote data access middleware described in Section 2.1.1, or via the database gateways and related tools described in this section. By contrast, batch transfer moves groups of records at a time, and is not exactly a middleware issue because it takes place offline rather than at runtime.

This section describes a range of data-oriented software products that are relevant for integration, including gateways, replication services, propagation services and transformation engines. It does not address batch file transfer utilities or data-cleansing tools, which are further from the concept of middleware, although we recognize their importance for many integration scenarios. It also does not cover batch-program-generating data extraction and transformation facilities such as Ardent Software’s Warehouse Executive (www.ardentsoftware.com); Computer Associates/Platinum Technology’s DecisionBase (www.platinum.com); Carleton’s Pure*Extract (www.carleton.com); Evolutionary Technologies’ Extract (www.eti.com); Information Builders’ Enterprise Copy Manager (www.ibi.com); and SAS’ Warehouse Administrator (www.sas.com).

6.1 Database Gateways

Data-oriented integration middleware provides access to heterogeneous data sources through direct protocol translation (in a database gateway), API translation (in an abstraction layer) or both. Examples of database gateway products include:

- Cross Access’ Cross Access Series4 (www.crossaccess.com)
- IBM’s DB2 DataJoiner (www.ibm.com)
- Information Builders’ Enterprise Data Access (EDA) Relational Gateways (www.ibi.com)
- Neon Systems’ Shadow Direct (www.neonsys.com)
- Oracle’s Transparent Gateway to DB2 (www.oracle.com)
- Sybase’s OmniConnect (www.sybase.com)

The topology of the simpler database gateway is like that of simple, remote data access middleware (see Figure 3 in Section 2.1.1). However, some of the newer forms of database gateways use an intermediate layer, such as a hub or intermediate server, to enable more powerful ways to tap into multiple data resources simultaneously. For example, Sybase’s OmniConnect, Information Builders’ EDA Hub Services and IBM’s DB2 DataJoiner use intermediate tiers to provide access to and joins between heterogeneous data stores which may be relational and/or nonrelational in their native form.

Gateways provide a high level of access transparency to heterogeneous data. However, direct access to heterogeneous databases and files is not a general solution for many integration problems because it has some limitations with respect to performance, functional richness and quality. Some of these limitations come from the need to translate between different SQL dialects, or, in the case of nonrelational data access, from SQL to other data manipulation languages.

There are also some inherent design implications that arise from reading or writing directly into another application system’s database. In the majority of circumstances, direct, real-time access to foreign data
through a database gateway is only practical for read purposes because it bypasses the application program logic in the participating application systems. Performance of updates and of complex (read-only) joins across sources, particularly those that involve nonrelational and relational data, has been unacceptable in many situations. Also, the functional richness offered by a particular DBMS is often traded off for the greater generality of a gateway’s programming interface. SQL gateways can also be expensive.

These concerns are particularly acute when operational, record-keeping data is involved. Offering direct access to OLTP data to businesspeople can create data quality issues, specifically those that are caused by misunderstanding the data semantics. Furthermore, if a gateway is set up to allow updates, data integrity problems can ensue if the user does not implement validation procedures that are equivalent to those in the operational application programs that are the primary update path for that data.

6.2 Replication and Propagation

Data replication and propagation services are different than database gateways and other middleware because they are invoked by a DBMS rather than directly by an application program. In most cases, replication and propagation services do not actually function as “middleware” because they do not send data to or from an application program. Furthermore, they typically run in copy mode, moving sets of related data as a group, rather than participating in a business transaction at runtime. However, in some circumstances replication technology is used to transfer individual update records asynchronously in near-real-time (i.e., an application program updates its local database, triggering a replication facility to send an individual update to a remote database).

Replication and propagation services can transfer the entire contents of a table or tables, or they can use database logs or triggers to identify changes since the last replication interval. They send insert, update or delete transactions from the source (“master” or “primary”) site to the target (receiving) database.

Replication services are used almost exclusively for intra-application (not integration) purposes. “Heterogeneous replication” is an oxymoron because replication services do not accommodate diverse data models and DBMS technology well. Roughly 70 percent of all data replication involves transferring data from an application database to a decision support database (e.g., data mart) or to a hot backup copy of the application database (i.e., the data is not physically distributed to multiple branch or regional locations). About another 18 percent of its usage is for primary site replication, wherein updates are only allowed at the primary central site and a partial copy is distributed to remote sites (e.g., branches or regional offices). In this role, replication helps ensure data integrity and reduces the support burden of synchronizing a replicate database. Another 10 percent of its usage is for uploading data from multiple regional or branch offices to the central site, where each remote site represents the primary site for its subset. Only about 2 percent of the usage of replication services is for maintaining distributed databases where updates are allowed at both locations — i.e., true multisite replication.

Replication services have no inherent reconciliation capability, although they offer a set of services that allow a developer to define business rules to be used to resolve potential inconsistencies. Replication services are not a general solution for distributed databases (i.e., involving updates of data copies in multiple locations) because of the complexity of the configuration and the risk of data consistency problems. Note that when replication services are labeled “bidirectional,” this usually just indicates that the replication may be initiated from the source (“push”) or the target (“pull”) — it does not mean that these services have eliminated the problems involved in allowing updates on both sides. Like copy management tools, replication tools can be initiated on a time or event basis by the source database. DBMS vendors provide most replication services, with basic functionality often now bundled into the price of the
database. However, some vendors, such as DataMirror (www.datamirror.com) and Sybase (www.sybase.com), also provide third-party, database-independent solutions.

Data propagation tools typically rely on database logs to move either all changes or net changes. Unlike replication services, however, they apply to heterogeneous environments, including nonrelational-to-relational transfers. Data can move in both directions, but only one way at a time. Some of these tools, such as IBM’s DataPropagator, enable the developer to define business rules between fields (or columns) to reconcile the source and targets (a “push” approach). Others, such as Platinum’s InfoPump, use more of a gateway-like approach, which handles the mapping and enables the user to define a query to run against the source (a “pull” approach). Both approaches create some overhead on the source system. This overhead is primarily affected by how data is accessed on the source side (via SQL directly against the production data, via the database logs or via an alternative set of logs), the volatility of the source database, and the resulting frequency of propagation. Total throughput is further affected by the degree of transformation and mapping performed, and the file transfer speed. However, the same data consistency problems associated with replication solutions can arise when updates are allowed on both sides. Thus, the manageability of this environment is critical to its success. The better propagation tools provide a GUI administrative console.

6.3 Transformation Engines

Transformation engines are a kind of data-centric integration broker, long on transformation capabilities but generally fairly limited in terms of their ability to redirect data (e.g., they have limited content-based routing to multiple destinations). They are typically used to load a data mart and, at times, a data warehouse. The transformation engine — or a separate program or utility — is used to extract data from the input sources (e.g., input files, program APIs, nonrelational databases and relational databases). The transformation engine performs the various stages of data preparation asynchronously and in parallel, a more efficient architecture than the serial process typically used when custom application programs are utilized for the same job (see Figure 22).

![Figure 22. Transformation Engines](image)

**Strengths:**
- Flexible configuration
- Asynchronous processing
- Extensible architecture
- Complex integration and transformation
- Improved support for operational data store

Source: GartnerGroup
The engine performs its various functions (e.g., data cleansing) at scheduled time intervals or when triggered by an event. The engine can optionally be deployed on a processor that is separate from the source production system and the target data warehouse or data mart, and can also leverage the scalability of SMP and MPP hardware technology to maximize performance and throughput efficiency.

Transformation-engine-based tools on the market today exhibit a variety of architectural designs. None yet completely conforms to our definition of the engine architecture (as depicted in Figure 22). Some are based on object models (e.g., Sagent Technology’s engine); while others are relational-model-based (e.g., those from Constellar and Ardent/Vmark Software). Some tools use a defined database as a staging area for holding and massaging data (e.g., Constellar), others do not, and still others leave this decision up to the developer.

An enterprise can use a transformation engine to “drip feed” new “changed-data” to a target database (e.g., a data warehouse) on an ongoing basis instead of using a massive overnight batch data load. This is particularly helpful for “operational DSS” requirements, and for combining an operational data store within a data warehouse architecture. By 2001, data extraction and transformation tools will support multisource changed-data capture in a way that is noninvasive to the operational application, e.g., based on replication technology (0.8 probability). For organizations with an extensive set of extraction applications already in place, a transformation engine can augment these applications and replace the complex logic required for integration, thereby reducing support and maintenance costs. Some transformation engines, such as Constellar’s Hub, are enhancing their record-at-a-time capabilities by adding better support for MOM (e.g., IBM’s MQSeries) and are evolving toward general-purpose integration brokers directed at both operational and decision support applications.

Examples of transformation engines include:

- Acta Technology’s ActaWorks (www.acta.com)
- Ardent Software’s DataStage (www.ardentsoftware.com)
- Carleton’s Pure*Integrate (www.carleton.com)
- Constellar’s Constellar Hub (www.constellar.com)
- Informatica’s PowerMart and PowerCenter (www.informatica.com)
- Microsoft’s Data Transformation Server (www.microsoft.com)
- Sagent Technology’s Sagent Data Mart Solution (www.sagenttech.com)
- ShowCase’s Strategy (www.showcasecorp.com)
- SmartDB’s WorkBench (www.smartdb.com)

7.0 Conclusions

Enterprises face a daunting array of modern business challenges: increasing competition, mergers, acquisitions, deregulation, new regulations, worldwide markets, margin pressure, political turbulence and industry restructuring. IS organizations can be a major force in helping to answer these challenges, or they can become part of the problem if they are unable to respond to the new business imperatives. A wealth of new hardware and software technology continues to come to market, but it is not obvious how best to leverage it. Middleware will play a key role in the success — or failure — of enterprise systems during the next five years.
Middleware is not inherently complicated; it only seems that way because the terminology is often muddled. Each of the kinds of middleware reviewed in this report has its areas of strength and its limitations. A large enterprise is likely to use at least one product from every category of middleware somewhere in its application portfolio. Middleware is not an end in itself, it is the means to an end — namely, effective distributed applications.

The migration from two-tier computing to Internet and other multitier applications will continue to drive rapid growth in the use of program-to-program middleware such as MOM, TPMs, ORBs, OTMs and other application servers. The growth of MOM will happen simultaneously with the spread of component-oriented application servers because the technologies are complementary rather than being alternatives for most purposes. MOM works with application servers by providing certain functions, such as asynchronous (store-and-forward) messaging. MOM is also particularly helpful for integrating independently developed application systems that have heterogeneous underlying technologies — e.g., disparate operating systems, application servers, programming languages and DBMSs. We recommend that enterprises have two separate middleware strategies: one for interapplication purposes and one for intra-application purposes.

Components and object-oriented design and programming languages are having a major impact on the way enterprise applications are developed. Other technology changes, particularly Internet-inspired technologies such as XML, will soon bring further significant improvements, along with some burdens, to mainstream enterprises. The only certainty in middleware is that hardware and software will never be homogeneous across the network at any point in time.

The subject of middleware will likely continue to challenge developers worldwide during the next five years because application architectures are changing at an unprecedented rate. The middleware technology, products and vendors that support today’s applications are strikingly different than those used five years ago and those that will be used five years in the future.
### Appendix A: Acronym Key

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3GL</td>
<td>Third-generation language</td>
</tr>
<tr>
<td>4GL</td>
<td>Fourth-generation language</td>
</tr>
<tr>
<td>API</td>
<td>Application programming interface</td>
</tr>
<tr>
<td>APPC</td>
<td>Advanced Program to Program Communication</td>
</tr>
<tr>
<td>CICS</td>
<td>Customer Information Control System</td>
</tr>
<tr>
<td>COM</td>
<td>Component Object Model</td>
</tr>
<tr>
<td>CORBA</td>
<td>Common Object Request Broker Architecture</td>
</tr>
<tr>
<td>CPI-C</td>
<td>Common Programming Interface for Communications</td>
</tr>
<tr>
<td>DBMS</td>
<td>Database management system</td>
</tr>
<tr>
<td>DCE</td>
<td>Distributed Computing Environment</td>
</tr>
<tr>
<td>DCOM</td>
<td>Distributed Component Object Model</td>
</tr>
<tr>
<td>DOM</td>
<td>Document Object Model</td>
</tr>
<tr>
<td>DSS</td>
<td>Decision support systems</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical user interface</td>
</tr>
<tr>
<td>HLLAPI</td>
<td>High Level Language API</td>
</tr>
<tr>
<td>IIOP</td>
<td>Internet Inter-ORB Protocol</td>
</tr>
<tr>
<td>JDBC</td>
<td>Java Database Connectivity</td>
</tr>
<tr>
<td>MOM</td>
<td>Message-oriented middleware</td>
</tr>
<tr>
<td>MTS</td>
<td>Microsoft Transaction Server</td>
</tr>
<tr>
<td>MW</td>
<td>Middleware</td>
</tr>
<tr>
<td>NFS</td>
<td>Network File System</td>
</tr>
<tr>
<td>ODBC</td>
<td>Open Database Connectivity</td>
</tr>
<tr>
<td>OLTP</td>
<td>Online transaction processing</td>
</tr>
<tr>
<td>OMG</td>
<td>Object Management Group</td>
</tr>
<tr>
<td>ONC</td>
<td>Open Network Computing</td>
</tr>
<tr>
<td>ORB</td>
<td>Object request broker</td>
</tr>
<tr>
<td>OSI</td>
<td>Open Systems Interconnection</td>
</tr>
<tr>
<td>OTM</td>
<td>Object transaction monitor</td>
</tr>
<tr>
<td>RPC</td>
<td>Remote procedure call</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>SNA</td>
<td>Systems Network Architecture</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>TI</td>
<td>Transport Independent</td>
</tr>
<tr>
<td>TPM</td>
<td>Transaction-processing monitor</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>