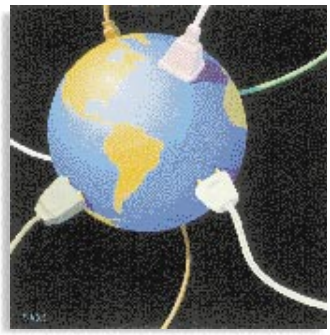
The Muddle of Middleware



Each month, the TSC examines a key emerging technology. This time, we evaluate the capabilities of middleware.

An important buzzword in the information technology vocabulary to emerge over the past few years is *middleware*. It has become a catchall for a large, sometimes confusing assortment of client/server technologies and has served as a rubric for all the products claiming to facilitate distributed applications.

The term is subject to almost as many definitions as there are IT practitioners and pundits. One popular definition comes from Orfali, Harkey and Edwards in *Essential Client/Server Survival Guide* (Van Nostrand Reinhold, 1994), who define it as the "/" in "client/server"; that is, the in-between stuff that connects the two. Another—more skeptical—definition comes from Mark Hanner of the Meta Group, quoted in *InfoWorld* (Dec. 25, 1995), who labels it "the thing that comes between you and your data." By any definition, middleware is a collection of important technologies that are much in the spirit of open systems.

Middleware Categories

For convenience, contemporary middleware can be divided into three categories: *data-oriented middleware*, intended to permit the client application to be written without regard for the physical location of the data; *message-oriented middleware* (MOM), for passing messages reliably between applications; and *request-oriented middleware*, for enabling distributed applications to execute transactions and make requests of one another.

The key component of data middleware is the database management system (DBMS). Most of the popular relational DBMSs either function directly as a conduit for distributed data access or, through additional data gateway products, are extended to enable remote data access. Stored structured query language (SQL) procedures are also a form of middleware, through which a client can request the retrieval and transmission of routinely accessed data for local use.

There are many different approaches to data-oriented middleware. Microsoft's ODBC and the related X/Open CLI interface allow considerable database independence and transparency in accessing one or more server DBMSs from a client program. Tools for data replication propagate changes in a master database to clients and data hubs to synchronize local copies of the data with the master copy. Complete data files can be copied automatically via managed file transfer products. Soon, DBMSs will provide distributed update capability using middleware, although today this functionality is still problematic.

The main advantage of data-oriented middleware is that applications require only minimal changes (if any) to access remote data. For the most part, the infrastructure required to move and access data is external to the program. While this simplifies the life of the programmer, it leads to one of the potential disadvantages of this class of products—the operational complexity of synchronizing data movement and the related difficulty of guaranteeing data consistency when something goes wrong. This approach moves the complexity from programmers to database administrators and operations staff.

Message-oriented middleware includes protocol-independent messaging and message queuing. Many MOM products guarantee reliable delivery of messages and queued data. MOM permits applications to be designed around the concept of filling queued requests and makes possible the implementation of informal workflow structures.

Request-oriented middleware includes products for transaction processing (TP), remote procedure calling (RPC) and object request brokering (ORB). The use of a TP monitor enables a distributed application to provide transactional integrity through two-phase commitment and preservation of the so-called ACID properties:

- Atomicity: the transaction succeeds or fails as a unit.
- Consistency: the results of executing a transaction are always consistent.
- Isolation: transactions are independent of one another.
- Durability: a committed transaction is permanent and must survive a system failure.

The RPC paradigm allows applications to request remote services as though they were local procedures. Object request brokers permit applications to make requests of one another transparently. One RPC product, the Distributed Computing Environment (DCE), also supplies the security, directory and time services required in a distributed applications environment.

The use of a request-oriented middleware product permits applications to be written according to the request/response paradigm. This application architecture fits well with graphical user interfaces and facilitates the migration to object technology. In fact, many organizations are creating informal object structures for common business objects and system services using existing TP monitors or RPC products so they may be in a position to exploit object technology as it evolves in CORBA-compliant and COM/OLE environments. They are using ORB middleware to position themselves to take

By Derek Kaufman

advantage of component technology when it becomes widespread.

Guiding Principles

The principles behind middleware are to permit applications to move freely within the computing infrastructure and the enterprise, and to insulate applications from underlying technologies, particularly the complexities of communications, networking and differing operating system application programming interfaces (API)s. Both principles are highly consistent with the aims of open systems.

The location independence provided by middleware permits applications to be deployed on the bases of performance, reliability, economics and business concerns. Applications become free to migrate from server to client; from server to middle tier and back; from desktop to server; from server to server; and, in the extreme case of mobile applications, to wander geographically.

In the isolation role, middleware is a strong technology for fostering platform independence. Middleware permits the business logic of an application to be written independent of the operating system context in which it executes. Middleware shields the application from knowing how it has been invoked and how it must return results.

A future vision for middleware builds on this open systems role to make possible an object-oriented architecture in which cooperating business objects make requests of other objects and provide services for one another. In this vision, today's middleware will ally with object technology to provide complete location transparency and platform isolation. In the future, middleware will provide the software sockets and backplane into which component applications can be fitted to build new application systems for the enterprise, thus ushering in the era of practical reuse and component technology.

How far off is that vision? In some environments, it can be realized in limited fashion today; some people claim to be selling it. In fact, it is close enough that many application development teams are building their systems now—using available middleware products—with the expectation that these applications will evolve into the object-oriented paradigm. The benefit of middleware to these teams comes not so much in the specific product features currently available, but rather in the ability to structure their applications to be service-oriented and event-driven.

Middleware is already an important open systems technology. If it is applied according to open systems principles and with an eye to the future, the benefits of middleware may include:

- Increased longevity for applications.
- Reduced response time to changes in the business and technical environment.
- The ability to make continuous improvement in the computing platform and development tools without impacting applications.
- Reduced development cost through reuse at multiple levels.
- Increased programmer productivity.
- Improved quality of application software: reliability, availability, throughput, accuracy, maintainability and manageability.
- Increased application adaptability.
- Increased application scalability.

Like all other open technologies, middleware must be applied with caution. Users must be certain that any middleware product used is standards-based and truly open to avoid lock-in and dependency on proprietary interfaces. IT

Derek Kaufman *is middleware manager for Levi Strauss & Co. in San Francisco and a member of the UniForum Technical Steering Committee. He can be reached at uslev5y4@ibmmail.com.*