Articles & White Papers

MIDDLEWARE DECOUPLES APPLICATION FROM LOW-LEVEL PLUMBING

Selecting the right "connectivity tissue" requires an understanding of technology choices, vendor products, realistic goals and vision

By Max Dolgicer

The major challenges that many I/S managers face today can be summarized in a few words: adaptability, responsiveness and leverage. To be successful, organizations need to deliver on these fronts. These challenges alone are big enough. To help deliver on them, I/S managers would like to have the advantage of business applications that are independent from heterogeneous communication protocols, operating systems, hardware platforms, databases, development tools and other lower-layer "plumbing." One benefit this independence would provide is the freedom for managers to decouple their systems and applications software buying decisions from this "plumbing." Whether corporate I/S is developing in-house or buying packaged software, "best-of-breed" software selection should be based on functionality, without any concern for the ability of that software to operate in a specific environment. For example, two programs communicating with each other across an SNA network using APPC LU6.2 should be able to continue to run in a TCP/IP environment without changes, provided that the "right" middleware architecture has been selected from the onset.

A NEW LAYER

To achieve this freedom, an isolation layer of software that shields systems software and applications from the "plumbing" is necessary. By coding to such a layer, corporate developers can also protect their investment. As new operating systems become widely available, no changes to the existing applications are required. The isolation layer itself must support these new entities. An existing Unix-based trading application should be able to run unchanged on Windows NT if it was originally written to such a middle layer, and this layer has extended its operating system's support to NT. This isolation layer is mandatory to achieve software and hardware independence, which in turn translates to greater adaptability, responsiveness and perhaps -- leveraging of existing systems. (See Fig. 1.)

Adaptability and responsiveness mean that business applications should be able to adapt quickly and respond adequately to ever-changing business requirements. For new client/server applications, the advantages and benefits of middleware are easy to quantify. While there is still a lot of confusion in what middleware can do, a "middleware" layer is a "must have" architectural component. When new client/server applications need to be linked to existing legacy applications, the situation is far more complex.

A NEW VISION

In the last two to three years most middleware vendors have bombarded us with words like encapsulation and wrappering. Listening to them, someone might think that encapsulating and wrapping existing systems is an easy task. In reality, this is far from the case -- the fancy words are just misleading. Selecting the right connectivity "tissue" between the new and the old systems requires an in-depth understanding of technology choices, vendor products, realistic goals and of course, vision.

So, what are the issues and what are the choices when linking new and old systems? This article attempts to address these issues and shed light on when to select which kind of middleware, given the challenge. To make it plain and simple, the right or wrong choice of middleware can make a tremendous difference across the board for managers, technicians and end users. While most I/S managers agree that middleware will be required to link old and new systems, they do not agree on how to do it.

A NEW APPROACH

Two approaches can be described for linking old and new systems: data passing or process transfer (message passing). This debate is also known as data-centric architecture versus process-centric architecture. Endorsement of one approach over the other will carry a tremendous weight in selecting the right middleware product.

The data passing approach implies the use of what is being touted as database middleware. Database middleware generally implies the use of database gateways, which are supplied by the middleware vendor. The client application issues an SQL request to the gateway, which in turn ships the request to the target database(s). Depending on the nature of the request, the gateway will translate it to the native dialect of the target database. For the most part, database middleware implies synchronous, connection-oriented communication (described later in this article).

The process transfer approach implies some sort of direct or indirect communication between application processes, which communicate with each other via messages. Both sides agree up front on the application protocol (or "contract") and the messages passed between the client and the server need to be predefined. Of course, this does not prevent any application program from accessing a database when necessary.

A NEW COMMUNICATION

When considering which communication model to choose, remember that the communication models offered by middleware have varying performance, flexibility and robustness characteristics. Efficiency is a basic requirement, since efficient communication for middleware is like having an efficient task scheduler for an operating system.

Two different communication technologies are at the heart of the process transfer approach: remote procedure calls (RPC) and messaging. Both the RPC and messaging models shield the requesting application from dependencies on the underlying data source -- data should be able to switch easily between different database management systems or even to an object-oriented database.

As the name suggests, RPC products are based on the concept of remote procedure calls. When using RPC-based products, communication code does not need to be hand-coded -- all communication code is being generated from the Interface Definition Language (IDL). For example, Boulder, Colo.-based Netwise Inc.'s TransAccess is an RPC-based product that provides high performance communication between clients and servers. It has been successfully used as an integration vehicle between a number of existing CICS and IMS applications and new Visual Basic- or PowerBuilder-based applications. TransAccess inherits all the advantages and disadvantages of an RPC-based solution. The disadvantages of the RPC paradigm are that it is synchronous and connection-oriented. Synchronous means that the client program is blocked, while the server program is preparing the answer; connection-oriented means that to communicate, an active connection must be maintained between application parts. Generally, the application components communicate with each other synchronously in a request-wait-reply fashion.

The messaging paradigm, on the other hand, can be connectionless -- no active network connection needs to be established between the application parts and where the information is being passed by enqueuing and dequeuing messages to and from the queues.

Two different mechanisms are used to get back the server's response: event notification or continuous polling. The latter mechanism requires polling, which means that one side must poll the other until the reply message can be received. This mechanism is not event driven, but it is easy to implement. Event notification does not require polling and takes advantage of the callback function. Callbacks are registered first and invoked later automatically via interrupts. For example, a named function can be registered by the sender and automatically invoked when send has been completed. Such a named function will receive control automatically when an incoming reply message arrives. Event notification, as opposed to polling, is somewhat harder to implement, but the payback is worthwhile. Callback allows a reaction to events as they happen. San Francisco-based PeerLogic Inc.'s Pipes Platform and Rosemont, Ill.-based Covia Technologies Inc.'s Communication Integrator are good examples of direct process-to-process messaging. The latter is implemented using the polling method, while the former uses the event notification method.

The bottom line is that the approaches of data passing versus process transfer, and synchronous versus asynchronous models are all closely interweaved with each other. When considering the "right" middleware to link new and old applications, these two factors are among the important, if not the most important, criteria.

DATA PASSING APPROACH

The data passing approach makes more sense when developing simple, two-tier applications. It also makes better sense for multitier decision support applications, especially the ones that need to access data from a variety of data sources concurrently. Other important data passing considerations include a desire to avoid changing existing applications. In most cases, new applications can be developed by accessing existing data sources, no modifications to existing applications need to be implemented. Often, developers want to use first generation application development tools (such as Concord, Mass.-based Powersoft Corp.'s PowerBuilder and Menlo Park, Calif.-based Gupta Corp.'s SQLWindows). Most data passing products are well integrated with such tools, although representatives of process transfer middleware are catching up fast. Data passing does not provide any possibility to control the amount of data being returned over the wire as a response to a client's request, which might be satisfactory if neither transaction throughput nor communication efficiency is a prime requirement. Of course, this list of considerations is far from exhaustive.

Products such as New York-based IBI's EDA/SQL; Emeryville, Calif.-based Sybase Inc.'s Enterprise Connect; IBM's DataJoiner; Burlington, Mass.-based TechGnosis Inc.'s SequeLink; Redwood Shores, Calif.-based Oracle Corp.'s Open Gateway and others are all database middleware products and represent, for the most part, the data passing approach. Some products support Remote Stored Procedures -- the ability of client programs to communicate with existing programs across the network. Examples of such programs are CICS transactions, Unix processes or any other existing programs/processes. Although this feature provides database gateways with RPC-like behavior (process-to-process communication), the underlying communication mechanisms used by database gateways are, in my opinion, not optimized for high performance.

DATABASE GATEWAYS

While on the surface database gateways provide similar functionality, there are substantial differences in their architectures. The most common architecture is a multidatabase gateway. A multidatabase gateway implies a three-tier architecture where the client, the gateway and the target database(s) all run on different platforms. (See Fig.

2.) Among such gateways are open gateways and RDBMS gateways. Open gateways such as Sybase/Enterprise Connect do not require the presence of a Sybase database on the gateway machine. In contrast, Oracle's Open Gateway requires an Oracle database on the gateway, which in turn connects to all other supported non-Oracle databases.

In addition to three-tier gateways, there are two-tier, point-to-point database gateways. These utilize connectivity components, which are provided when a database product client connects to the gateway, that reside in front of the target database. Point-to-point database gateways such as TechGnosis' SequeLink potentially can provide better performance than three-tier database gateways. Microsoft's ODBC and products that have implemented standard ODBC APIs can also be considered point-to-point database middleware, with one caveat -- the translation from ODBC calls to the native dialect of the target databases is performed on the client side.

The bottom line is that database access middleware is an attractive solution for two-tier, and possibly three-tier, decision support applications where a split second response time is not a requirement and synchronous connection-oriented communication is sufficient. Please note that the word "tier" applies to the application tier and even applications that use database gateway products that are three-tier, but will still be considered two-tier as long as the client application makes a request to the target database. In sum, new client/server applications can be easily linked to existing legacy data utilizing many of today's database gateway products.

PROCESS TRANSFER APPROACH

The process transfer is best suited to full three- or even multitier complex applications that require a subsecond response time as well as scalability. This is where efficient and diverse communication models utilized by non-database middleware products come in handy.

Whether you choose an RPC- or messaging-based solution, both models provide sophisticated communication subsystems that are by far more efficient and diverse than data-centric communication models utilized by database middleware. As a result, the process transfer approach is more suitable for high volume, mission-critical applications. Furthermore, this approach also shields application components from any programming language dependencies -- it allows all application parts to be developed in any programming language, rather than being confined to variances of SQL or even Sybase's Transact-SQL- based communication, as the case generally is when the data passing approach is implemented.

As businesses become more global and interenterprise communication becomes a more common way to connect different applications across diverse subsidiaries and enterprises. It is simply not feasible to link all these applications via a "data-centric" approach. In such a situation the transfer process is the most feasible approach.

MIDDLEWARE CHOICES

To date, more than 90% of client/server applications have been developed using a two-tier data-centric approach, according to International Systems Group Inc. As client/server applications become more complex and adhere to multitier architecture, the need to link new to existing processes (for example, CICS and IMS transactions) becomes more apparent. It is especially true when the reality that mainframes are not going to disappear in the foreseeable future sinks in. While the transfer process approach is more efficient and feasible than the alternative (for example, the data passing approach), corporate developers should be aware that despite buzzwords such as encapsulation and wrappering, there is no magic, and for the most part existing CICS or IMS transactions have to be modified.

The question is then, which process transfer middleware technology approach should be used? Today there are multiple middleware technologies existing under the banner of the process transfer approach -- RPC-based middleware, object request brokers, messaging and TP monitors. Which technology to choose depends on your specific requirements. Furthermore, although the questions might remain the same, the answers might change as vendor products become more mature and provide more functionality. Before selecting anything, one of the more important questions to ask is whether you are trying to build a corporate-wide middleware infrastructure or whether you intend to develop a few applications with a limited life span.

RPC-BASED MIDDLEWARE

Technology such as DCE might be appealing to corporations trying to build a uniform middleware infrastructure. Availability of services such as distributed security, naming, distributed time and platforms support is very atrractive. It is hard to imagine that technology of such a magnitude will be brought into an enterprise for the development of a few applications, regardless of the complexity of these applications. To adopt DCE is a complicated process that requires an enterprise-wide commitment as well as a substantial labor allocation. Corporations should also be aware that DCE resource requirements are quite high, language support is limited (at this time DCE only supports C as a programming language) and the RPC model is too limited to address diverse application requirements. At the same time, corporate developers need to be aware that DCE is available across many platforms including MVS, and today new and old applications can be linked together in a cohesive fashion using IBM’s Application Support for CICS and IMS.

ORB-BASED MIDDLEWARE

What about object request brokers (ORBs)? There are many reasons (subjective and objective) why the interest in ORBs is on the rise. First of all, by now many companies have been convinced that "objects are the only way" in developing distributed applications, and that the lightweight nature of some ORBs (as well as support for C++ interfaces) are good reasons for the current attraction.

Like DCE, a higher level of the interface represented by the IDL is appealing; all the necessary code for data marshaling and cross-platform communication is generated automatically. Unlike DCE, some ORBs support more diverse communication models, including message queuing, publish/
subscribe and asynchronous event notification mechanisms. Examples of such ORBs are Dublin, Ireland-based Iona Technologies Ltd.’s Orbix; San Diego-based ExperSoft Corp.’s PowerBroker and a few others. Also unlike DCE, ORBs do not provide a set of consistent services such as security, naming, transaction support, load balancing, message queuing and so forth. Support for these services varies among products. For example, Digital’s ObjectBroker supports security, while Iona’s Orbix extends standard, synchronous and connection-oriented Corba communication by supporting the publish/subscribe model.

While the level of interest in ORBs is high, as of this writing, most ORBs fall short when linking new and existing applications. The reason is that many ORBs are not available on platforms where the majority of existing application reside -- MVS, OS/400, VMS and so on. As a result, at least for now, using products such as Iona’s Orbix, ExperSoft’s PowerBroker and Mountain View, Calif.-based PostModern Computing Technologies Inc.’s ORBeline, means corporate developers will have to master their own integration. Integration of ORBs with messaging middleware, or writing directly to the communication protocol APIs, can be used for this purpose.

Some exceptions to the rule do exist, however. IBM’s DSOM plans to support MVS, including CICS and IMS, within the next few months. In addition, IBM will also provide objects the ability to communicate with each other asynchronously by integrating DSOM with MQSeries on many platforms including the mainframe. Digital's ObjectBroker already supports MVS, including full IDL implementation, but without CICS and IMS support.

In the absence of such product integration, companies will need to provide their own solutions. New York-based Level 8 Systems Inc., an integration company, successfully implemented the combination of SOM/DSOM and MQSeries for a large bank. The company chose to connect clients and AIX-based middle servers using simple TCP sockets. SOM on AIX has been used to encapsulate MQSeries-based message queuing objects, while communication over WANs to MQSeries on MVS provided the connectivity to the existing CICS applications. It should be noted that client to middle server communication could have been handled by DSOM, which is now available on the Windows platform as well. The existing CICS applications were modified, for example, CICS transactions had to become aware of the existence of message queues. To minimize the total number of CICS transactions to be modified, all CICS transactions were "front-ended" with a single transaction that was responsible for managing the queues. (See Fig. 3.)

MESSAGE-ORIENTED MIDDLEWARE

Message-oriented middleware (MOM) seems to be one of today's most mature middleware technologies, and there are plenty of commercial products on the market. As with ORBs, MOM support for services such as security, transaction management, naming/directories, time synchronization across the network and others is fragmented and represented unevenly among different products. Support for such services varies drastically.

There are products that provide bare-bone communication APIs (send/ receive verbs) supplemented with primitive naming services and the ability to run under the control of major TP monitors, such as CICS or IMS. There are also products that provide a full-blown naming service with dynamic service registration and automatic event notification in addition to the basic communication API. IBM's MQSeries and Digital's DECmessageQ provide explicit queuing including transactional message queuing, which is available already on some platforms. Transactional message queuing is very powerful and important for some applications. It allows message queues to act as a regular resource manager such as databases, files and so on. In practical terms, any messages stored on the queue will be backed out unless committed explicitly or implicitly.

There are also publish/subscribe MOM products. Publish/subscribe allows an application program(s) to subscribe to specific information while other application program(s) can produce this information. Neither publisher nor subscriber needs to have any direct communication established between them, nor do they need to know about each other's existence.

To date, in my opinion, MOM products are probably among the more flexible middleware technologies on the market. They provide the best platform coverage -- many MOM products are available on numerous platforms with support for major communication protocols. Also, many products furnish the same API across all supported platforms, including mainframes, where they offer a high performance communication capable of sustaining serious transactional throughput. All this, combined with the fact that most MOM products' resource requirements range from reasonable to very modest, make MOM an attractive alternative for linking new and old applications.

Decision makers need to be aware of the following MOM drawbacks, however. As of this writing, there are no standard APIs for MOM, and there is no established interoperability between any MOM products. In practical terms, this means that all application components or different enterprise applications will need to use the same

MOM product in order to communicate. Different MOM products cannot communicate with each other, at least not out of the box, but a number of advanced consulting firms can offer such an interoperability. Most MOM products support neither native C++ interface nor object-oriented paradigms. Again though, with help from some advanced consulting firms this issue can be addressed. MOM products are not only incompatible, they also provide different levels of functionality and communication paradigms.

The bottom line is that there is no product on the market that supports all communication paradigms, including direct process-to-process messaging, explicit message queuing and publish/ subscribe. It should be noted that at least a few MOM products could add value to ORBs, with features such as high performance communication extensions to the mainframe, naming service or event notification.

TP MONITOR MIDDLEWARE

There is no question that TP monitors provide a lot of the functionality required for enterprise distributed computing. Provo, Utah-based Novell Inc.'s Tuxedo, which probably is the most mature Unix TP monitor on the market today, supports a long list of features. On the communication front, Tuxedo supports all types of communication models, including synchronous (blocking) and asynchronous (non-blocking) connection-oriented as well as asynchronous queue-based connectionless models. Novell also announced that Tuxedo Release 6 will include the API for a publish/subscribe model, which will include automatic event notification. This combination makes Tuxedo a strong contender.

TP monitors as a group offer an impressive number of fancy services, including security with an access control list, load balancing, naming service, thread support, robust administration and others. Once again, support for all of these services differs among TP monitors and implementation. For example, Pittsburgh, Pa.-based Transarc Corp.'s Encina is a TP monitor built on top of DCE services. Thus, it inherits all the strengths and weaknesses of DCE. The strengths could be considered support for services such as security and naming, and the weaknesses could be limited communication models (for example, RPC only), large footprint and so on. At least for now, the same applies to IBM's CICS/6000. It is built on top of Encina and DCE. In contrast, Novell's Tuxedo and Dayton, Ohio-based AT&T's Global Information Solutions (GIS) Top End do not require the presence of any other technology. Most of the services are built into the product itself and there is no dependency on other technologies.

When it comes to mainframe connectivity, each TP monitor provides a different solution. CICS and UniKix, from the UniKix Division of Bull H N Information Systems Inc., Phoenix, Ariz., are the only TP monitors that allow programmers to use the same API across all supported platforms including MVS. For all other TP monitors this is not the case -- their respective APIs are not available on the mainframe, and custom programming to link to existing CICS and IMS applications is required. Usually, LU6.2 CPI-C code needs to be developed under CICS, IMS or any other MVS subsystems to communicate with Encina, Top End or Tuxedo servers. The difference in the APIs and the fact that not many corporate developers are excited about writing low-level LU6.2 CPI-C communication code, makes TP monitors a somewhat less attractive alternative for linking to existing applications. At the same time, TP monitors offer many features and functions that at this point, other middleware vendors do not provide.

So why doesn't everyone buy into the idea of TP monitors, especially in light of the fact that TP monitors are being repositioned as middleware for general mainstream applications rather than middleware for transaction processing only? There are a number of reasons why compared to other types of middleware, non-MVS distributed TP monitors have a growing, but still tiny market share. Some potential customers either do not understand or need all of the functionality furnished by TP monitors and are embracing other middleware technologies such as ORBs or MOM. Some customers prefer a different development paradigm, namely C++ or Smalltalk (TP monitors do not support C++ or Smalltalk yet) and want to take full advantage of features that such languages and object-oriented technology generally provide. These customers either consciously or unconsciously are ready to compromise, at least temporarily, the functionality provided by TP monitors for benefits provided by object-orientation.

Also, TP monitors do not provide the same level of platform support as other middleware technologies, which DCE and MOM do. With the exception of CICS, all other TP monitors support Unix variances as servers, and different Windows, Mac, Unix and X-Terminal versions as clients. Given the market realities, support for NT servers will be shortly forthcoming. Many other platforms including MVS, AS/400, VMS and others are simply left out, at least for now. Of course, more reasons can be noted to justify why TP monitors are not everywhere. This is not the issue. What is important is that TP monitors provide functionally rich middleware solutions. At the same time, a TP monitor might not be the best choice for connecting the new and existing applications, especially when linkages to CICS and IMS applications are required.

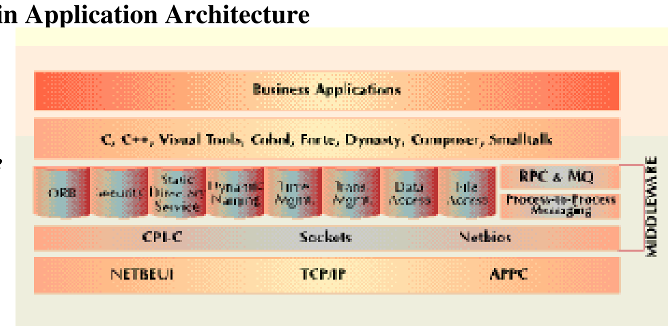
CONCLUSION

Plenty of middleware technologies are on the market. Some are truly competitive and some are not. Furthermore, together, some middleware technologies can be highly synergistic -- future integration of DSOM with MQSeries and existing integration of Orbix with Isis are just two examples where two plus two can be five for corporate developers. Such win-win situations are desirable, especially when it comes to linking new and existing applications together. After all, there is no magic -- some middleware products have their roots in Unix environments, while others have their roots on mainframes. Given the differences in functionality, "religion" and "culture," it is safe to say there is no universal "have it all" middleware product on the market across all important platforms -- at least, not now.

It might be that by bringing different middleware technologies together, some weaknesses of one technology could be eliminated, while some strengths of another technology could be enhanced. As the trend to combine different middleware technologies continues, it should become easier to link new and old applications. Nobody really knows how far this trend will go, but many of us can contribute one way or another to continuity. All there is to say is, stay tuned. We'll keep you posted. *

Where Middleware Fits in Application Architecture

Source: International Systems Group Inc.Fig. 1



Middleware acts as an isolation layer of software to shield business applications and development tools from operating systems, hardware platforms, communication protocols and other lower-layer "plumbing."

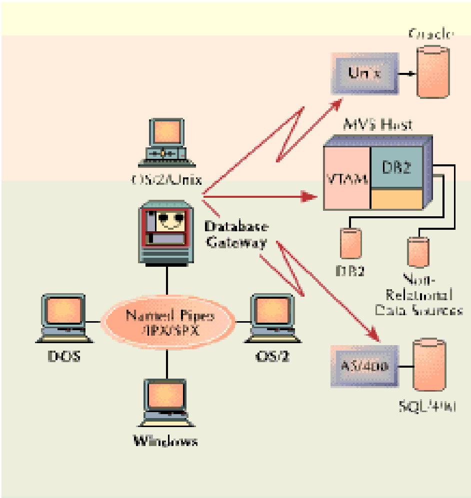
Max Dolgicer is one of the directors of International Systems Group Inc., an independent information technology consulting company based in New York. With over 19 years of technical and management experience, he is an internationally-recognized speaker and consultant to end-user companies and software suppliers. His E-mail addresses are: 6086839@mcimail.com and mdolgicer@aol.com.

The right or wrong choice of middleware can make a tremendous difference across the

board for managers, technicians and end users.

The process transfer is best suited to full three- or even multitier complex applications that require a subsecond response time as well as scalability.

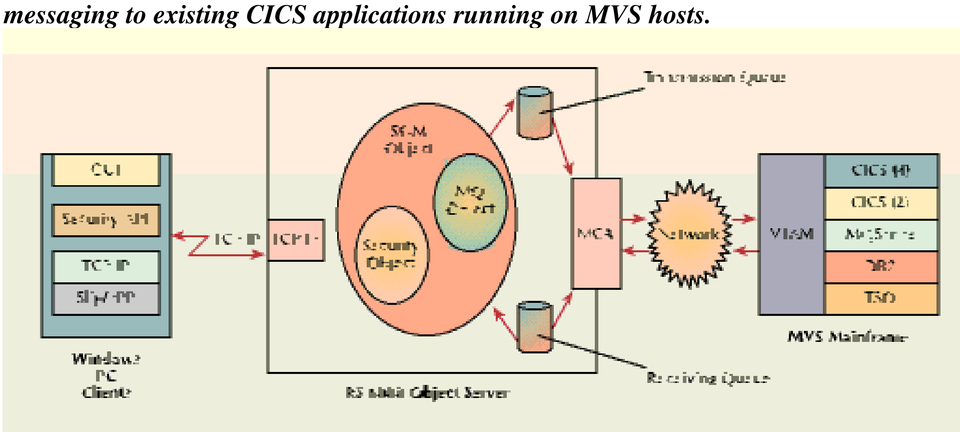
Multidatabase Gateway
Source: International Systems Group Inc.Fig. 2



A multidatabase gateway implies a three-tier architecture in which the client, the gateway and the target database all run on different platforms. Some such gateways require the presence of the associated DBMS, such as Oracle's Open Gateway, and some do not, such as Sybase/Enterprise Connect.

How ORBs and Messaging Can Work Together
Source: International Systems Group Inc.Fig. 3

IBM is planning to integrate its DSOM object request broker with its MQSeries messaging middleware on multiple platforms. With the help of an integrator, a large bank implemented an application that combines SOM and MQSeries middleware on an AIX server, using TCP sockets to Windows PC clients, and the MQSeries messaging to existing CICS applications running on MVS hosts.



The bottom line is that database access middleware is an attractive solution for two-tier, and possibly three-tier, decision support applications where a split second response time is not a requirement and synchronous connection-oriented communication is sufficient.