Articles & White Papers

Middleware and Architecture

by max dolgicer

In the past, development tools determined middleware/architecture.
Today, many companies are finding the opposite to be true.

Should development tools dictate middleware or vice versa?

In the early days of client/server, your development tool was your architecture. Indeed, the application architecture and middleware (if any) were virtually dictated by the functionality of development tools such as PowerBuilder from Powersoft/Sybase Inc., Emeryville, Calif. Consequently, the majority of the resulting applications subscribed to a two-tier architecture that did not include middleware (a layer of software residing between the business applications and the "plumbing layer" of heterogeneous operating systems, hardware platforms and communications protocols). To be fair, two-tiered solutions were acceptable for many departmental client/server applications. Furthermore, there always was (and always will be) a class of applications that should be implemented as two-tier and for which a three-tier architecture, even with the benefits of middleware (for example, universal access to heterogeneous databases and communications protocol independence), would be overkill.

As companies have tried to develop enterprise-class applications, however, the rules of the game have changed, leading to changes in the requirements and vice versa. In particular, the need for multitier application architectures has become more and more apparent. What is not so obvious, though, is where this architecture should come from; that is, who should supply this architecture and how will the architecture reconcile with the development tools and practices of an organization? Increasingly, companies have had to answer two difficult questions:

What comes first, an enterprise architecture or the fast development of business applications?

How should middleware choices that constitute a distributed applications architecture be reconciled with the application development tools?

Simply stated, if a corporation endorses a middleware architecture that is independent of integrated development tools, can tools like Dynasty (from Dynasty Technologies Inc., Naperville, Ill.) or Forte (from Forte Software Inc., Oakland, Calif.) be used for application development, and can those tools take advantage of the chosen middleware products? On the flip side, if a company has selected integrated application development tools, can an independent middleware architecture provide any benefits? This issue of development tools versus middleware was discussed in an earlier column in Application Development Trends (January 1996). This article expands on several points raised by that column.

When middleware comes first

From International Systems Group Inc.'s experience with Fortune 500 companies, we have found two potential scenarios in which the middleware architecture, independent from the development tools selection, comes first. In the first scenario, corporations are establishing a single corporate architecture with middleware being a cornerstone of such an architecture. The rationale is sound and simple -- with an enterprise infrastructure in place, corporate developers can focus on the business requirements of the applications that they are developing rather than on such issues as security, naming/directories, fault tolerance and so on. In this scenario, the middleware choices are open and the probability of selecting best-of-breed middleware is good. It should be noted that, in this scenario, the corporate architecture should not reflect a single business application requirement. On the contrary, the architecture should be based on the requirements of a broad spectrum of existing and future applications. Furthermore, the collection and assessment of such requirements are generally done by an unbiased application-independent architecture group.

The second scenario is generally driven by the necessity of developing a complex business application that needs to be scalable. In such situations, the application developers are basically faced with almost the same challenges as the architecture group in the previous scenario; that is, robust middleware is required. For the second scenario, however, the development effort is localized to a specific application. Nevertheless, it is still a good business practice to design such an application with enterprise middleware in mind. Sometimes the complexity of a single application and the breadth of a middleware solution selected are so substantial that the same middleware solution ends up being used in other applications within an enterprise, thus paving the way to becoming a de facto standard.

Our experience with clients indicates that in the past year there has been a strong trend toward building a single enterprise architecture. Indeed, companies are starting to dedicate separate budgets for architecture development and are establishing architectural groups with a clear charter of delivering a single corporate architecture to business application developers. For example, about a year ago a major financial institution hired a new CIO and one of his first initiatives called for the development and deployment of a message-based middleware backbone that would span many different lines of business as well as multiple computing platforms. There were several goals driving that initiative:

3 1

To provide a universal end-to-end application connectivity, that is, a single API that would enable applications to exchange information seamlessly and in synchronous and asynchronous fashion. By incorporating this API into existing and new programs, business applications that had not been able to exchange information in the past would be able to do so in the future. From a business perspective alone this can yield substantial benefits. First and foremost, different lines of business within this corporation could share information seamlessly, thus enabling management to make better decisions.

3 2

To provide location transparency that would enable business applications (which can be viewed as service providers) and service consumers to locate and communicate with a desired service similar to the way in which a person might look up a phone number by dialing directory assistance. Obviously, in order for a service (or, to be precise, an application that provides this service) to be located, the service has to be registered with the middleware. In fact, the ability of middleware to support dynamic (that is, runtime) registration and deregistration of services eliminates the need to maintain any extra preconfigured tables containing the locations of the services on the network. This, in turn, would decrease the need for network administrators and would eliminate the necessity for creating a more automated environment that was less prone to error.

3 3

To provide a security service that would enable business applications to use a consistent authentication and authorization service regardless of the platform that an application was running on.

3 4

To provide application developers with a common and consistent environment that would reduce complexity and allow developers to concentrate on solving business problems.

For companies implementing an enterprise architecture, support for additional middleware services would obviously be a critical issue. It is important to note that no single product currently on the market offers a complete set of middleware services. This deficiency is especially painful for corporations that are developing an enterprise architecture with very demanding middleware requirements. After all, such companies are trying to accommodate many existing and future applications throughout a large enterprise. In the majority of cases (the financial institution mentioned above was no exception), more than one middleware product must be integrated in-house to compensate for any missing functionality and to create a best possible solution. With any luck, a company will have to integrate only two (and not three, four or even more) middleware products.

To demonstrate this point, consider a hypothetical example in which a particular object request broker (ORB) such as Orbix from Iona Technologies Inc. of Marlborough, Mass., has been selected as the middleware solution. Furthermore, assume that the example requires programs to communicate with each other asynchronously, in time-independent fashion with guaranteed delivery, and that message queuing middleware products have been designed to provide just that functionality. Unfortunately, there is no product currently shipping that would combine the functionality of an ORB and message queuing to allow objects to communicate with each other not just synchronously, but also asynchronously. The solution would be to integrate an

ORB with a message queuing product, which a company could do by itself or by working with vendors. Incidentally, similar integration of middleware products has been done before -- an offering called Orbix+ISIS preserves Orbix functionality, but also inherits the fault-tolerance capabilities offered by ISIS from Stratus Computer, Marlborough, Mass. Nevertheless, the effort involved in integrating multiple middleware products or in developing additional functionalities to existing middleware can be substantial, requiring a combination of skilled architects and developers.

In the case of the financial institution mentioned earlier, the skills set was not a problem. The group that had been chartered to deliver the corporate architecture possessed substantial talent. Equipped with skilled and creative management, experienced system architects and serious C and C++ developers, the group had no substantial problems to meet the challenge. Furthermore, the firm's programming paradigm for business applications development had been C++ for some time. As a result, a substantial inventory of C++ classes for any serious application development had been building up for a while. Given the pool of talent available and the past programming practices, the firm made the right decision to develop an enterprise architecture and to continue developing business applications in C++ using the newly developed message-oriented middleware infrastructure. More than likely, the skill set was the key to success.

As another example, a telecommunications company decided to assemble and integrate best-of-breed middleware for enterprise-class client/server application development. However, after dealing with the complexities of integrating Distributed Computing Environment (DCE), Encina from Transarc Corp., Pittsburgh and Rogue Wave from Rogue Wave Software Inc., Corvallis, Ore., among other products, the company changed directions and decided to acquire an integrated development tool: Forte. Although DCE and Encina are complex middleware products in their own right, the project was compounded by a lack of skills that were required to accomplish such an integration.

Despite this example, we still maintain that the establishment of a single corporate architecture, without being locked into a specific development tool, is a good idea. Corporations that are thinking seriously about such an undertaking, however, should not underestimate the two major challenges of establishing a single corporate architecture:

31

The most complete architecture will derive from a number of middleware products that will have to be integrated.

32

For the most part, complete freedom in selecting best-of-breed middleware choices also means using C, C++ or Smalltalk for developing applications, especially those involving business functions (that is, business services).

To be fair, it should be noted that there are middleware products that already support Microsoft's Visual Basic, PowerBuilder and SQLWindows from Gupta Corp., which is now Centura Software Corp., Menlo Park, Calif., among other development tools. Such support is generally offered by third parties. With IBM's MQSeries, for example, client support for Visual Basic and PowerBuilder can be obtained from third-party products. In other words, a client application using MQSeries can be developed with Visual Basic or PowerBuilder instead of with C and C++. Additional examples -- TransAccess from NetWise, Boulder, Colo., which was recently purchased by Microsoft; Tuxedo from Novell, which is now marketed and distributed by BEA Systems Inc., Sunnyvale, Calif.; Encina; CICS from IBM; and Top End from AT&T's GIS, which is now NCR -- can all support PowerBuilder interfaces, thus providing some relief to the corporate developer. For the most part, however, these interfaces are not yet very mature; they cost extra; and they limit both the use of certain data types and the accessibility to some functionality by the middleware products. Companies should be aware that such solutions are relevant to the client side only; business functions are still being developed using conventional
languages.

Also worth noting is the fact that some ORBs like Iona's Orbix do allow a Visual Basic program and other PC tools (for example, Excel) to invoke a Common Object Request Broker Architecture (Corba) object. Such functionality is offered in the standard Orbix product at no additional charge -- a nice change from previous similar offerings, which required a third-party product that incurred an extra cost.

The bottom line is that companies preferring to decouple their distributed architecture (that is, middleware choices) from application development tools should be aware of the challenges that lie ahead. At the same time, companies that are capable of overcoming those obstacles are going to win big with an architecture that is completely independent from any application development environments or 4GL tools.

When one-stop shopping makes sense

Again, judging from our real working experience with Fortune 500 companies, we have found that there are a number of situations in which the integrated, "one-stop shopping" tools make more sense. First of all, the debate is not about whether or not middleware should have a place under the sun. For the most part, this debate has already been settled: Most people agree that middleware is one of the key components for every enterprise-class multitier application. This is a nonissue, at least for now. Even Microsoft acknowledges that middleware is a necessity for any serious application and the company is offering Object Linking and Embedding (OLE) Services/Common Object Model (COM) including the still-pending Network-OLE (N-OLE).

The real issue is whether or not the middleware that is provided by development tools is good enough. In other words, does the middleware component, which for the most part is built into the development tool, offer enough services and functionality in comparison with middleware that is available from traditional middleware suppliers?

To put it plain and simple, corporations need to ask themselves whether or not application development tools should become their distributed architecture. For the most part, the middleware services offered by integrated development tools do not measure up to services provided by general middleware vendors.

For example, Forte, one of the most powerful development tools, does not currently support message queuing or external transaction processing (TP) monitors. (Note: Encina support has been announced, but is not yet generally available.) Forte does provide, however, automatic failover and distributed transactions across homogeneous databases -- features that are ordinarily found in TP monitors. Still, distributed transactions for updating heterogeneous databases are not supported; a TP monitor is required to accomplish this. Furthermore, Forte by itself does not provide access to legacy applications and data. To address this deficiency, Forte relies on third-party middleware vendor products such as TransAccess.

Dynasty (Forte's main competitor) does support Tuxedo, which is one of the most mature and powerful TP monitors on the market. Using Dynasty, developers do not need to code to ATMI (Tuxedo's API) because Dynasty itself can generate all the necessary code. Tuxedo support is also provided by the JAM development tool from Jyacc Inc., New York City.

Although the independent middleware suppliers generally provide better middleware than the development tools vendors, the latter furnishes developers with powerful 4GL tools that are capable of developing multitier, client/server applications. By using a single 4GL tool, all parts of distributed applications including the business logic can be developed without the use of conventional languages.

Use of a high-level 4GL tool (as opposed to C or C++) will cause a certain loss of control that some developers might desire over the building process of the application. Many developers, however, are ready to compromise such tight control to avoid conventional coding. This is also true for object-oriented (OO) developers who would otherwise be forced to develop in C++.

An example will help to illustrate this point. A commercial software development company that specializes in providing packaged applications for the health-care industry decided to convert one of its major single-tier systems to a three-tier OO application. The company evaluated its application development and middleware choices very carefully. During the evaluation, it became quite evident that building each application tier using the best tool for the job would require C or C++ development. This was especially true for the middle, or application services, tier. I still remember the quote from the company CEO. During one of our meetings, he said, "I have nightmares when I'm thinking that my whole development team will be coding in C++. We simply don't have the existing skills, nor do we have the time to acquire them." Listening to that CEO, I realized that an integrated development tool was the answer. Three months later the company opted for Forte and the development process got under way.

In summary, an integrated development tool is a good fit when:

A specific line of business application needs to be developed by a certain fast-approaching date.

3 2

There is a lack of skilled personnel capable of developing applications in C++ (and other modern 3GLs).

3 3

A heavy investment in developing applications in C++ (or other 3GLs) has not yet occurred.

Companies opting for a single integrated development tool should, however, be aware of several drawbacks. First, most of the 4GL tools are proprietary, whereas no one would call C++, or even Visual Basic, a proprietary language. Second, when it comes to middleware selection, the only choices available are either the ones that are supported by the development tool (for example, Tuxedo for Dynasty) or offered by the tool itself (for example, Forte has a built-in messaging-based middleware component within its product). Third, when it comes to building application graphical user interfaces (GUIs), developers are advised to stick with the facilities that are available within the development tool. Officially, tools like Visual Basic and PowerBuilder are supported for GUI development. In practicality, however, such tools are not well integrated with other elements of the development environments (that is, the repository, version control and so on). Lastly, no matter how powerful a 4GL tool is, it simply cannot measure up to the flexibility that can be obtained by using a 3GL.

Middle of the road?

In closing, I should mention that there are a number of products that can be positioned somewhere between being a development tool and middleware. Representative of such products are Boston-based Open Environment Corp.'s Entera and New York City-based Magna Software Corp.'s Magna X. In the case of Entera, developers can decide on what tool (for example, Visual Basic or PowerBuilder) to use for the GUI part of their application. When it comes to middleware, Entera offers support for either DCE or Entera's own TCP/IP-based middleware. Entera, however, does not generate any business logic for the middle tier. For such functionality, Magna X generates Cobol code for the middle tier and also supports a generation of Tuxedo or Encina as middleware choices.

Although these middle-of-the-road products provide several welcomed choices in selecting the front-end tool (that is, Visual

Basic or PowerBuilder) and middleware (for which options are much more limited), a substantial effort is still required when it comes to delivering an enterprise-class distributed application. Although products such as Entera and Magna X decrease the amount of integration required to glue all of the pieces together, developers should be aware that their choices of middleware are limited to those already supported by the product.

To summarize, corporate developers need to ask themselves about their need for an "open" architecture versus the ability to have one single tool for both application development and middleware. In the final assessment, much will depend on the corporate culture, skills and, of course, other factors such as the costs and politics involved.1

Pop quotes:

No single product currently on the market offers a complete set of middleware services

To put it plain and simple, corporations need to ask themselves whether or not application development tools should become their distributed architecture.

Middleware Services and A/D Tool

Source: Forte Software Inc.Fig. 1
Middleware services from the view of an application development tool (in this case, Forte from Forte Software Inc.).

What is middleware?

Simply defined, middleware is an enabling layer of software that supports multiple communication protocols, multiple programming languages and multiple execution platforms. Middleware resides between the business application (the "upper" layer) and the network of heterogeneous platforms and protocols (the "lower," or "plumbing," layer). What is important is that middleware decouples business applications (either packaged or developed in-house) from any dependencies on the plumbing layer of operating systems, hardware platforms and communications protocols. One benefit is that such independence provides I/S managers with the freedom to decouple their buying decisions of systems and applications software from this plumbing layer. With middleware, it does not matter whether packaged applications are used or whether those applications are developed in-house because I/S managers are no longer confined to specific development and runtime platforms, nor to specific communications protocols (see figure below). This freedom, of course, assumes that the middleware supports multiple platforms and protocols.

For application developers, middleware delivers several tangible benefits:

31

Corporate developers can focus on business application requirements rather than on having to develop low-level middleware code.

32

Business applications can be developed independently of the operating systems and communications protocols. Obviously this independence yields substantial benefits such as application portability and investment protection. As new operating systems become widely available, changes to the existing applications will not be required.

33

Best-of-breed packaged applications can be purchased. Thus, the purchasing decisions can be made based on the functionality of the packaged software rather than on the ability of that software to run in a specific environment, assuming, of course, that the purchased software has been architected with a middleware layer in mind.

The loose definition of middleware reflects the reality that is taking place in the market. Over the past three to four years, middleware has become one of the most overly used terms. One strict definition categorizes middleware into five types of products:

3

DATABASE MIDDLEWARE,

3

MESSAGE-ORIENTED MIDDLEWARE (MOM),

3

PORTABLE TRANSACTION PROCESSING (TP) MONITORS,

3

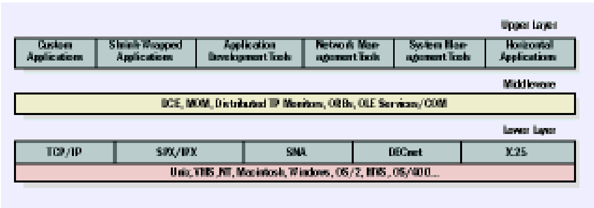
OBJECT REQUEST BROKERS (ORBS) AND OLE, AND

3

REMOTE PROCEDURE CALLS (RPC)-BASED MIDDLEWARE.

Developers should be aware that, as the middleware industry continues to evolve, these definitions may also evolve. One belief is that consolidation rather than further fragmentation of middleware types will occur in the future. Such consolidation might result from mergers and acquisitions of middleware companies, and from products belonging to one category of middleware beginning to offer functionality that is currently being provided by another type of middleware product.

Where Middleware Fits



Three-Tier Application with Middleware

Source: International Systems Group Inc.Fig. 2
Among other various services, middleware is the glue that integrates different tiers (either physical or logical).

Middleware services

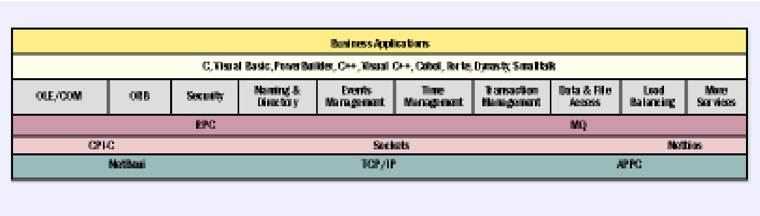
The obvious value of middleware is that the technology supports uniform access to heterogeneous databases and provides standard methods for performing program-to-program communication. The $64,000 question is whether these features alone will be sufficient to propel middleware technology to the forefront. In other words, does middleware need to offer additional features and services beyond heterogeneous database access and standard program-to-program communication?

One opinion is that good middleware products should support a multitude of additional services, which could include synchronous (process-to-process) and asynchronous (message queuing) communication between application components, naming and directory, security, load balancing, transactional integrity across distributed databases (both homogeneous and heterogeneous) and other resources (for example, files, queues and documents), data marshaling, event notification, fault tolerance, instance management, data-dependent routing, dynamic registration and discovery of business functions and objects, seamless integration with legacy systems, and application partitioning (see figure above).

At a minimum, middleware should facilitate the development of platform-independent applications. Although considered a giant leap forward just a few years ago, such a benefit is almost taken for granted today. Indeed, many middleware products are currently providing platform independence.

At a maximum, middleware should provide certain services (listed above) that are essential to developing and deploying scalable, enterprise-class distributed applications. From an application development point of view, the middleware layer should provide a set of platform-independent APIs that expose to the developer such services as communications (synchronous or asynchronous), naming (locate, advertise and de-advertise), transaction management (transaction begin, transaction end, transaction commit and transaction abort), event notification, timeout management and data marshaling. Or, those APIs could, in some cases, be generated by the development tools. Other functionality -- for example, load balancing, fault tolerance and instance management -- could simply be provided as built-in features within the selected middleware product(s). Obviously, support for additional services increases middleware's chances of becoming the key component within a distributed infrastructure throughout a corporation.

Examples of Middleware Services



Source: International Systems Group Inc.