FINANCIAL MIDDLEWARESPECTRA Volume 13, Report 1

Contents — *FINANCIAL* MIDDLEWARESPECTRA — February 1999

2. **Object models, information flow and middleware**
Ron. Thieme, President, SCG partners

10. **Managing successful middleware projects**
Mark Allcock, Vice President — Global Middleware
JP Morgan Asset Management Services

16. **Processes, security and middleware: hole or whole?**
Phil. Manchester, Consulting Editor, **MIDDLEWARE**SPECTRA

22. **Managing middleware**
Will. Cappelli, Analyst, Giga Information Group

30. **Emerging and converging middleware: application servers and message brokers**
Keith Jones, Senior Software Consultant, IBM Corporation

38. **Is inter-operability between object types feasible?**
Rosemary Rock-Evans, Consultant

46. **Enabling EAI with MQIntegrator**
Jay Lang, Chief Technologist, Distributed Computing Professionals

Object models, information flow and middleware

Ron. Theime
President
SCG Partners

Management introduction
The strategic use of middleware by the financial sector is hardly a secret. It is, however, often assumed that the financial sector has a distinct lead in the deployment of strategic middleware — particularly in the context of managing risk and co-ordinating multiple activities which have critical implications.

In this interview, Ron. Thieme (and a team of his object modelers, designers, developers and project managers at SCG Partners, of Nashua, NH) discusses how Delta Air Lines has evolved the application of middleware to support any number of activities which are critical to the customer and operational success of the Airline. While middleware is at the base of the solution, this interview is less about the middleware itself and more about how a team has exploited it to support a complex and constantly dynamic business environment.

Superficially this might not seem relevant to the finance sector. But, as emerges, the challenges which Delta faced — and solved — are similar to those found in the financial sector, albeit expressed in different metaphors.

All rights reserved; reproduction prohibited without prior written permission of the Publisher.
© 1999 Spectrum Reports Limited

Business drivers

Like any business, Delta is driven by business pressures to provide excellent customer service and operational performance while controlling costs. The lifeblood of airline operations is information, including events which represent the continuing changes and exception conditions that arise every day. The applications I shall describe in this interview collect that information from myriad sources and move it in the form of messages to users and application programs throughout Delta. Several of the applications also analyze incoming information and generate new events representing exception conditions. One challenging aspect of airline operations is that the data involved is connected in a complex web of relationships.

SCG Partners worked with Delta over a period of years on the development of an event-based information system to support operations as well as to improve customers' experience when flying with the airline. This system, which makes heavy use of middleware, now encompasses several major applications and has been driven by three strategic imperatives which will be recognized by virtually any business in any industry — the need to deliver:

- **excellent customer service**
- **operational excellence (which for an airline means on-time performance)**
- **wise use of its resources.**

As any traveler knows, airline operations are a dynamic environment in which inevitable daily events — thunderstorms, mechanical problems, missed connections, etc. — force constant adjustments to an airline's operation and to passenger itineraries. In order to function, the whole organization requires timely information. SCG's work for Delta has focused on the collection, generation and delivery of that information.

Passenger rebooking

Take, for example, passenger rebooking. This is a 'feature' with which most travelers are familiar. But it is also a task which, if it is to be executed well, requires a great deal of information. Accordingly, it offers a recognizable introduction to the problems faced.

Delta strives to make flying as positive an experience as possible for its customers. However, 'things happen'. Every so often a flight is cancelled, or a connection is missed. Passengers are inconvenienced by such events and need to be rebooked on other flights. To accomplish this they must:

- **learn what alternatives exist, including those on other airlines**
- **obtain a seat, if possible, on one of these alternatives**
- **if an overnight delay is involved, make accommodation arrangements**
- **cancel or rearrange meetings or other plans at the destination location.**

Moreover, passengers want to complete these steps quickly, lest he or she miss an excellent alternative. What a passenger does not want to do is:

- **board an alternative plane that itself becomes delayed or cancelled**
- **discover — too late — that the original connection could have been made (perhaps that flight had also been delayed).**

Therefore, if Delta wishes to make rebooking as smooth as possible for customers (for instance by making new arrangements and handing to passengers revised tickets as they step off their incoming flight), then it will need a great deal of accurate, up-to-date information in a short time. Rebooking 120 passengers on a flight that arrives late into a busy hub might require upwards of a thousand alternative flights to be looked at within a minute — as well as consideration of information about the passengers involved (itineraries, special needs, etc.).

This is only part of the problem. Flight cancellations are even more complex. While an airline's schedule has plenty of capacity to handle the airline's daily flights, a severe weather problem at one hub can delay operations nationwide. Furthermore, when operations start to go bad they tend to become still worse: in such conditions the airline must proactively select flights to cancel. This requires tools to help identify the actions which will cause the least disruption.

Thus, in actual practice, rebooking means:

- **not merely reacting to events, but anticipating problem conditions (such as likely missed connections, or overcrowding at a hub) and reducing or avoiding problems by identifying and calling passengers who can be rebooked onto alternative itineraries**
- **providing employees with correct and actual information (erroneous information, acted on aggressively and proactively, only tends to make matters worse)**
- **possessing sufficient information to ask 'intelligent' questions: for example, instead of selecting a flight for cancellation based on 'which plane has the fewest number of passengers?' Delta staff need to know enough to avoid making choices which might (say) oblige a minor or a person with a medical condition to stay overnight.**

To provide this level of service requires a wide variety of information, derived from multiple systems:

- **some based on schedules and reservations**
- **some based on actual flight operations**
- **some (such as anticipated missed flights) generated in real time.**

The need is to assess the significance of events as they occur. Given the dynamic nature of airline operations, this requires a continual flow of information. At Delta the availability of this information is not the result of some recent bright idea. It comes from the long term development of a comprehensive airline information system — and the flow of information is based on middleware.

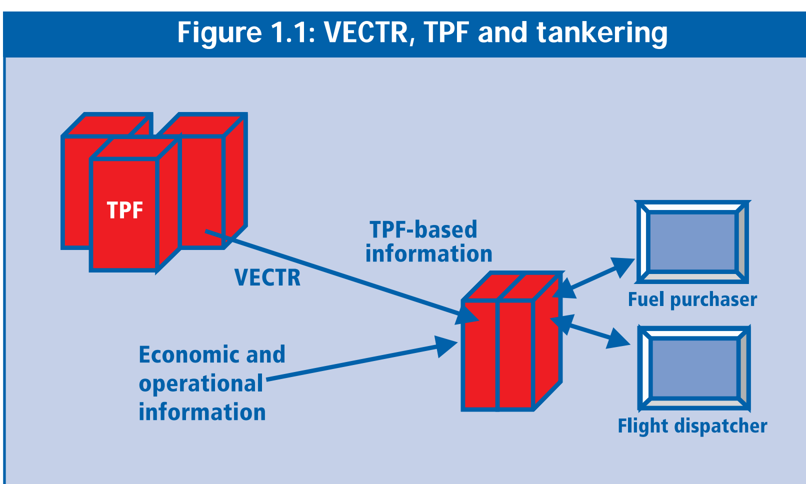
How this started: fuel purchase optimization

SCG Partners' work with Delta began several years ago, with a fuel purchase optimization — 'tankering' — application. In tankering, an airline tries to purchase and carry fuel so as to minimize fuel costs. For instance, it might pay to carry an additional 8000 pounds of fuel to Florida rather than refuel there, since Florida has a high fuel tax. The task of the tankering application was to calculate how much extra fuel a flight could and should carry — something that dispatchers previously had figured by hand (when possible) but skipped whenever operations became hectic. This one application provides Delta with annual savings in the tens of millions of dollars.

In order to compute the tankering it is necessary to know all the events associated with a flight. Once this had been achieved Delta possessed a real time model of the current state of its fleet, which it has been exploiting and extending ever since.

To gather the necessary information, Delta had to build an extensive infrastructure. Today this captures all the major events generated by airline operations, from all the flights and from all aircraft — anywhere. SCG helped Delta to build the message oriented middleware (known as VECTR) which delivers reliable, relevant data from the host Transaction Processing Facility (TPF) systems out to the distributed systems which collect the data needed to make tankering recommendations (Figure 1.1).

Figure 1.1: VECTR, TPF and tankering



All this was deployed in the early 1990s. Today, Delta captures at least 250,000 events per day. Moreover, VECTR is used by thousands of people. When you call Delta's reservations, your sales agent is using VECTR.

Building up: delivering flight event information to 'consumers'

What SCG and Delta did next was to extend the capabilities and create a system called 'FPES' that delivers flight events in near-real time out to the applications and people that

need the information. With FPES Delta can tell how well it is doing 'right now' in terms of operational reliability — at several levels, from the whole airline or at any individual station.

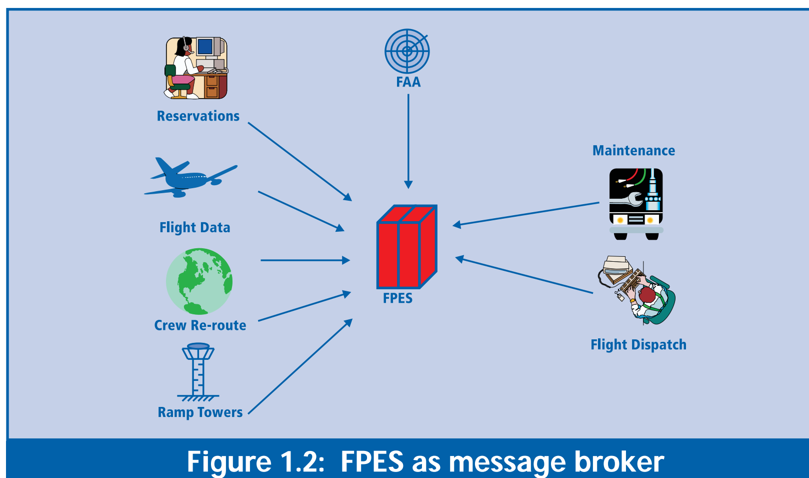


Figure 1.2: FPES as message broker

The information, which is gathered via middleware and assembled by means of a message broker, comes from many sources (Figure 1.2) including:

- **basic information from the airline's TPF-based system, via VECTR**
- **gate information from airports through other systems**
- **passenger information from the reservation system**
- **in Atlanta, estimated takeoff and landing times (supplied through an FAA-sourced data feed called Surface Movement Advisory Messages)**
- **actual landing times from the planes themselves (generated and transmitted through a system called ACARS) or from an alternate source of information for those planes (727s) not equipped with ACARS**
- **maintenance.**

Assembling the information involves more than just merging it by means of Delta's message broker. Some of the data (from TPF) arrives in large chunks instead of discrete events: in this instance the application service must determine what is different from that which was received in the previous chunk and then generate the events representing each of the changes.

The server that runs this application:

- **keeps in memory the exact state of flights (and of the whole airline) from an operational point of view**
- **pushes these events out to clients that 'subscribe' to (request) them, according to several filtering mechanisms**
- **stores the events in a database of current flight information — which is accessed by a number of applications, including customer enquiries via the Internet**
- **formats and stores historical information for later analysis.**

Several aspects about the technical structure of this server are significant for this (and other) applications:

- **incoming data is analyzed: new events, of interest to users, are generated based on this analysis (the system does much more than simply move information around)**
- **the 'state of the airline', including all the connections between the objects, is maintained only in memory: it is this approach that provides the performance needed to make the whole solution feasible (but it is backed by databases from which the state can be reconstructed if/when necessary)**
- **events are 'pushed' to internal consumers: airline employees no longer must enquire to find out if there is relevant information — rather the application will proactively tell him or her about events of relevance in which they are interested, as they happen.**

Flight status monitor: managing by exception

After passenger rebooking the next application Delta and SCG Partners built was the Flight Status Monitor (FSM). This application visually demonstrates what middleware can achieve, because the

changes occurr on the dynamic display reflecting real events (in real time) communicated over Delta's various networks (Figure 1.3) about what is happening with current flights. It is deployed in ramp towers at Delta's hub cities (a ramp tower controls the movement of planes on the ramp, which is the ground between the gates and the FAA-controlled taxiways and runways). FSM can even be accessed remotely, for example from laptop computers over dial-up lines. It enables the ramp towers to 'manage by exception'.

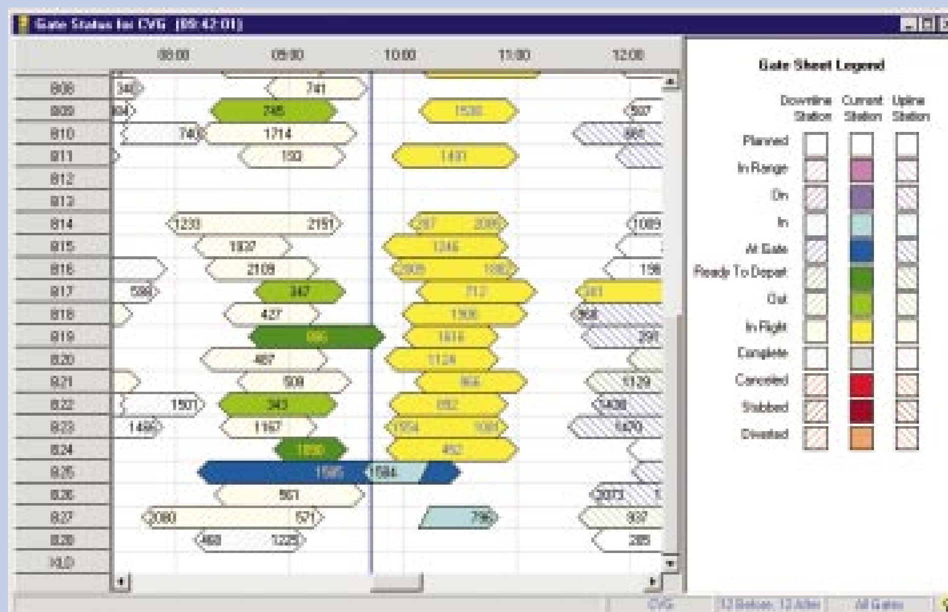
The FSM application uses the same server and much of the same information as did the previously described applications. It analyzes the state of the airline as the events occur and applies business rules in order to identify 'irregular operations'. When it finds these, it generates alerts — presenting these on the workstation as red flags. By clicking on a flagged flight, a user can drill down and determine the nature of the problem. FSM performs several functions, including:

- **displaying information graphically and in real time**
- **detecting exceptional situations by correlating data and events, and then displaying these in the form of warning graphics so that operations personnel can manage the exceptions**
- **selecting, filtering and pushing exception information to clients based on what clients have asked to receive**
- **providing an enquiry function — supporting queries such as 'tell me the state of things' or 'tell me the flight history and what events occurred on this flight' — which helps operations understand the background to unexpected situations**
- **serving as a source of consistent information, a system of record, so that airline passengers can be given timely, scrubbed and consistent explanations — rather than equivocation and inconsistent reports — when events do not go to schedule.**

Exceptions — not only reporting them but also analyzing their significance — are what airline people are attuned to. For example, perhaps a crew has not filed final weight data and yet it is within 10 minutes of departure; or an inbound plane is late now and, given the estimated time of its subsequent departure, there is not enough ground time to service the plane.

Early information of this sort, analyzed to determine its significance, enables staff to react proactively to exceptions. Because FSM pushes

Figure 1.3: FSM



information to Delta's staff as soon as it is available — rather than providing it only when they ask — staff know more and earlier and so are able to make informed and timely decisions (Figure 1.4).

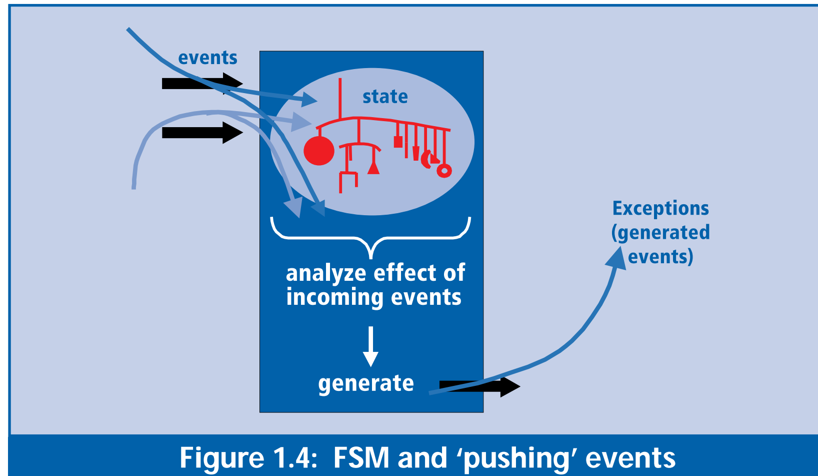


Figure 1.4: FSM and 'pushing' events

Timely, available information can also help turn unexpected changes into opportunities. For example, when an outgoing flight's departure is postponed, if the information is immediately and widely known, Delta can assist other late-arriving passengers to connect who might otherwise have missed or not known that that flight is still going. FSM brings the focus right in to solving the problem; it enables Delta to make the associations which indicate possible changes so that these can be addressed before any single one becomes a major problem.

Success factors

The Flight Status Monitor application is, unsurprisingly, popular at Delta. It is worth summarizing some of its success factors.

The first point is that Delta's users were involved from the first day. FSM was to be their product and their design to solve their business needs. Bolstering this was the impression that users felt that management and IT were not only heeding their requests but recognizing their expertise. The development team — both SCG and Delta — took the position that users knew their business and what their problems were — that they had been experiencing and thinking about these issues for a long time.

The development team also tried new ways to keep users involved. In some cases they were able to design right in front of users — as soon as users came up with an idea SCG would actually show it on the screen. In other cases a joint application design (JAD) session would occur in the morning where the users would say, "what we really need is ..." or "if we could only get timely information when this event occurs ..."; by the afternoon a live application demonstration would be available containing the ideas from the morning. In many instances, live data was used. This made the project exciting for everyone — and it encouraged the users to generate ideas.

Later, at the time of the 1996 summer Olympics, Delta had the first prototype running in Atlanta. The workstation was off in a corner but there was often a large crowd around it watching the event reports (such as 'wheels up') being displayed within seconds after the actual events occurred. Indeed, early users of the system often had information before the people working the flights obtained it. A flight would land and someone in the tower would say 'Flight 888 has landed'. Those watching the monitor would say 'Yes, and the time was 11:23' (which proved the system because to identify a flight you must read the tail number on a plane on a runway some miles away). Or a plane would be parking at the jetway and the users of the new system could report 'they just opened the door' (this action cannot normally be seen from the tower). All this encouraged use and built faith in FSM.

Another success factor was one particular visionary within Delta with whom all liked to work. This person was on the business side, not in IT. He understood how processes ought to work — and he also understood the potential of IT. This person's contribution to success was significant because he:

- **captured what the business problems were**
- **had great ideas about how problems might be solved**
- **was popular within Delta because he showed that he really cared about what people thought — he brought the people working on the line into the process and made them feel part of the process.**

For SCG this individual helped in the design. He thought in terms of managing by exception. "We airline people do not have time," he emphasized, "to look at mundane information about things going right. We need to know about problems as early as possible, because there are things we can do to make customers' lives easier — if the information arrives soon enough". He was able to drive this point across and generate enthusiasm about it.

The lesson we learned here is that a mutually beneficial relationship, between implementers and an effective leader in the user organization, is an important success factor. Delta people were just as critical to the success as our input.

Objects, their implications and lessons

The applications I have been discussing also illustrate the complex web of information that is involved in operating an airline. One crucial point, however, needs to be made here. The use of middleware helps the data get to where it is needed. But it does nothing to address the information content.

SCG and Delta mastered complexity by developing and using an object model encompassing airline operations (Figure 1.5). The adoption of that object model came, in part, from SCG's intellectual history — many of the principals at SCG came out of the artificial intelligence industry where object technology had its birthplace. But it also came from Delta's realization that architecting and implementing applications in an object oriented way provides critical benefits.

An object represents an idea. Objects allow one to think about design in a way that is close to how the business thinks about business problems. As such, objects offer a good meta language with which to think about:

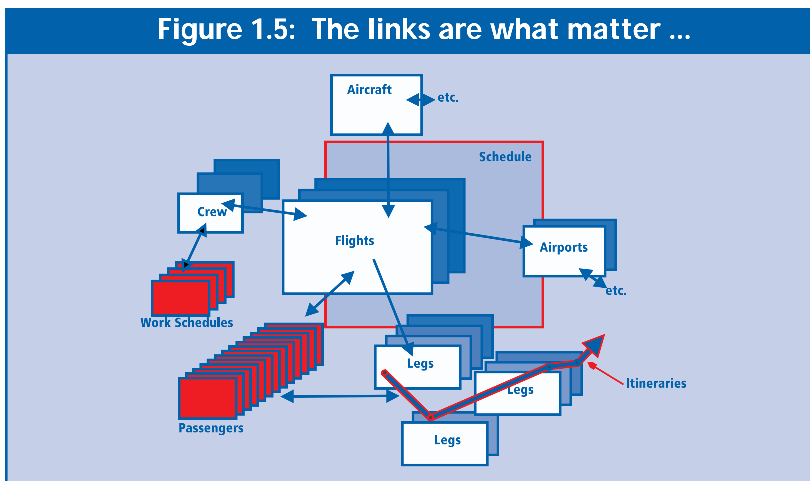
- **data**
- **the connections between data.**

For example, it is much more intuitive to talk about flights, segments, legs, ships, passengers, crews, itineraries and gates than to talk about relations between rows in a database. Add to this variety the many types of relationship among the multiplicity of 'objects' in an airline, and it is not clear that success would be possible without an object oriented approach. However one eventually implements, SCG has found it best to model the business problem in terms of objects.

After you understand the individual objects, then you must think through the relations between them. It is here is where you must design your object model with care. The value in the information lies in the links. The airline industry has complex objects with many connections between them. Thus the data is navigational by nature. Indeed the real value is in the connections between the objects (rather than in the objects themselves). A passenger is connected to a flight connected to a ship connected to a schedule.

The key point is that any of these can become important when you have to take an operational decision. It would be difficult (if not impossible) to design databases and formulate SQL statements that could gather all that you need to know at any time — and still supply good performance. In contrast, with an object system and suitable supporting (middleware) infrastructure, you can navigate the links through all that data.

Figure 1.5: The links are what matter ...



Distributed objects and lessons learned

At Delta, SCG designed applications to maintain the object model throughout the system. A server maintains a complete object model including flights and aircraft and passengers along with all of their attributes and interrelationships. It receives and analyzes all events affecting this model. The server also 'serves' this information to the various applications.

Realize, however, that applications — or clients — generally are interested in only a subset of the overall object model. Therefore, the server distributes to clients only the events that affect the objects that those particular clients are interested in (Figure 1.5). When a client initially accesses a new object (by sending a message to the server) the server:

- **sends to the client the current information about that object (for example, a flight)**
- **notes that this client is interested in the object(s) just discovered**
- **subsequently needs to keep such interested clients up to date by pushing appropriate events out as new (relevant) events occur.**

For the programmers, an interface is provided that enables the programmer to navigate around the objects in the model. Programmers do not need to think in terms of different relational tables, one with passengers, another with flight segments, another with flight legs and another the type of planes on those legs. Instead, they simply go through collections of passengers or segments or whatever to obtain the appropriate information. With objects they just call methods to obtain whatever information they need.

Looking back, we (at Delta and SCG), were able to leverage much of what was learned when working with the tankering and FSM applications into the delivery of an object oriented environment interface which uses existing systems. To the application programmer, it appears that a distributed object system exists — although the reality is that there is no such technology in the system.

The transmission of the event information is done through messages, over middleware. Clients use the information they receive from the server to maintain those objects with which they are concerned. By achieving this, Delta is able to manage the complex, interrelated and voluminous data that is required for its operational business solutions.

Management conclusion

SCG Partners has built a series of applications for Delta which display an ever-increasing sophistication and re-use, starting from the solving of the tankering challenge. Each application extracts new value out of information by analyzing it and by moving it to all interested users, in real time. These applications create value because they enable people to take informed decisions.

As with other middleware projects, a deep understanding of the data was a prerequisite to success. Indeed, in the Delta instance, this understanding is embodied in a comprehensive object model of the Airline. It was this which provided the foundation on which the applications could be built.

In keeping with the object paradigm, recent (as well as future) applications are not and will not be given access to underlying databases, but instead must invoke application services via message. In fact, by the time the Flight Status Monitor was completed, SCG Partners had provided both an object oriented model as well as an object oriented interface for application programmers to use.

For the finance sector, one other point is worth noting — that the applications include not just movement of information but analysis as well, in order to detect exceptions. Actual incoming events are collected by a server which relates each new event to an internally maintained 'state of the operation' and, through analysis in real time, detects anomalous situations (irregular operations or exceptions) and publishes alerts in the form of new event messages. The relevance to (say) exposure control or compliance could not be clearer. And for those who suggest that a 'whole model' is not practical, Delta has disproved this.

While middleware technology is important here, the key issue was putting information to work for the business — by adding value to it through analysis and then by distributing it to those who need it and in time to be useful. But to exploit middleware requires more than just the middleware itself: it needs additional technologies — in the shape of models, architecture and methods — as well as personnel commitment.

Managing successful middleware projects

Mark S. Allcock
Vice President — Global Middleware
JP Morgan Asset Management Services

Management Introduction

The market for middleware in the finance sector is, and has been, evolving at an incredible pace. This pace of evolution is proving to be challenging for both vendors and those adopting middleware. Indeed, experience demonstrates that delivering successful middleware solutions is defined by much more than simply matching requirements to product functionality.

In this analysis, Mark Allcock outlines some of the essential factors to be considered when selecting and managing products in what is still an evolving middleware arena. Mr. Allcock considers:

- ***whether middleware justifies a different project lifecycle***
- ***the need to define such a 'Middleware Development Life Cycle' (MDLC) before selection***
- ***the implication of 'faster, better, cheaper' middleware***
- ***the creation of a middleware habit***
- ***essential steps for managing your middleware vendor(s).***

All rights reserved; reproduction prohibited without prior written permission of the Publisher.
© 1999 Spectrum Reports Limited

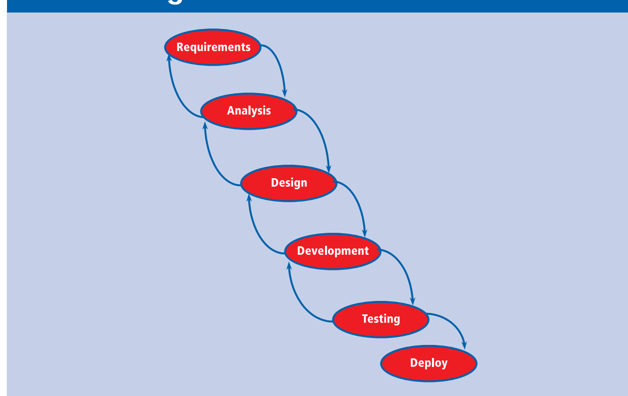
Does middleware justify a different project lifecycle?

Achieving success with middleware — and thereby realizing your business objectives — can frequently be improved if you review how you traditionally approach software development projects and how this might change when introducing middleware. Unless you do this there is a real risk that you will fail to comprehend how middleware will introduce small numbers of often subtle but significant changes to the conventional system development lifecycle.

Traditionally, a typical software development life cycle (SDLC) iterates through a number of analysis, design and development phases in order to arrive at a deployed software solution. One well-known variant of an iterative SDLC is known as the 'waterfall method' — its name being based on the fact that, when depicted graphically, the flow from one phase to another is said to resemble a waterfall (Figure 2.1).

While middleware can certainly be applied successfully in a traditional and project specific manner, my experience is that maximum benefit is achieved when you take the time to consider how your chosen middleware is likely to affect your SDLC — and adjust this accordingly.

Figure 2.1: Traditional SDLC



In my terms, such a modified SDLC constitutes a 'Middleware Development Life Cycle' or MDLC. What is different is that an MDLC considers the likely impact of the ways in which middleware products and tools are used and how best to exploit them. Experience shows several specific aspects requires emphasis, including:

- **the on-going strategic impact of introducing the middleware itself**
- **the effect of middleware on tools and skills partitioning.**

Thinking strategically — in advance

The impact of these is readily understood if you consider that the initial decision to invest in middleware is typically justified by specific project needs (rather than any over-arching strategy). Experience shows that, having successfully introduced middleware in a number of projects, organizations decide to build on that success and move towards the adoption of middleware as a strategic building block. Frequently this occurs via a major deployment and/or via mandates to promote the selected middleware throughout the organization.

By this time, with a number of projects already deployed (and potentially many more in development), moving from an individual project orientation to a strategic all-encompassing one can be difficult. Too often different project teams may have created their own standards, frameworks, utilities and plans — which are incompatible with one another and even with the proposed middleware (which may be different to that originally introduced). Such teams may be unwilling simply to exchange what they have developed for what they view as 'the latest middleware silver bullet'.

Overcoming — or, better still, avoiding — this sort of problem is vital. A strategic plan for middleware should be adopted and communicated for, and before, the first candidate middleware project is initiated. Mapping out a clear vision — a strategic blueprint — to communicate what middleware will be used for, and the timeframe over which it will be used, is essential if success is to be attained.

Once described, the strategic blueprint should be communicated broadly and widely to explain which projects will use middleware, and how that use will be extended. Put simply, the need is to position your applications and systems in a middleware context.

With this vision must come guidelines for selection and usage. Middleware tools are not typically 'one size fits all'. Furthermore, in order to realise strategic advantage, it is important to select projects where middleware will be appropriate — using success driven criteria. Depending on the culture and objectives of your organization, selection criteria will be a mix of qualitative and quantitative measures, most likely including:

- **business based cost/benefits for each project, and the contribution middleware can make (compared to a solution implemented without middleware)**
- **the contribution of middleware in reducing initial and ongoing costs**
- **the duration of each project**
- **the degree of technical suitability and risk**
- **the productivity (operational) benefits.**

Everyone will understand the importance of metrics in developing selection criteria. What too few do is carry the importance of metrics across the lifecycle. Without this you diminish your chances of success.

Middleware can blur development distinctions

Another important consideration for successful middleware projects is to understand in advance how middleware products and tools have been designed and implemented. Many middleware products obscure the distinction between the phases in a traditional SDLC. For example, message brokers blur aspects of analysis with development, making it either inappropriate or inefficient to follow a traditional waterfall SDLC.

Consider Figure 2.2 which represents a comparison between phases and activities in a traditional SDLC with those in a typical MDLC. The traditional SDLC approach separates these phases and activities in order to:

- **allow for iterative refinement**
- **reflect the different skill sets (say) of business analysts and programmers.**

Figure 2.2: SDLC vs. MDLC

Traditional activity	Traditional SDLC Phase	Middleware activity	MDLC Phase
Identify requirements	Requirements	Gather requirements	Requirements
		Determine viability of using middleware	Middleware assessment
Evolve requirements	Analysis	Define integration framework/ flows	Middleware architecture
Create system design	Design	Define messages Define and test transformations Define and test message routing	Middleware development
Develop program design and code	Development		

But many middleware products today support highly graphical and intuitive interfaces, rather than relying on language coding. The good news is that the business analyst is able to undertake much more of the work of a number of the phases as a single activity. In contrast, the traditional SDLC places emphasis on getting the analysis and design iterations right to ensure that the costs and time to develop remain predictable.

With middleware tools, the emphasis on analysis and design is more critically important because the software creation/development has blended with the analysis. This creates one of the subtle but significant changes I mentioned earlier. For example, when (and where) should testing now take place? If we have enabled the analyst to define and develop the middleware, why not incorporate the testing there too?

One MDLC which I have seen work is as shown in Figure 2.3. The final determination of phase and sequence in the MDLC clearly rests with each organization. You should, however, consider how the use of certain tools (for example, message brokers) — and the way those tools are implemented — might suggest who should do what.

In addition, success with middleware will be determined — at least in part — by the appropriate partitioning of skills. Clarity about who does what — before any project is initiated — is a necessity.

Defining the MDLC before selection

I started this analysis by setting the scene and introducing the rationale for a Middleware Development Life Cycle. I finished by emphasizing the importance of skills partitioning. Two key questions now need to be asked:

- **where do you gain the up-front knowledge about what constitutes an effective MDLC for your organization?**
- **how do you do this before you know what you have chosen or how it works?**

Most organizations seek to address these by attempting to learn during a product evaluation or, more typically, on the first middleware project

they undertake. The latter is the most dangerous of all. If the project is:

- **significant, learning 'on the job' risks its success and/or exposes the organization itself**
- **insignificant, it is quite possible that the lessons learned will also be insignificant and thus be poor indicators.**

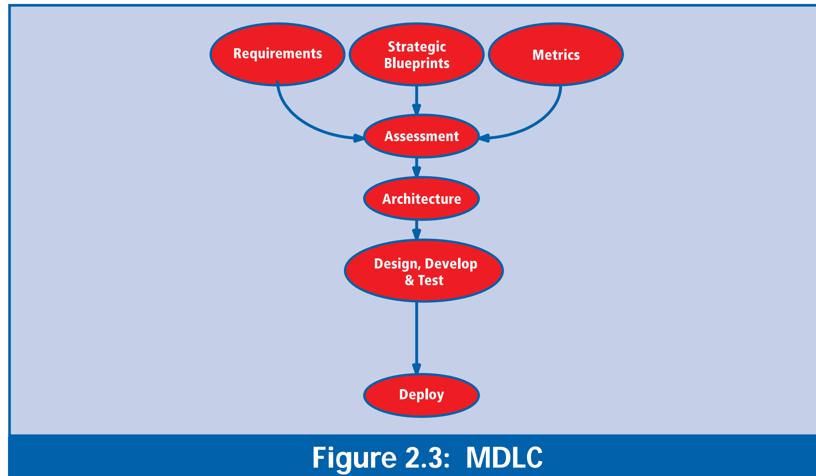


Figure 2.3: MDLC

Instead it is my experience that an effective strategic blueprint, an MDLC and an appropriate skills partitioning profile can all be created during the product evaluation phase. With these items defined and in place before your first real middleware project commences, you are likely to be better positioned for success. To achieve this, the emphasis needs to be on:

- **really knowing what you want, in detail, and then defining the criteria for knowing when you have it**
- **including typical as well as extreme scenarios: do not run the risk of oversimplifying**
- **basing evaluations on previously completed projects (where metrics are already available); rigorously review those metrics against what you achieve during your middleware evaluation phases**
- **creating an evaluation team with a membership which reflects all the groups that have a stake in the success of your ongoing (not just immediate) middleware projects; do not omit managers with relevant strategic responsibilities**
- **considering the roles and responsibilities for the evaluation team; these should reflect how your teams will be organized once the evaluation completes**
- **making sure your skills in the evaluation team are equivalent with those that will be in place after the evaluation completes; a stronger, or weaker, evaluation team can create problems**
- **ensuring the vendor's skills are on-par with those that will be in place after the evaluation completes; look for the warning signs of dependency on the vendor's R&D staff**
- **committing both you and the vendor to an initial as well as an extended evaluation: the extended evaluation validates that vendor (functions plus support, responsiveness, financial viability, etc.) as well as giving you the time to understand what you are buying — before you commit to purchasing.**

Other positive actions to undertake include:

- **avoid partnering with vendors that are not prepared to match your extended evaluation; this is an indicator of a lack of commitment or of resources or viability**
- **test, test and test again — before you make any commitments**
- **consider using contracts based on measured success for the first few projects (this also forces you to think about appropriate metrics)**
- **map out your existing architecture and the effect/impact that middleware will have on it — using a timetable that makes sense for your organizational planning horizon**
- **define where you will use middleware and, just as importantly, where you will not.**

Faster, better, cheaper middleware

Many middleware products use proprietary scripting languages or have integration 'hooks' into other products or programming languages. There are lessons to be applied here:

- **resist trading one language $ for $, especially if the language you are trading for is not widely adopted in your organization (or even your industry)**
- **measure the productivity — traditional vs. middleware; make metrics part of every project (and your life) by maintaining a library that can be used to assess the suitability of your middleware for each and every project**
- **apply assessment criteria rigidly: only use middleware where it makes sense to do so and when past metrics support the assessment.**

Creating a middleware habit

Introducing middleware is as much about managing change as it is about technology. In order to achieve success you must not only demonstrate technical suitability, but you must win the hearts and minds of those who will be using the technology.

Several considerations will need to be addressed, including:

- **train or transition: communicate with your staff to ensure they clearly understand how to realize the benefits you expect middleware to deliver**
- **be aware of market adoption trends and the impact on your people; products which are being widely adopted, especially in the finance sector, are more likely to win the hearts and minds of your staff (Java, irrespective of its technical pluses or minuses, is a clear example of the power of this effect)**
- **managers buy productivity; developers are swayed (attracted) by technology**
- **do not forget the suicide software bomber: staff can kill tools they do not like or which they perceive to be limiting.**

Essential steps for managing your middleware vendor

In a middleware product market that is still evolving, active vendor management is essential to realizing your expectations. Too many middleware vendors are still in the phase of business development which precedes widespread adoption by customers of their products.

This can be very good news, or bad news. The choice is largely up to you. Forward thinking vendors value the opportunity to speed their development and that of their products by working with constructive customers — and this can be mutually exploited. In managing your vendor:

- **identify those with staying power, funding and the revenue/potential to support growth; sometimes this will mean choosing the most aggressive — for they may have the 'will to win'**
- **do not pay for services during the pre-sales cycle**
- **pilots, evaluations and proof-of-concepts are part of the pre-sales cycle**
- **pay on success — but define what success means to you**
- **beware the beta software vendor: define your policy about using beta software up-front as well as a policy for access to new versions of software and ensure you know what you are running (beta software may give your organization a competitive edge, but it may also impose an additional management burden)**
- **do not permit development of standard product features on your dime unless you are clearly benefiting from those developments.**
- **value your intellectual assets and leverage these for maximum value from your vendor**
- **mobilize your sector to benefit you and your vendor: many industry sectors can benefit from generic features that do not expose your competitive advantage (and, overall, a stronger vendor is better for everyone)**

- **be persistent — influence early, frequently and consistently**
- **leverage your marketing value: limit the use of your name and reference in return for services and commitments**
- **test support extensively and over an extended honeymoon period to see if it will last the anniversary**
- **understand if there is a difference between what the contract says and what the reality is: do not be afraid to enforce the contract**
- **establish executive relationships between your organization and the vendor**
- **create a forum for regular communication with both executive teams**
- **watch out for vendors who 'need' high revenues from consulting; you could be buying a 'product' foundation that needs constant support**
- **introduce skill profiles (and fee rates) with your vendor from the start of the contractual relationship; that way you will less likely be caught out.**

Management conclusions

Introducing middleware for the first time looks like a challenge. It need not be if you understand that middleware requires discipline and some changes to your current orthodoxy.

If you want to be successful — and who does not — the following summarize the lessons that I have learned in deploying middleware:

- ***define a strategic blueprint during the middleware product selection stage: also, consider what makes an appropriate process and review your skills partitioning early***
- ***supplement your efforts by developing your own Middleware Development Life Cycle***
- ***use initial and extended evaluations to validate the short and medium term capability of your vendor and its middleware before moving to a real project***
- ***define success-based criteria for each evaluation phase, and contractually for the first few real projects***
- ***consider organizational issues around people and communication: change early***
- ***plan to win the hearts and minds of those using the middleware you select***
- ***be firm but fair with your middleware vendor and understand the areas where you can leverage your intellectual or brand assets in exchange for services and/or commitments***
- ***establish an approach which allows you to manage your vendor, and not vice versa.***

Processes, security and middleware: hole or whole?

Phil. Manchester
Consulting Editor
MIDDLEWARESPECTRA

Management introduction
Security used to be about locking the computer room door. Since the advent of networks — particularly those that expose corporate IT systems to the outside world — security covers a much broader range of activities. Access management, virus attacks, transaction security and data integrity all fall within the remit of the security system in the context of the modern networked environment. There is little use in locking the computer room door if security breaches can come in through a communications port.

Middleware and security procedures, therefore, are being logically drawn together. But can this logical imperative be addressed in practice?

All rights reserved; reproduction prohibited without prior written permission of the Publisher.
© 1999 Spectrum Reports Limited

The security imperative

The expansion of the scope of security is the inevitable result of the growth of the enterprise network and its extension beyond the boundaries of the corporation. Furthermore, modern enterprises are increasingly organized in ways that add further to security problems:

- **geographically dispersed sites**
- **greater use of mobile workers and telecommuting**
- **increased use of outside contractors in 'virtual organizations'**
- **'public' Internet connectivity.**

All of these factors combine to create a complex and vulnerable technology infrastructure and middleware is increasingly a key focus for security concerns in this new environment — and a potential area for conflict. While middleware allows for greater connectivity between platforms and networks, it also exposes them to potential security risks. Environments that include several platforms and applications that are distributed across networks are especially vulnerable.

The shift to open networked systems as the basis for modern commerce comes with an inevitable price tag. Businesses now find their networks are exposed to the outside world as never before. Increased use of electronic mail and the race to offer Internet connectivity are just two examples of the pressures to open up networks to a wider community — both within the enterprise and externally.

In a 1998 survey by the FBI and the Computer Security Institute, 64% of US companies reported security breaches — a rise of 16% on the previous year. Over half of those polled cited Internet connections as a point of attack — adding to internal security breaches which have, historically, been the main source of problems.

The move to electronic commerce will add to the security burden on IT systems. As businesses increasingly communicate electronically with their suppliers and their customers, they will become more vulnerable to attack. The cost reductions and marketing opportunities promised by electronic commerce are only beginning to become apparent; as they gain momentum, the pressure to open up networks and provide access to corporate data will grow.

At the same time, networks are becoming technically more complex — embracing many different operating platforms and software regimes. Security policies are much more difficult to build for an environment that spans thirty years of technological development — encompassing:

- **mainframes**
- **'Wintel' PCs**
- **UNIX-based minicomputers (and smaller)**
- **cross-platform middleware**
- **and a myriad software applications, from the smallest single user application to the largest ERP suite.**

While security is an obvious priority, it has to be balanced with accessibility. The business imperative demands that some parts of the network must be accessible in order to encourage people to use it. Internet connectivity is widely accepted as a key technology both for maintaining competitive edge and for improving IT services. Indeed, the main purpose of many of these applications is to improve that accessibility.

But opening up the network to meet these demands also exposes it — and precious corporate data — to the security threats, whether from viruses or unauthorized access by criminals or information systems terrorists.

Security is not just about denying access and protecting data, however. It is also about ensuring that important transactions are fulfilled — and, indeed, have a genuine legal basis as a contract between parties.

This aspect of security is well-established in traditional centralized mainframe systems where it is relatively easy to control. But the same criteria must be applied to distributed systems based on UNIX and Windows NT, which do not share the same depth of expertise or technical capabilities (in either people or software).

Every enterprise, therefore, must consider its security policies — and the tools it uses to enforce them. Security and middleware have to be consid-

ered together in the context of what has been deployed, and will be deployed, in the infrsatructure.

Application or infrastructure?

There are, broadly, two approaches to implementing security:

- **within the application**
- **within the infrastructure.**

Most organizations will try to apply security procedures (if they try at all) in both. Both are concerned with similar issues:

- **authentication**
- **authorization**
- **administration**
- **auditing and accountability**
- **data integrity.**

The current trend is towards centralization of security — partly as a reflection of a general trend towards central control of IT functions through systems and network management. Inevitably this means greater emphasis on overall network or infrastructure security rather than on piecemeal approaches based on individual applications.

This move towards central control and network-based security is most evident in the growing use of single sign on (SSO). IBM's Global Sign-On (GSO) and Millennium's Firststep are examples of products which offer SSO facilities.

In place of individual security systems for each application, SSO means that users can gain access to all the applications they need to use through a one-off authentication procedure. This brings several advantages:

- **greater convenience to users who do not have to remember multiple passwords and access procedures**

- **centralized administration and control over security policy**

- **reduction of cost and effort in implementation of security procedures**

- **improved security resulting from simpler procedures.**

An IDC survey of US companies last year shows that smaller organizations are leading the way with SSO — with over half of those surveyed already using it. Larger organizations are expected to catch up, however, and within two years about 70% are expected to have implemented SSO.

This will inevitably lead to greater emphasis on network-wide security procedures rather than on single application protection.

Where does middleware fit?

With greater centralization of IT systems security administration and concepts such as SSO, the role of middleware in the security regime becomes an important issue. Under SSO, for example, once users have gained access to the system they could, theoretically, get to any application within the SSO domain. Similarly, a transaction that comes into an application from an Internet connection might be secure within both domains but could, potentially, be at risk on the boundary where it is translated from one domain to another.

Where security is built into an application — ensuring end-to-end protection — middleware acts merely as a transportation channel. In other words, there is no need to consider security issues within the context of middleware. But, as is increasingly the case where security is applied universally at the network level, middleware must — at the very least — be security aware and might even need to include security features of its own to ensure complete protection.

There many good reasons to apply security at the network level rather than for each individual application. Network level security:

- **cuts costs**

- **presents a consistent security interface (SSO)**

- **reduces the complexity of a security regime — making it less vulnerable**

- **provides for centralized administration.**

But it comes at a price. The analogy of building an alarmed security fence around the perimeter of a military facility in place of individual security measures for each building in the compound is appro-

priate. The fence must be robust and it must be 'policed' to ensure it remains so.

Furthermore, it is not quite so easy to fence off a modern enterprise network — simply because it is not so easy to define its 'shape'. The resulting enterprise network is likely to comprise a mixed bag of 'systems' including:

- **mainframe-based legacy applications**
- **local departmental server systems**
- **one or more PC local area networks**
- **an open cross-platform electronic mail system**
- **Internet/Web connections**
- **external links to customers/suppliers.**

Network level security must, of necessity, embrace all of these. That is, the security fence must reach around all of them and it must be 'watched'.

This is especially true in the context of increased use of inter-object communications middleware like OMG's CORBA, Sun's Java and Microsoft's ActiveX. Dynamic monitoring is the only possible solution to the potential problems caused by these mechanisms.

Strategies and tools

Developers concerned with middleware design issues will increasingly need to be aware of enterprise security strategies — especially as the trends towards centralized administration and network-level security grow. This awareness will have to start early, at the design rather than coding stages. Otherwise the appropriate instrumentation (both management as well as security) will not be incorporated — and retro-fitting is prohibitively expensive.

Enterprise-wide IT systems management tools — such as those supplied by Tivoli and CA — are a key point of reference for developers. Many aspects of security — administration, dynamic monitoring, data integrity — are best accommodated within an enterprise management strategy.

The first line tactical tool is the use of 'firewalls' to protect parts of the network from risk. Firewalls typically sit at the main connection nodes of a network and process all traffic going through the connection — assessing its validity based on security policy rules assigned by the organization.

There are three categories of firewall — each with their own advantages and disadvantages:

- **packet filtering**
- **application gateways**
- **stateful inspection.**

Packet filtering — usually implemented in router hardware — provides a basic level of application-independent security by examining packet headers to determine their destination. While it offers users the advantages of direct connections, low overheads and little effect on network performance, packet filtering offers only limited security control and fails fully to isolate networks.

Application gateways, however, provide full network isolation, much greater control over connections and easier auditing. Application gateways set up a proxy process which handles all interaction between the application and external activities.

The major disadvantage with this approach, however, is the increased overhead in processing and the lack of adaptability. Every new application added to the network will need a proxy process to implement the security scheme. Network Associates' Gauntlet, Axent Technologies' Eagle and Secure Computing's Sidewinder are examples of application-level proxy firewall products.

Stateful inspection offers, perhaps the best option — combining the adaptable, low-level, high-performance approach of packet filtering with the greater safety associated with application gateways. Stateful inspection traps packets at the network layer and analyses them based on data gleaned from all seven layers of the standard open communications model. This enables stateful inspection approaches to apply a similar level of security to application gateways — but without the disadvantage of reduced performance or flexibility. New applications can be added without the need to build extra 'gateways'.

More important, stateful inspection allows for 'virtual' sessions and the use of connectionless protocols — making it especially suited to the complex middleware regimes used in many heterogeneous networks. Check Point's Firewall-1 is the best-known example of a stateful inspection-based firewall package.

It is also evident that firewalls and centrally-administered systems/network management regimes are drawing closer together:

- **Check Point introduced its OPSEC platform based on standard protocol support last year**
- **Tivoli and CA are also accommodating security mechanisms within their system management frameworks.**

Fitting into the infrastructure

Most larger enterprises increasingly see security as one of a range of functions and services that are necessary to sustain a network. It follows, therefore, that the best place to find a solution to security problems is within the broader scope of systems/network management.

Logically, this is the best place to deal with security in open, heterogeneous networks. It allows central control over resources and access and reduces the administrative burden. The trend in systems/network management over the last year or two has shifted:

- **away from the low-level concerns of managing the telecommunications transport system**
- **towards accommodating higher-level, system-wide issues like security.**

Modern network management tools, therefore, can help companies fulfill their security needs in the context of managing the infrastructure (from network up to application). This is far better than using the Band-Aid approach which has been the default. Yet even this can often leave one hole which is too often under-considered.

Middleware and the process-to-process trap

By its nature middleware tends to be process-to-process intensive. It operates invisibly ‘behind the scenes’ — as queues talking to queues, as messaging systems talking to message systems, as object brokers talking with object brokers. So far so good. But all of these are working in the background, often with little or no direct connection to anybody who has been specifically security authorized for what is occurring.

Herein lies the trap. A user may log onto a system and application — using any or all of the security systems above. He or she, with that application for which he or she has been authorized, may initiate a series of processes which will execute outside the ‘security purview’ of the secured application. For example, taking a sales order may initiate a message to a work/event flow manager which in turn starts a number of long running processes over accounting, manufacturing, inventory and distribution systems to manage that sales order through to completion.

The difficulty lies in how you grant the authorization to the work/event flow processes (and the messages moving between the various systems supporting the work/event flow manager). Do these ‘take on’ the security persona of the originator (which is probably what would happen within a Public Key Infrastructure), even though the originator may have no more to do with that order?

Alternatively, is ‘automated security’ created by the middleware? If the latter, this is granting systems the capability to grant security — which is not a concept that is built into much middleware (or even into middleware connected applications). Or is the presumption — as with the camp fence example given above — that once inside, then security no longer matters? But even this has problems if that originator was someone outside the enterprise — say a customer filling in the order form.

At base, the issue is that the implications of middleware — as processes running processes and data between systems — need security to be considered in ways that are very different to the norm. What is even more disappointing is that when these issues are raised with the security or middleware or systems management vendors, the most common impression given (occurring in >90% of enquiries) is that of blissful ignorance that there is even an issue — never mind that it needs a solution. This is disappointing and — in a world where use of inter-enterprise connections is becoming a business imperative — unsatisfactory.

Management conclusion

The growth of inter-enterprise communication, plus the Internet, is driving a trend towards central control which would like to introduce security procedures covering the enterprise (including middleware) as never before. The use of SSO pushes

enterprises more and more towards an enterprise-wide security strategy in place of piecemeal deployment of security based on applications.

This inevitably has implications for middleware design strategies. While they may not need to be experts in the detailed implementation of security policies, middleware developers need to be aware of the scope of security within the evolving network infrastructure. They must, therefore:

- ***understand the emerging structure of network security standards***
- ***see where middleware and security overlap***
- ***ensure that enterprise-wide security strategies embrace middleware***
- ***not ignore the process-to-process trap***
- ***monitor security strategies to assess their impact on performance of middleware.***

Managing middleware

Will. Capelli
Analyst
Giga Information Group

Management introduction

Managing middleware is not straightforward. Indeed, it is often forgotten about until it is put into production — when an uncomfortable hiatus occurs as operations tries to work out how it will know what is happening when it cannot even see what middleware is working (or not working).

In this analysis, Will Cappelli reviews the practical and operational importance of IASM (Infrastructure, Application and Service Management). He differentiates between the various approaches to IASM (application, platform and framework) before going on to explore the different approaches of two of the leading vendors and the deficiencies of SNMP in a 'managing middleware' context.

After examining what Computer Associates and IBM/Tivoli offer (as two leading vendors), he considers the impact of IASM on message oriented middleware and then on object request brokers. His conclusions do not make for comfortable reading.

All rights reserved; reproduction prohibited without prior written permission of the Publisher.
© 1999 Spectrum Reports Limited

The importance of IASM

Most large organizations implementing middleware postpone thinking about Infrastructure, Application and Service Management (IASM) issues until after their middleware configuration has been designed and even deployed. Such postponement is highly problematic, however, for two fundamental reasons:

- **such postponement (or worse) of consideration of service issues risks poor customer satisfaction**
- **IASM infrastructures necessarily are invasive.**

In the first, Giga's research shows that corporate user satisfaction with the results of IT projects is more strongly correlated with on-going levels of service (for example application performance, availability, etc.) than with the specifics of the functionality delivered. Without some kind of IASM management capability in place, the ability to monitor and control middleware service levels will be limited (and, in many cases, non-existent). Hence, postponement of systems and network management considerations risks low user satisfaction with the middleware implementation.

Second, IASM infrastructures are undoubtedly invasive. If a middleware layer is to be managed effectively:

- **persistent probes will need to be inserted in the right places**
- **hardware, software and network resources will need to be to allocated to the management application.**

Given the extent of the invasiveness, a pre-IASM middleware configuration will often require a thorough overhaul if it is to accommodate the systems managing it. In other words, if an organization wants to avoid a second round of design and deployment, the middleware layer and its associated network and systems management infrastructure must be architected and laid down at the same time.

Refining definitions and scale

Before diving into additional detail, three definitions are needed in order to make sense of the IASM context (Figure 4.1):

- **an application**
- **a platform**
- **a framework.**

An application is defined as a piece of software that monitors and/or controls the performance, configuration and access to a given computing hardware, software or network component. For example, BMC's Change Manager for MQ is a configuration management application for MQSeries.

In contrast, a platform is defined as a collection of such applications — all servicing a single component. Candle's Command Center for MQSeries is such an integrated performance and configuration management platform.

Finally, a framework integrates multiple platforms. Computer Associates' Unicenter TNG (for example) provides integrated management of MQSeries, SAP's R/3, CICS, etc.

Giga estimates that, at present, approximately 30M discrete network and computing resources — owned or rented by the Global 1000 — have been 'instrumented' and are being monitored and/or controlled through some kind of management platform or framework. Of these 30M:

- **approximately two thirds are being ultimately managed by platforms exclusively focused on network entities — entities defined on or below the TCP level in TCP/IP (or its OSI or SNA analogues)**
- **the rest are being ultimately managed by platforms capable of monitoring and controlling entities defined above the TCP level (or its analogues); in many of the latter cases, the entities defined at the network level are being directly managed by network management-specific platforms — but these platforms, in turn, pass critical information on to more inclusive platforms.**

Due to the increased concern about assuring service levels from the end user's viewpoint (and coupled with the increased volatility in the contents and structure of IT Infrastructures, Applications and Services occasioned by the implementation of Internet technologies), Giga anticipates that by 2003 over 90% of the instrumented discrete network and computing resources owned or rented

by the Global 1000 will be ultimately monitored and controlled by management frameworks (Figure 4.2). These will be capable of representing and acting upon entities defined across all levels of the network and distributed system hierarchy.

The IASM market today

Today, the IASM market embraces a broad — if loosely related — collection of applications and technologies which link such applications together. Either this is delivered through a common user interface for launch and monitoring or it is delivered through a repository capable of representing managed components to which the various applications have access.

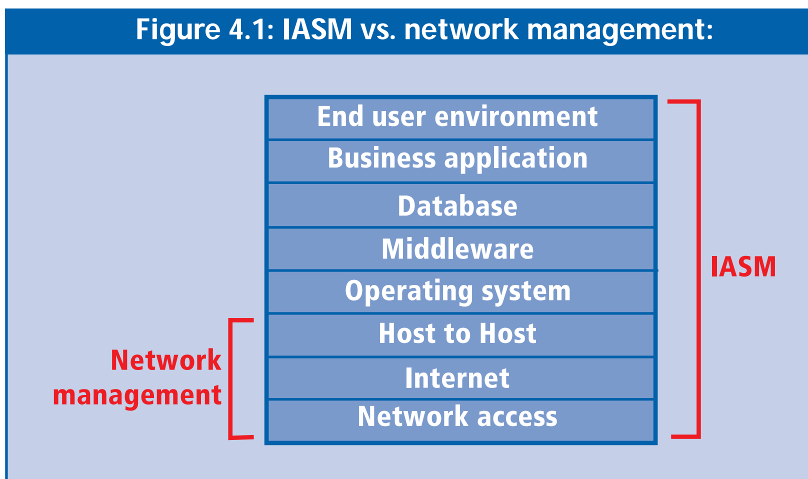
Despite vendor claims (and ambitions), platforms and tools supporting network management applications (for example HP's Network Node Manager, IBM's NetView, Cabletron's Spectrum, Sun's Domain Manager, Nortel/Bay Network's Optivity) remain sharply segregated from platforms supporting 'distributed systems management' (CA's Unicenter TNG, Boole & Babbage's CommandPost, IBM/Tivoli's TME).

The past two years, however, have seen a significant effort on the part of vendors — on both the network and distributed systems fronts — to enable large users to implement processes and technologies which can harmoniously tie together all aspects of IASM functionality. Discussions with Giga's clients — as well as a review of publicly available market data — suggest that the pure network management platform vendors are not expanding as rapidly in the distributed systems management space as are those vendors with origins in distributed systems management. The latter are expanding into the network management space due to three factors:

- **the network management platform vendors have tended to use SNMP MIB structures to represent distributed system entities; as is argued below, SNMP must be included as a part of any integrated IASM solution, but it simply does not have the semantic resources (on its own) effectively and efficiently to represent complex system objects like a database management system or an ERP package or middleware**

- **the network management platform vendors' traditional core customer base lies in network administration staff; over the past five years, network administration staffs have been increasingly integrated into the overall IT organization, have lost much of their autonomy and, consequently, when decisions and selections about management platforms have to be made, the IT decision makers tend to overrule their network administration colleagues and elect to choose vendors that have traditionally supplied them with their distributed systems management functionality**

- **in some cases, leading network management solutions have, through acquisition, become part of a distributed system management vendor's broader portfolio (for example, NetView); corporate users, once the case for an integrated IASM approach has been established, are increasingly opting for those vendors with their roots in the distributed systems management space.**



Giga research identifies four key business requirements which are relevant to making the case for integrating network and distributed systems management functionality. These can conveniently be summarized as reflecting a balance between the following:

- **cost**
- **benefit**
- **flexibility**
- **risk.**

Cost

The cost scrutiny under which corporate IT organizations are being placed is particularly targeted at operational support activities — those activities which do not directly generate perceived business value. IASM tasks lie at the heart of these intensely scrutinized activities. Consequently, any changes to technology or process which promise to yield cost savings in this space are enthusiastically welcomed.

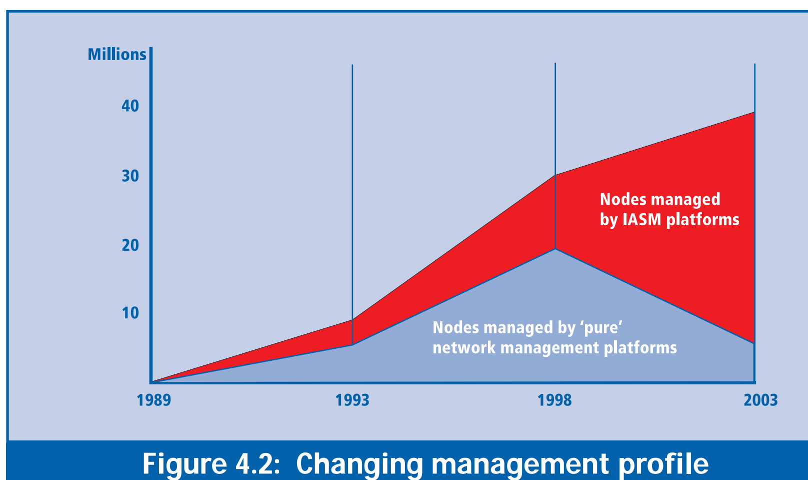


Figure 4.2: Changing management profile

Today, a technology or process IASM task portfolio promises cost reduction in at least two areas:

- **simplification and rationalization of systems administration processes**
- **simplification and shortening of the systems and network management application integration process.**

Benefit

Giga's research has shown that end user community satisfaction, and hence perceived value, with regard to corporate IT organization operational support is largely a function of three variables:

- **application availability**
- **application response time**
- **problem resolution turn-around time.**

The first two metrics, while simple to state and easy to comprehend, have proven particularly difficult to generate, monitor and control in environments where systems and network management functionality have been kept separate from each other. The availability and response time of an application — as perceived by an end user — is the result of a complex interplay of both network and distributed systems factors (including the middleware).

Systems management and network management platforms/tools are each capable of generating vast quantities of information about the specific managed objects that fall into their respective domains. Yet it is only with difficulty that an administrator can weave these different information intputs together into a seamless whole that accurately reflects the end users' perception of a situation.

With regard to the third metric, the integration of IASM functionality allows for more rapid event correlation and problem diagnosis. In many cases it makes possible a broader array of responses — many of which may be susceptible to automated actions or reactions.

Flexibility

Among the many sources of accelerating change faced by businesses, two in particular have relevance for IASM:

- **frameworks**
- **tools.**

The first, and most obvious, is the evolution of computing and networking technology itself — and the resultant need to refresh (almost continuously) the base of corporate IT assets. The second is the on-going breakdown of IT infrastructure boundaries among corporations — whether as a result of commercial alliances/mergers and acquisitions or through vertical value chain integration with customers and suppliers.

With regard to the first source of change, since many if not most IT assets impact both network and distributed systems infrastructures, the efficient addition and disposal of such assets would be greatly aided if networks and distributed systems could be managed jointly. With regard to the second, the effective coupling (or decoupling) of existing infrastructures requires the ability to view both networks and distributed systems as collections of managed objects through a single, coherent lens. Clearly middleware must be reflected in both, which is a proving to be no insubstantial challenge.

Risk

The reduction of risk is tied to the increase of knowledge. Unfortunately, in current corporate experience, knowledge about the disposition of IT assets can often be hard to come by. Repeatedly, over the past ten years, corporate decision makers have made expensive errors or have missed revenue generating opportunities because they did not know — and had no way of obtaining — accurate information about the extent and structure of their IT asset base. Such knowledge resides in the technologies and processes supporting IASM.

If these processes and technologies are not integrated, there is a significantly greater risk that critical data points will fall 'in the gaps.' More importantly, the knowledge that would arise from seeing how networks and distributed systems interact with one another can be lost. The role, and contribution, of middleware encompasses one of those risk categories which could all too easily 'fall between the cracks'.

Although the business case for integration of network and distributed systems management seems clear, there are two fundamental barriers to its achievement:

- **the existence of a de facto standard — SNMP — which is by itself unable to serve as a focal point of integration (but which must also be taken into account in any attempt to provide an integrated approach to IASM)**

- **the fact that any technology providing an integrated representation of all the various components of an IT infrastructure (plus application portfolio plus service portfolio) will itself consume significant quantities of CPU resource and network bandwidth in its own right.**

SNMP is a constraint

IP has established itself as the dominant corporate network protocol. In its wake SNMP v1 has been accepted as the fundamental protocol for network management. This means that SNMP is unlikely to be uprooted. It must, and therefore has to, be dealt with at some level in any attempt to provide an integrated approach to IASM.

Two factors have, however, made this situation highly problematic:

- **first, despite IP's rapid growth, there remain — and these show little sign of disappearing in the forseeable future — large, business critical corporate network segments which are dominated by other protocols (for example, SNA and IPX)**

- **second, and even more importantly, the fundamental architecture of SNMP renders it inadequate as a means of representing and manipulating complex systems objects (the semantic poverty of the MIB grammar and the lack of provision for security are only two of the bigger constraints).**

The simultaneous ubiquity and inadequacy of SNMP has led to radically different solutions on the part of the major systems and network management platform vendors:

- **IBM/Tivoli treats network management, and the underlying network models (including SNMP) upon which it is based, as an application which functions in ways that are largely independent of the distributed system model accessible through the Tivoli Enterprise Console and Desk Top**

- **Computer Associates in Unicenter TNG has created an inclusive object model to which SNMP provides crucial, but not exclusive, data feeds.**

The IBM/Tivoli solution, while currently allowing an administrator to access relevant segments of the network map from objects presented in the distributed systems object model, fundamentally requires networks and distributed systems to be treated as separate self-contained structures. It is largely left to the administrator to correlate (manually) the distributed system object events and processes with those related events and processes affecting the network infrastructure.

This 'liberation' of the distributed systems model from the network model makes the task of representing end user views of IT functionality comparatively easy. Unfortunately — given the increasing role that network availability, latency, etc. are playing in the end user's perception of a given application's performance — the sheer independence of the network model limits TME's overall effectiveness.

In contrast, the Computer Associates solution, of all those currently available on the market, comes closest to offering a truly integrated view of network and distributed systems architectures. Unicenter TNG begins with SNMP as its fundamental data source for configuration information. The object model, however, is only loosely tied to the SNMP topology. Not only can it take input data from other network/network management protocols: virtually any feasible data source can make its contribution to the resulting integrated model.

Dealing with resource consumption, however, there is a brute force quality to both CA's and IBM/Tivoli's enablement of their integrated views of networks and distributed systems. Both vendors pay for their integration with excessive resource consumption. Nevertheless, both CA and IBM/Tivoli have sought to address this through a variety of strategies — by exploiting, in different ways, the underlying manager/agent architecture contained within each framework.

In a manager/agent setting, infrastructure components, applications and interfaces to services are 'instrumented' (linked locally) to a two-part software entity, the agent:

- **the first part stores data about the component, application or service interface**

- **the second part contains processes which can read the data, act directly on the component, application or interface — in response either to the data or an external message — and then transmit messages (in response either to the data or the external message).**

Agents are typically linked across a network to one or more managers. A manager is also a two-part software entity, usually residing on a workstation or server:

- **the first part stores all of the data, at the least in summary form, contained within the various agents to which it is linked**

- **the second part contains processes capable of presenting the data (in an organized manner) to a human administrator and is able to launch a variety of applications which can act upon managed objects through the offices of the agents attached to those objects.**

Both CA and IBM/Tivoli have greatly enhanced the data filtering and local response capabilities of their agents. This allows agents to take a wide array of actions on managed objects without having to consult the manager tier of the architecture. In addition they have both made it possible to build hierarchies of agents and managers, ensuring that even in cases where a 'higher tier consultation' is required, the trail does not necessarily have to lead all the way back to some centralized point.

In general, these two enhancement strategies have been successful. Three years ago, these products reached their limits (in terms of performance and the usefulness of the information generated) at 10,000 nodes or managed objects. Now, they are each capable of comfortably managing 100,000 node topologies.

The Internet and footprint impact

Despite their combined success, however, these strategies have come at a price. That price is growing as the Internet further increases network size and complexity. To date, most businesses have introduced only a single tier between the top-tier managers and the bottom-tier agents.

The introduction of an extra tier has greatly reduced historical levels of network management-related traffic (particularly in the SNMP sphere). But further reductions in network management traffic (with a significant impact on the traffic associated with the management of components above the network level) requires the introduction of more tiers. Unfortunately, because of the difficulties involved in designing and monitoring four or more tiers of a manager/agent hierarchy, this step has not been taken. Consequently, the burden of further reducing network management traffic (as well as reducing higher-level component-related traffic) has fallen on the first strategy — adding new function and filtering capabilities to the lowest level agents.

As a result the footprints of agents, as well as the overall footprints of IASM systems, are growing to the point where — in some environments — an IASM enabled-system requires resources (measured in terms of numbers of processors and storage capacity) amounting to 35% of the resources being managed. 20% is closer to what might reasonably be expected. Furthermore, while there are some economies of scale to be obtained on the manager side of the architecture, IASM-attribut-

able resource consumption tends to grow linearly with the number of objects or nodes being managed.

Computer Associates' attempt to deal with the problem of 'overweight agents' has been largely confined to:

- **incremental improvements in code efficiency**
- **increased attention to efficient Unicenter TNG implementation methodologies.**

In contrast, IBM/Tivoli, in the upcoming Tsunami release, intends to take a more drastic step. Recognizing that many individual agents share a number of common functions, a new layer of 'agent services' will be interposed between agents and managers which will allow, according to beta site users, an order of magnitude in size reduction in the agent footprints.

Unlike the peer-to-peer style relationship between agent and mid-level manager, the relationship between agents and the 'agent services' layer will be strictly client/server, largely erasing the complexity of the design issue. Furthermore, while mid-level managers serve fundamentally as filtering and switching stations for agent-generated traffic, the 'agent services' layer will be functionally rich and hence of much greater use in rationalizing traffic patterns attributable to system components above the network level.

Despite these virtues, however, equipping TME with an 'agent services' layer will have its negative consequences. Since most third party (and many IBM) management applications effectively function as agents in the TME framework, any fundamental revision to the agent architecture will complicate and lengthen the task of building interfaces to non-Tivoli applications and frameworks.

In addition, Internet-driven extranets and electronic commerce are increasingly forcing companies to share their respective IASM architectures, albeit transiently. Consequently, a premium will be placed on those frameworks which, whatever their resource demands, ease rather than complicate the integration process.

On this point, CA's incremental code improvement approach — which leaves the basic agent structure intact — will give Unicenter TNG and its descendants a decided advantage. (This is not, of course, to suggest that Unicenter TNG agents will wholly avoid periods of temporary bloat as new functionality is added.)

Managing MOM implications

Given the market's move towards framework-based integrated approaches to IASM, where does this leave the management of middleware? For the sake of specificity, focus on the management of:

- **message oriented middleware (MOM)**
- **object request brokers (ORBs).**

Although queue-driven message oriented middleware — particularly IBM's MQSeries — is proving popular for process integration and legacy system connection, tuning the implementations of this technology for acceptable levels of performance and availability is proving difficult. While both IBM and BEA provide rudimentary management functions 'inside' MQSeries and BEA MessageQ, these are pretty primitive compared to what is needed for IASM. For MQSeries, at least, there are tools on the market supporting a number of higher level systems management disciplines (performance management, configuration management and security) from the likes of Candle, BMC/Boole & Babbage and Nastel. More recently, support has been added by leading framework vendors like IBM/Tivoli and Computer Associates.

Unfortunately, while these tools are (with varying degrees of sophistication) capable of pinpointing local bottlenecks (for example an overused queue), the biggest factor affecting overall performance is the underlying design of the MOM topology. In other words, unless the basic configuration of the queues, channels and queue managers is laid out effectively, the chances of radically improving overall performance are low, no matter how much 'after the fact' tinkering is done.

What are needed are methodologies and tools to support the 'analysis and design' phases of MOM implementation. These are, as yet, sorely lacking.

Given the complexity involved in structuring an optimal MOM topology — it is, in fact, a so-called NP-hard problem, which means that one has about as much chance of hitting upon a truly optimal MOM design as one has of discovering the private key of an RSA encryption — methodologies in the

short term will consist of 'rules of thumb' gained only through extensive experience. These are not likely to be available on the market for at least two more years. Until then, significant reductions in MOM-induced latency will be more a matter of luck than application of IASM-based disciplines.

Managing ORB implications

Even more so than is the case with MOM, ORBs have been deployed with a minimum of attention to their manageability. Instead the emphasis has been almost entirely upon their ability to wrap rich layers of abstraction around legacy systems while (hopefully) laying a groundwork for an infrastructure of re-usable components.

This emphasis on functionality (over manageability) is typical of the first implementations of a new infrastructure technology. But failing to take manageability into account at an early stage all too often leads to acutely grave problems with regard to ORB performance and service availability soon after initial deployment.

The reason is that the performance and availability characteristics of an ORB configuration do not appear to be that amenable to incremental tuning. In other words, if it is not done right the first time, it is difficult to 'make it right' through post-implementation modification. (The reality is that re-building is usually the only practical option.)

In addition, anecdotal evidence suggests that the key factor impacting ORB performance and service availability is the physical proximity of an object to the application invoking its methods. Arguably, the whole point of an ORB is to mask the underlying physical topology (of the infrastructure within which its services are located) — from both the user and the application developer. Indeed, one can further argue that requiring the application developer to code with an eye to the physical topology in mind undermines most of the justification for deploying ORBs in the first place.

Nevertheless, it remains true of ORB implementations that application developers need to minimize the physical distance between an object and the various points where its methods will be invoked — if performance is to be optimized. But this only makes the inclusion of ORBs in IASM even more problematical.

Management conclusion

Too often the management of middleware is considered 'after the fact'. Often it is not possible to retrofit IASM capabilities, unless the middleware infrastructure is rebuilt. This is not a conclusion that those beginning to deploy middleware want to hear.

But, as Mr. Capelli illustrates all too well and as most large scale users of middleware already know, middleware which is not being managed is a liability. It is not sufficient to develop in the hope that an operational dimension can be added later.

With middleware this is simply not acceptable. Instead, solutions which need to use middleware must have suitable instrumentation built-in from the start. Above all else this applies to any 'middleware infrastructure'.

Emerging and converging middleware: application servers and message brokers

Dr Keith Jones
Senior Software Consultant
IBM Corporation

Management introduction

Some 20 years ago client/server emerged as the architecture to accommodate deployment of personal and departmental processing. It fueled a debate over how best to exploit low-cost MIPS for business value. Corporations experimented with configurations they thought would meet their needs. Over the years, however, clients grew fat and servers proliferated.

In the 'e' world it is middleware, when combined with the Internet, which is allowing adopters to focus more on applications for competitive advantage and less on infrastructure. Two of the more relevant developments in the middleware arena are:

- ***application servers***
- ***message brokers.***

Somewhat confusingly, both cover similar ground. For example, both promise options to connect new applications to old ones; both claim to improve 'enterprise response times' in a fast changing world. Keith Jones uses this analysis to explore the two different forms before commenting on what might be different about each — and some possible choices.

All rights reserved; reproduction prohibited without prior written permission of the Publisher.
© 1999 Spectrum Reports Limited

The rationale for application servers

As recently as three years ago the Internet emerged as a technology with more applicability than merely the academic or research aspirations of its previous 25 years. But, in the short period since that arrival, it has both unified and simplified the client domain as well as brought structure to the server domain. At the same time middleware has expanded to include every type of enabling technology which links client with server, server with client and server with server.

The explosive growth in the numbers of end users connected by the Internet has forced many corporations to explore ways of exploiting the Internet as an additional medium for growth. Visionaries have proposed a whole new commercial world order in which end user clients — around the globe — will dynamically connect via the Internet to services of every conceivable shape and form for just enough time to inquire and transact before moving on.

Business has responded by adapting its processes to the World Wide Web, initially by providing web-enabled access to information about products and services. This was followed by 'applications' of increasing sophistication (and complexity) as the capability to undertake transactions (purchases) was added. So aggressive has this evolution been that the time taken by corporations to make this transition is sometimes used as one measure of an organization's ability to survive.

The path chosen by most businesses was first to introduce Web servers (Figure 5.1) to handle electronic client access in parallel with more traditional call-center customer service systems. Web servers were designed to:

- **accept large numbers of concurrent client requests, using HTTP-flows over IP connections**
- **respond with static pages of useful information coded using HTML and delivered using the same HTTP-flows over IP.**

At first, page content was statically coded and presented in response to each client's request. However, the inefficiency of static pages was quickly realized and replaced by a more dynamic creation of information — usually extracted from purpose-built databases which were interrogated behind the Web server (using CGI scripts or similar).

The third step taken has been to present dynamically created forms explicitly designed to accept end user requests for products and services along with credit card and other payment details. Whilst concerns for privacy and security caused some enterprises to adopt this slowly, by late 1998 there was explosive growth in the revenues collected via Internet connections. The technology needed to achieve this evolutionary step involved introducing application logic behind the order forms which had been dynamically created and presented to end users.

A fourth step is now being taken. This standardizes on 'thin-client' Web browsers for new applications which access applications and resources on Intranets (intra-company networks) and Extranets (closed groups of connected users from multiple organizations). The rationale for this has been the desire for a reduction in the total cost of ownership for end user systems either in-house or in extended virtual corporations that share business processes.

Application servers are proving essential for providing such access to information as well as the business logic needed to implement the associated business processes. Often these are re-engineered from legacy or existing applications specifically for the purpose.

Three reasons for considering application servers

From an architectural point of view, the positioning of business logic behind HTML pages delivered by Web servers (where it, the information or process, can be shared by large numbers of users) makes great sense. However, early implementations did not always benefit from separating application business logic from the presentation logic needed to construct the HTML pages.

The concept of an application server has brought this value sharply into focus (Figure 5.2). But this has not been the sole driving force behind the introduction of so many application server products in recent months. There have been at least two other parallel justifications; the need to:

- **link end users, via the Web, to legacy applications**
- **provide application reliability and sustainability.**

For many businesses the introduction of Web servers to enable Internet access to their products and services extended the use of existing legacy applications. Although these applications were primarily developed for internal use, adaptation and re-use of existing core business functions (and databases) proved to be the only realistic option if Internet access was to provide immediate business value. Attempting to re-engineer solely for the Web was realized to be both too expensive and too time consuming.

Unfortunately, the functions needed by Internet users often turned out to be 'composed' of accesses to two or more different corporate systems of record. This meant transaction processing and similar considerations came to the fore if such Web-fronted applications were to satisfy users. The application server concept soon evolved to satisfy this need — by providing 'connectors' to legacy transaction applications to support the new business logic behind Web server pages (Figure 5.2).

The third motivation behind application servers came from the realization by leading e-business enterprises that opening for business on the Internet invites enormous — and frequently unpredictable — workloads. In many scenarios Internet access from around the world imposes the requirements for 24 by 7 by 52 availability — in addition to coping with unpredictability and providing reliability. As some would have it, 'your competitor is just 2 clicks away'; in many cases these operational requirements for the Web are more stringent than those needed for in-house only applications.

Application servers today provide the linkage to legacy applications as well as separation of business and presentation logic. They are now emerging with the appropriate reliability and availability capabilities that can, if needed, mask older internal applications failures and service interruptions. As such, application servers have added:

- **life to the inventory of existing application assets, by enabling these to be connected to a 'third party' (the application server itself)**
- **reliability and sustainability to combinations of existing applications which previously, in concert, did not possess these capabilities**
- **separation of the business and presentation logic**
- **new ways to create new applications (whether mixed with the old or wholly new).**

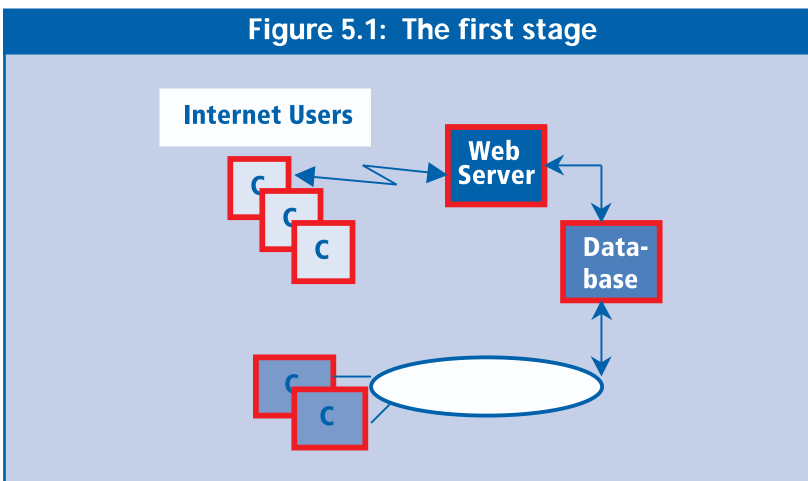
How to recognize an application server

The difficulty that many IT managers are facing is that of choosing the most appropriate application server solution from the plethora of recently announced products. Most managements tend to focus on the basic need — to provide a robust execution environment for applications implemented in a number of different forms:

- **procedural COBOL or C programs**
- **objects written in C++ or Java**
- **more recently Java Beans and/or Enterprise Java Beans (EJB).**

Such execution environments will most often provide separate threads for each application service request being handled — in order to provide the throughput and scalability required as well as the failure isolation needed for reliability. For larger scale implementations, some application servers even provide clustering services to provide for dynamic cloning of server execution environments at times of high workload — with distribution and balancing of end users' requests across active execution environments in the cluster. Such application servers come with:

Figure 5.1: The first stage



- **sophisticated administration, configuration, tuning and debugging capabilities**

- **the capability to handle large numbers of concurrent end user 'sessions' (management of session state information including security and possible affinities between requests and particular databases or legacy systems).**

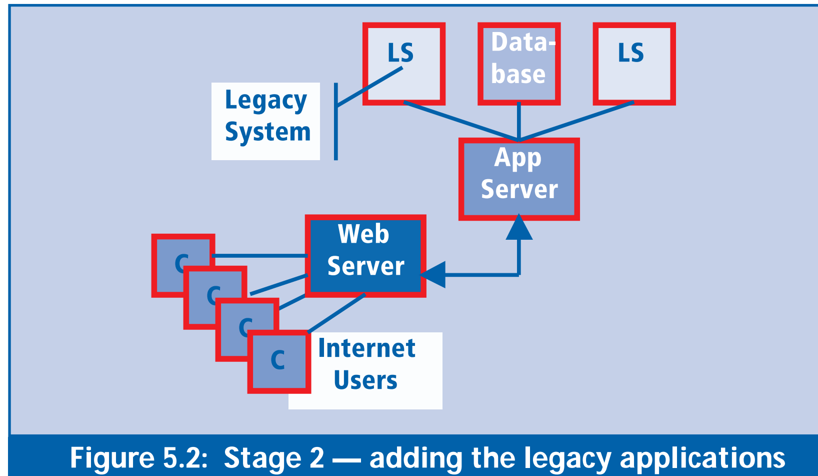


Figure 5.2: Stage 2 — adding the legacy applications

In addition to these capabilities, most application server solutions will also include:

- **an integrated development environment for applications written in a language/ mode of choice**

- **a set of utility programs for administration, monitoring, tuning and maintenance of the active environment.**

Applications server choices

Given the richness of functionality needed to achieve a robust highly-scalable application server solution it will not be a surprise to learn that many of those already announced come from vendors with considerable investment in established core technologies (Figure 5.3). For example:

- **Netscape's Application Server 2.0 and IBM's WebSphere Application Server 2.0 (Standard and Advanced) are solutions closely affiliated with the corresponding Netscape and Apache Web Servers**

- **the Inprise Application Server and Sapphire 5.1 Application Server are closely affiliated with the corresponding Integrated Development Environments from Inprise (originally Borland) and Bluestone.**

Similarly, Oracle's Application Server 4.0 and Sybase's Jaguar CTS are closely aligned with the corresponding Oracle and Sybase database products. BEA's (originally Weblogic's) Tengah 3.1 Server and IBM's Websphere Application Server (Enterprise Edition) exploit the corresponding Tuxedo and TXSeries transaction monitor products. And there are many more application servers emerging.

Indeed, this segment of the middleware market is expected to expand rapidly before it begins to consolidate. It has the distinct attraction that it is associated with application development, especially the association with speed and fast results. This makes application servers particularly appealing to the AD community.

Messaging and message brokers

Messaging has emerged in the past 3-5 years as the paradigm that seems best to support distribution of procedural function and data across heterogeneous systems. As corporations have accumulated application investments running on different platforms (either through acquisitions/mergers and/or purchases of packaged solutions), so they have equally acquired the need to connect the resultant 'islands of computing'.

Increasingly, messaging is now the mechanism of choice for delivering this. The growth in messaging systems deployment has also been aided by the need for greater processing parallelism and, in some scenarios, replication of databases.

By focussing on messages (which can also be described as 'application defined data structures') and their distribution through queues to application processes (in an Intranet or Extranet network), businesses can connect their disparate systems. In the past this had always proved too difficult. But the introduction of simple proprietary messaging APIs delivered a breakthrough.

Today bus, hub-and-spoke and snowflake network configurations are used for delivery of messages in high volumes between applications in industries as varied as securities trading, banking, manufacturing and health care systems. Transactional messaging — with both synchronous and asynchronous modes — has earned messaging systems a respect that is similar to transaction monitors and RDBMSs in the pantheon of mission critical middleware.

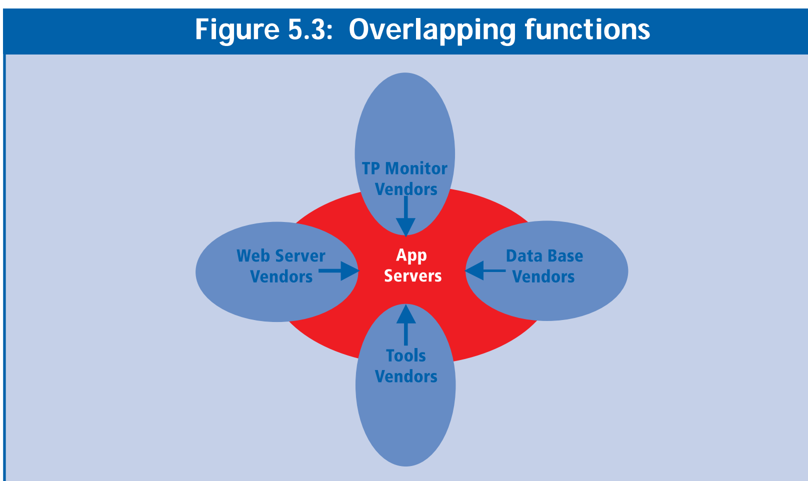
Starting with messaging: moving onto message brokers

The first step taken by many organizations was to introduce singular connections between individual applications. This provided a point-to-point mechanism for the exchange of data between designated platforms. Message queue managers were introduced to ensure the flow of messages, and their recoverability in the case of failure.

The difficulty with this approach was that messages were designed to encapsulate the data exchange needed on an application-by-application basis. The point-to-point connections were all too often 'one-offs', with existing application programs being adapted to 'Get' and 'Put' messages as required.

The second step taken by many was to establish a messaging 'backbone' between several corporate application systems (Figure 5.4). The realization that messaging can provide a reliable mechanism for connecting systems — and for enhancing availability — then motivated some corporations to exploit the messaging paradigm for new applications, not only for connecting and re-using existing legacy systems.

Figure 5.3: Overlapping functions



This, coupled with message broadcasting, publish and subscribe and fault tolerant message protection promoted new types of application which are now well established. But, as the number of platforms needing connection increased, so the number of potential connections increased exponentially n(n-1) = the number of possible connections (where n is the number of systems to be connected). Beyond 10 or so systems, the complexity of the infrastructure became too great to manage at a reasonable cost. A different approach was required.

By 1998 a growing number of vendors had focused on extending the business value in messaging systems by adding what are called 'message broker' functions. These presented new options for the third evolutionary step in messaging.

Message brokers have become 'switches' or intelligent agents, configured between messaging applications, which can recognize, transform, route and re-route messages according to business rules or policies that are defined outside the original applications. With such intelligent, non-invasive agents it became feasible to build automated business processes (so-called event or work -flow applications) which:

- **exhibit scalability, reliability and availability**
- **reduce the n(n-1) complexity to a manageable size.**

How to recognize a message broker?

The message broker solutions now available broadly assume some form of base message queuing and transport services with complementary utility functions for administration, configuration tuning and problem determination. The additional functions focus on:

- **data/message construction**
- **formatting/transformation**
- **message flow management automation.**

The first reduces the amount of work needed to exploit data/messages going to/coming from legacy

applications. The second reduces the resources needed to handle messages in a complex business process and improve quality, while the third opens a door to work or process or event -flow management (or any mix of these).

Today, most message broker implementations include comprehensive data conversion, message formatting and transformation services. Their objective is to avoid having to 'invade' the source of the message-originating applications (as new uses for the data from these appear). The flow of data from one platform may need to be converted before it can be utilized on another platform, just as the data elements may need to be reorganized and reformatted before they can be consumed by any destination application.

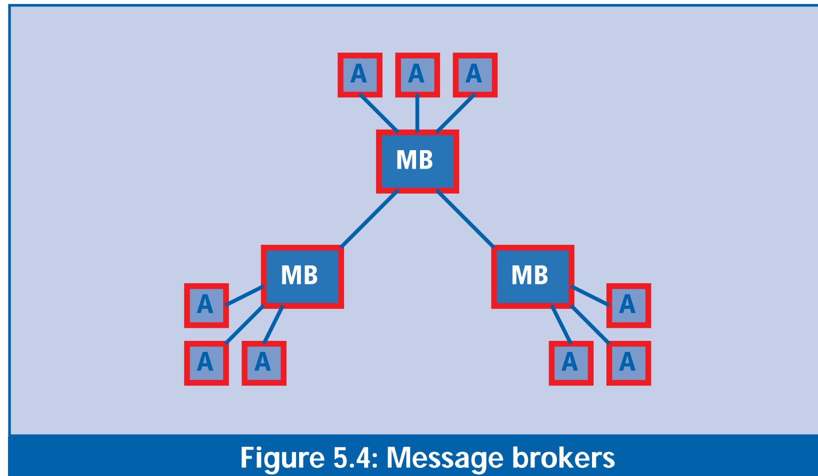


Figure 5.4: Message brokers

Message brokers perform this function by recognizing certain message formats and creating new ones according to rules. Support for XML as a tagging language for message content by some message brokers shows another way forward.

Once message formats have been catalogued and standardized using message broker functions, it becomes possible to focus on message flows for business value. Message brokers provide a range of functionality for management of message flows including:

- **routing to applications based on message content**
- **collection and accumulation of messages from disparate sources**
- **replication and broadcasting messages to disparate target applications**
- **application of business rules based on message content**
- **creation of 'message warehouses' (for audit and other analyses).**

These facilities together make it possible to apply business logic to data flows within a messaging system. By linking complex processes — where previously human intervention was mandated to 'integrate' different applications on different systems — work (and event and process) automated flows can be introduced.

Message broker choices

Vendors offering message brokers include:

- **IBM/New Era of Networks (MQSeries Integrator)**
- **PRL (I/O Exchange)**
- **Glotech Solutions (MBS)**
- **Candle (Roma)**
- **HIE (Cloverleaf and OM3)**
- **MINT (Mint Core)**
- **Sopra (A&P)**
- **TIBCO (Active Enterprise).**

Each solution is effectively a collection of message based products which together deliver some combination of the function described earlier.

Differences and similarities between application servers and message brokers

Enterprise level application server implementations include a wide range of adapters or connectors to:

- **databases — such as Oracle, DB2, Sybase, SQL Server and DL/I**
- **corporate transactional application execution environments — such as IMS, CICS, SAP's R/2 and R/3, MQSeries, CORBA ORBs and communications servers.**

Caching of connections to these environments — so that they can be shared for efficiency by large numbers of concurrent application threads — is

also a prime requirement. Indeed, the caching and the degree of tight coupling in these connectors will probably prove, over time, to be critical factors in the overall performance of Internet connected applications.

Confusingly, message broker solutions also include a wide range of adapters or connectors to:

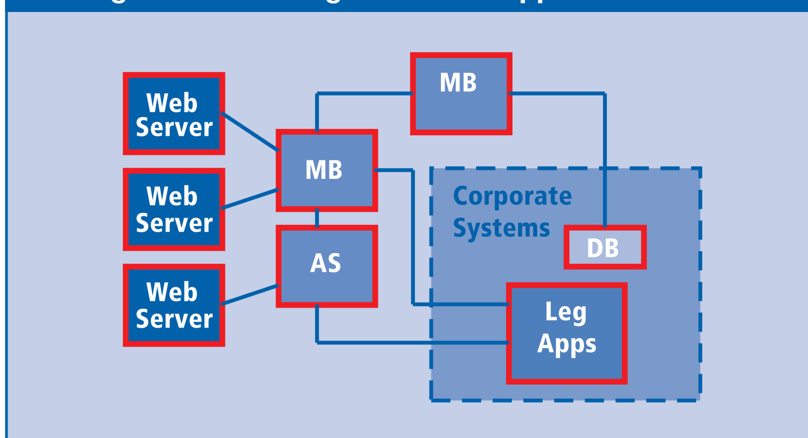
- **databases — such as Oracle, DB2, Sybase, SQL Server and DL/I**
- **corporate transactional application execution environments — such as IMS, CICS, SAP's R/2 and R/3, MQSeries, CORBA ORBs and communications servers.**

Similarly, caching of connections, intelligent handling of trigger and alert messages, transactional synchronization and exploitation of parallel processes are some of the more sophisticated facilities to be found in fuller function message broker solutions. While not identical to those provided in application servers, it is arguable that the only aspect of real difference is in the starting or purchase point.

Both application servers and message broker solutions offer new options for integrating disparate applications via a middle tier (the applications server or message broker). As such, choosing between the two might be argued to be about where you start:

- **application servers are front-end, AD-focused**
- **message brokers are back-end, operations and process oriented.**

Figure 5.5: Message brokers + application servers



This is not to say that application servers and message brokers cannot be combined (although this may well prove expensive). Applications which service Internet requests behind Web servers may well use messaging for communication with corporate systems of record (Figure 5.5).

Another way to view these two technology families is to consider them as being complementary and converging. In such scenario, message brokers will continue to expand their functionality to include rules engines, transactions and both synchronous and asynchronous modes of operation. As Internet access to enterprise business processes increases (and becomes more sophisticated), it becomes both feasible and attractive to automate the handling of requests from end users.

The effect of message brokers can be to improve the handling time and the quality of response to such requests. Indeed, the likelihood is that message brokers will handle requests from Web servers and provide intelligent transformation and routing services — to application servers.

Rather than using messaging to interface to existing systems, application servers will accomplish this, with publish and subscribe messaging handleing the reverse flow of information from corporate systems to end users, where strict sequencing of request responses is not needed. Or you can view this the other way round — with legacy and new applications being created via application servers but using message brokers for the interconnections.

Management conclusion

The differences between application servers and message brokers — while sustainable for the moment — will be short-lived. The differences are more of style than substance: both are trying to achieve much the same, albeit by different means and from different starting points.

As it is the ends which justify the means, choices are going to depend on perception as much as anything — although this should not distract from the fact that availability, reliability and scalability in the systems deployed to handle Internet access are critical operational considera-

tions if an enterprise is to remain competitive (especially on a 24 x 7 x 52 basis). As systems configurations grow with workload, it will become increasingly important to manage complexity. Adding application servers and message brokers to already complex configurations will increase the number of moving parts — not reduce that number.

Indeed, one other thought may confirm the overlaps/differences. Would it make that much difference to talk about 'application brokers' or 'message servers'?

Is inter-operability between object types feasible?

Rosemary Rock-Evans
Consultant

Management introduction

If object types cannot inter-operate, developers may find they have developed hundreds if not thousands of objects which will not work with any middleware product on the market — with thousands of $s/man hours of effort disappearing down the tubes. As Rosemary Rock-Evans argues, inter-operability is the key issue.

Middleware vendors often talk about inter-operability and inter-object communication as if both were a current reality and feasible. In fact, there seems to be a widely held belief by multiple organizations that all objects are much the same; therefore inter-communication between them is easy.

This is an intensely dangerous misconception and one which could have serious consequences in any organization. If object types cannot inter-communicate, then a company is restricted to the tools and middleware supporting only that type of object — in effect to use only the tools and middleware supporting Java Beans (for example) or just CORBA objects or just DCOM components. In effect the developer has locked his or her organization into a highly specific technology.

Given the rapidly fading state of the CORBA product market — with DAIS from ICL being withdrawn from the market (although PeerLogic has rescued it in principle), ObjectBroker (now from BEA) being put on ice, NEO (from Sun) being withdrawn, VisiBroker (from InPrise) disappearing into Delphi, IBM moving Component Broker to Enterprise Java Beans, ... and so on — management has cause to be nervous. This is true of CORBA in particular.

All rights reserved; reproduction prohibited without prior written permission of the Publisher.
© 1999 Spectrum Reports Limited

What is an object type?

Let me first define an object type. Objects or components at their very simplest are blocks of code in any language accessed by a defined interface or interfaces (in the case of Microsoft's components). The interface names the object itself and defines the methods which can be invoked along with the parameters of those methods.

The interface is key to what defines an object type. Interfaces are defined using an Interface Definition Language (IDL). This is usually a programming language independent way to define the object, its methods and parameters, but a language nevertheless with its own syntax and semantics.

An object type is defined by which Interface Definition Language is used to describe the object. In effect, an object type is described by one Interface Definition Language with its unique syntax and conventions. We will see why this is important in a paragraph or two.

The main object types of which I am aware —- in other words, the different Interface Definition Languages which exist — are as follows:

- **CORBA IDL — for CORBA objects**
- **DCE — for XIDL objects**
- **Microsoft IDL — for ActiveX components**
- **Sun's Java Beans IDL — for Java Bean components.**

There are (or were) other, more obscure variants on the same broad theme. But this only demonstrates that not only are there many types of object, but that object types can go out of favor with their vendors (leaving the customer with code which is no longer supported). Examples include:

- **Netscape's LiveConnect objects — which is still supported (just)**
- **OpenDoc — from IBM/Apple (which has largely disappeared)**
- **SOM Objects — which were different from CORBA objects and have now been withdrawn**
- **individual tool vendor's objects — every tool has its own definition of an object**
- **proprietary ORB objects — each of which is defined using different interface definition languages (OIL, CDL and so on).**

The result is that there are multiple object types on the market today. In order to maximize investment in building any object — or ensure that that investment is protected from future change — developers must be able to make these object types communicate with one another.

Inter-object communication

To 'plug in' an object you need middleware to support the inter-object communication. The key to inter-communication is the interface the object presents to the middleware. But ORB middleware products are geared towards the acceptance of only one type of interface.

The reasons why ORB middleware will only accept one type of interface — and hence only one type of object — are tied up in the way IDLs are used by products. Generally speaking, the IDL is — in the static case — generated into programming language 'stubs' (C++, Smalltalk, C, OOCOBOL, Java, etc.) which can be used from within programs to invoke the objects (and perform functions like data format conversion and marshalling).

The interface is, therefore, vital to the generation of the code which handles some of the middleware communication functions. The stubs/skeletons then become part of the client or server executable and communicate with the middleware runtime libraries to achieve the marshalling/unmarshalling and so on. A middleware product is inseparable from the type of interface it will accept — because its processing is dependent on it.

The way the interface is defined (Figure 6.1) is critical to inter-operability between objects:

- **if the interface and IDL are the same, and the stub code generated from the IDL is the same, then all objects (whether they come from Microsoft, CORBA, Netscape, Lotus, Sun, IBM or any other company) will be able to communicate with other objects using any ORB**

- **if the interfaces differ, the customer will only be able to use the middleware which supports that interface and will only be able to use objects with that type of interface.**

As we have seen, interfaces differ. In effect, without some clever extra software, COM components

cannot communicate with CORBA objects using either DCOM or a CORBA ORB. Java Beans cannot communicate using either CORBA or the Java RMI (Remote Method Invocation) with CORBA objects ... and so on. As it stands, if you develop an object of one type you will be unable to use it with middleware supporting another type of object.

Surmounting the interface problem

One of the ways in which ORB vendors have attempted to surmount this problem is by using special translation runtime software. CORBA ORB vendors have been particularly active in this area, as they recognize the reality that a developer is likely to use Visual Basic or Visual C++ on the desktop (for example) which will create a COM object that expects to communicate using DCOM.

The bridging software they have had to develop enables a client using Microsoft's COM object model to invoke remote CORBA objects as though they were COM objects. All remote invocation, however, is underpinned by the CORBA ORB. Much of the bridging software that is being used works using dynamic invocation rather than static invocation. This needs an explanation.

Both Microsoft (with COM) and the Object Management Group (OMG) with CORBA support dynamic invocation. If interfaces are used dynamically, the Interface Definition Language is not used to create stubs (as we saw earlier in the static approach). Instead, the IDL is run through the IDL compiler and a file is created which holds a definition of the interface which is looked up at runtime:

- **CORBA uses a file called the Interface Definition File**
- **Microsoft uses a machine readable form of the IDL (which can be used by tools and other services at runtime) known as the Type Library.**

The Type Library first has to be specifically loaded by the client. Once loaded the developer obtains the information needed about the interface from the Type Library and then uses the IDispatch interface to obtain information about the location of the functions in it. The general term used for this approach is termed OLE Automation.

When a COM client is to communicate with a CORBA object, the bridging software translates between the different formats at runtime. The COM client — using OLE Automation and built using tools supporting OLE Automation such as Visual Basic — can use the OLE Automation IDL and interface to communicate with CORBA objects on the server which appear to the client as though they are COM objects. The bridging software translates the OLE Automation calls to CORBA object calls, giving:

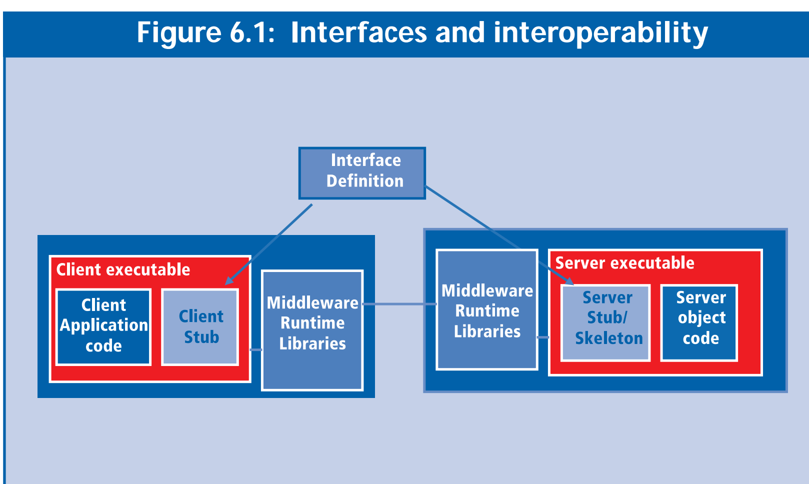
- **the CORBA objects the impression they have been invoked by a CORBA object**
- **the OLE Automation objects the impression they are invoking OLE Automation objects.**

In this configuration, an Automation Controller in the client actually works though proxy objects in the bridging software which have been generated during development. The proxy acts as the local representative for the remote CORBA object.

The proxy object (Figure 6.2) provides the same interface as the target object but in normal OLE Automation syntax. The proxy object then communicates via the bridging software and from there to the remote CORBA object. Replies are then channeled back via the same route. During development, the server IDL is usually passed through an OLE Wizard and the wizard automatically generates code for the OLE Automation controls which are to act as proxies.

On the face of it this presents a reasonably good solution to the problems of inter-operability — if you

Figure 6.1: Interfaces and interoperability



can ignore the fact that there is a real performance penalty in adopting this route. Dynamic invocation is slower than static invocation, plus you have the extra overhead of run-time translation. It could also present administrators with problems in configuring all the clients with the necessary modules.

However, in principle, and at this very basic level, this solution will work. Indeed, exactly the same approach could be used to enable COM components to communicate with Java Beans or to enable Java Beans to communicate with CORBA objects.

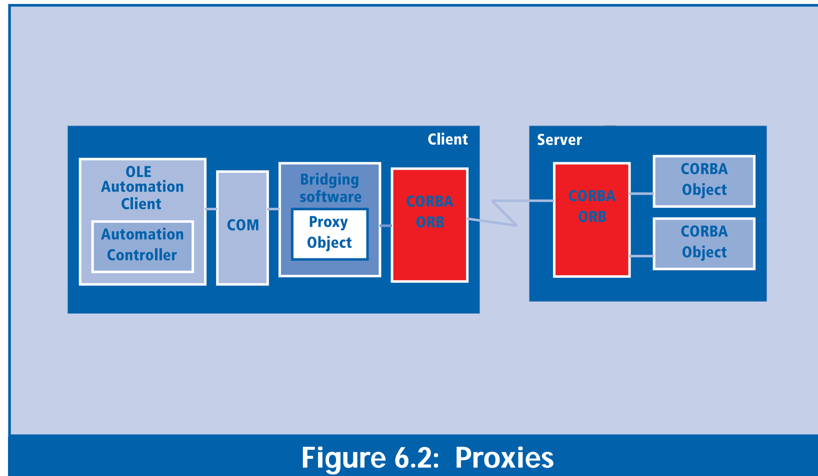


Figure 6.2: Proxies

The extra problem — service invocation

If we were to take inter-operability to mean only inter-communication between object types, we may believe that we have solved the problem and that we can use any object type happy in the knowledge that there will always be bridging software to help us maximize our investment in the development of objects. But, in order to inter-operate, an object must also communicate with the middleware upon which it relies.

Middleware provides the developer with sets of services — everything from:

- **security services**
- **automatic multi-tasking**
- **memory management**
- **triggering**
- **fault handling**
- **automatic restart**
- **queuing**
- **deferred delivery**
- **guaranteed delivery**
- **etc.**

Altogether I estimate that there are over 400 possible services which could be provided by middleware products.

The good ...

On top of such a wealth of choice, the way middleware products provide service provision varies greatly. There are some products where the number of commands provided to the developer is tiny and the service provision is almost entirely automatic. You find this in the best products — Tuxedo, MQSeries, BEAmessageQ and so on. In effect, all the hundreds of services are never invoked, they just happen under the covers as part of the automated provision of the distributed environment.

From a developer's point of view this is the ideal situation to have, as it simplifies and also speeds development. The fewer commands there are to learn, the fewer mistakes will be made and the less the product will be 'bent to fit' (doing what it was not intended to do). It also makes the middleware vendors' job easier, as updates to the middleware itself are unlikely to result in a change to the developer's API. The vendor can upgrade the services, happily knowing it will not cause any upgrade problems for customers.

The ugly ...

On the other hand, there are some products available which have as many as 800 service call APIs. Many CORBA products fall into this category. More worryingly, the Java APIs are starting to multiply (breed?) at an alarming rate. Even raw DCOM (without the tools) is already in the hundreds.

Oddly, it seems to be the ORB based products — CORBA ORBs, DCOM and the basic Java architecture — which:

- **are the most awkward**

- **require the developer to go through complex sequences of invocations in order to do even the most simple of tasks — often ones that should be automated anyway.**

All of the ORBs use the same approach to service invocation. The developer uses the same IDL to invoke services as he does application objects. Thus a developer uses the CORBA IDL to invoke the security service, the transaction service, triggering services and so on. To enable object types to inter-communicate, therefore, the service invocations also have to be translated.

Fundamental problems

There are two fundamental problems which then occur because of this. The first is that every IDL-based service call in the application object has to be translated at runtime to a different IDL using the same approach we saw above — dynamically using Type Libraries/Interface Definition Files and proxy objects for every service used. This has its own implications:

- **potentially the client could be enormous, with proxy objects needed for every service invocation**
- **every service call will result in considerable performance degradation, as the Type Library has to be looked up and translation performed**
- **the client has to have a file of all the 800 or so service interfaces stored on its hard disk**
- **services must be defined so that they are capable of being dynamically invoked (not all of them are).**

The second is that service calls in one ORB may have no direct equivalent in another ORB. It is worth pointing out that this is not just a problem of CORBA object to COM, or Java Bean to CORBA object, communication. It is worse than this. Services in CORBA ORBs may have been implemented differently with the result that a CORBA object written for one CORBA ORB may well not work with another CORBA ORB.

This latter problem presents an almost insurmountable problem. With 800 plus interfaces in the CORBA specification, a number of non standard interfaces in the products, a separate and different set of interfaces in DCOM and a gradually increasing number of service APIs in Java, it is difficult to see how any vendor could produce a truly bridging product which actually mapped one set of commands to another set. Suffice it to say that no one has succeeded in doing it yet.

Some examples

In order to illustrate the problem, I will describe how each ORB vendor tackles various services, concentrating for simplicity's sake on the gulf between CORBA and COM. Note that the same gulf exists between CORBA and basic Java, but I have restricted these descriptions to keep it simple.

The examples selected for discussion include:

- **memory management**
- **triggering**
- **persistent storage**
- **fault handling**
- **initialization code.**

Memory management

A developer using DCOM (but not Visual Basic, which does ease matters) has to handle all memory management himself. But the DCOM library provides a number of APIs which are used to obtain and free memory, as well as an interface through which methods can be called. The IMAlloc Interface has methods to allocate, reallocate, free, obtain the size and minimize the heap.

Most CORBA ORBs have no in-built memory management facilities. It is possible to 'implement' memory leaks (the allocation of memory on the heap and the subsequent inability to deallocate the memory because of the loss of all pointers to it). No interfaces are provided to handle this in CORBA. The programmer must use the facilities of the development language itself.

Triggering

Whenever a client requests a server, ORB middleware may check to see if the server is running. If the server is not running (that is, it exists in the library but not as a running task), an instance of the server is created. This process is known as triggering.

In DCOM, the client is passed back a pointer to the IUnknown interface. This interface uses two methods which must be called by every client as they use the object. The client must call the methods each time it uses a new interface pointer on the object.

When the client calls the object it calls the AddRef method. This method increments a reference counter maintained by DCOM which keeps a count of how many clients are using the object. When the client stops using the object, it must call the Release method. The Release method decrements the same counter. Once this counter has gone to zero, the object is unloaded from memory.

In contrast, with CORBA ORBs (once triggered), the server program must keep a reference of the number of clients it is serving using a method on the generic class 'Object'. This method increments and decrements the reference count. When the reference count reaches zero the server is shut down automatically.

The rules used to shut down the server differ depending on the activation mode. The client must specify the activation mode in the invocation:

- **shared mode — once all the requests for all the objects in the server have been completed**
- **unshared mode — once all the requests for that object in that server instance have been completed**
- **per method basis — once all the requests for that method of that object in that server instance have been completed.**

Persistent storage

DCOM has an in-built persistent storage mechanism and clients use interfaces (and calls on methods in those interfaces) to tell DCOM what they wish to do with storage:

- **IPersistStorage — this interface enables the component to read and write its persistent state to a storage object**
- **IPersistStream — this interface enables the component to read and write its persistent state to a stream object**
- **IPersistFile — this interface enables a component to read and write its persistent state to a file on the underlying system directly.**

The CORBA standard also makes provision for a persistent storage facility. But in many product cases this facility has not been implemented. In any case, the interfaces used to invoke persistent storage are totally different from the ones used in DCOM.

Fault handling

A developer using CORBA must add code to the bind to ensure more specific error messages are received if the bind operation fails. The code generated by the IDL compiler by default returns a NULL value if the bind operation fails. Environment code has to be added to find out why. The client invokes methods by calling the stub generated from the class.

In DCOM, the COM interface functions and COM Library API use specific conventions for error conditions which the client must pass back with the reply. Where the reply is used, the return value, status and error information (for example a Boolean result of true or false) and a failure/success field must be specified. The key field used in DCOM error reporting is the HRESULT field — a 32 bit field which is used by DCOM to report communication errors, RPC errors and so on. It is used (rather than the parameter return values in the interfaces) and is structured as follows:

- **severity code: 0 = success, 1 = error**
- **reserved (for later DCOM use)**
- **facility: a code classifying the error — some facility codes are Microsoft defined but they can be user defined as well**
- **code: the actual error code.**

It is worth noting that not only is the whole approach to fault handling different in DCOM, but the error codes used bear no similarity to those used in CORBA. An error reported through CORBA could not be handled by a COM object expecting to be working with DCOM, or vice versa.

Adding initialization code

In CORBA, both the client and server may include code to initialize the ORB and register the code. Depending on the CORBA ORB (here again there are differences in the code used between different CORBA implementations), the developer may specify within the initialization code the size of:

- **the send buffer to be used by the network transport mechanism**

- **the receive and intermediate buffers to be used (if the intermediate buffer is not specified, the ORB will maintain a pointer to the argument and will not make an intermediate copy; if used incorrectly, this can seriously affect performance)**

- **any shared memory buffer the client or server wants to use (where they are on the same platform).**

The contrast with DCOM is total. No initialization code may be needed.

What does all this mean?

My general conclusion from looking at this problem of service invocation is that it is not practical to convert the service APIs of one ORB to another ORB. But this means that object type inter-operability is essentially infeasible using ORBs themselves as the middleware.

This is a negative conclusion to draw — and it has serious implications. In essence, if a company wants to use object types, it has to pick the model it perceives to be the long term winner in the object type race. This may be the COM model or the Java model (particularly the Enterprise Java Beans model) — but it is unlikely to be the CORBA one (especially given that IBM is now backing the Java one).

Whether a developer organization goes the Java route or the COM route it is, nevertheless, locked into that model of processing. If it goes the COM route it is tied to using COM objects with Microsoft supplied DCOM (plus all the other DCOM ports supplied by Software AG, Digital, HP, Silicon Graphics, etc.). It is also very much tied to using Windows NT as the strategic platform. On the other hand, it will be able to:

- **use multiple languages (C, C++, Java, even COBOL)**

- **access applications and data in and on other non-Windows platforms (although the essence of the access is inter-operability with the existing, not true support for new, developments on non-Windows platforms).**

Conversely, if the Java route is chosen, your organization is locked into the Java language as the primary development language with all the implications of this — the most obvious ones being:

- **the performance limitations of Java**

- **the divergence of Java Virtual Machines (‘write once, test everywhere’)**

- **the difficulty of accessing legacy (working) applications in other languages.**

Are there any other solutions to inter-operability?

Let me go back to the main reasons for wanting inter-operability between objects of different types:

- **in the first place we wanted object types to inter-operate in order that we can maximise the investment we have made building objects, reuse objects and be able to pull in third party objects where needed to build an application**

- **second, however (and perhaps more important), we wanted to know that an object we build will work with multiple middleware products — that if we develop an object type we are not locked in to one technology but can move to another middleware product (if the vendor goes bust, pulls out or — worst of all — changes stance about what object type will be supported).**

Clearly, one solution is simply not to bother using objects. By using conventional middleware — such as CICS, Tuxedo, MQSeries, BEAmessageQ, TIB, TXSeries, etc. — you can develop applications which (with some work, admittedly) can be rewritten to work with another product. In effect, because all of these products only use a fairly small number of service calls, if you want to change the application to use another middleware product, the pain of changing the internal calls is not that great. Work will be needed — but not as much as with an ORB.

If you really want to use objects, however, there are alternative routes now being offered by companies such as:

- **IBM with Component Broker**
- **BEA with Tuxedo (and its acquisition of Top End).**

To look at the latter, both Tuxedo and Top End support a rich set of services geared towards mission critical high availability transaction processing, but where the services are automated. None of the richness is exposed to the developer in the form of an API. Services are either totally automatic or controlled by administrator set parameters.

When it bought Digital's ObjectBroker, BEA dismantled ObjectBroker and extracted just the bare bones needed to support inter-object communication. It then removed all the object oriented calls needed to access the services in a conventional ORB. Instead, the 'calls' used are confined to those needed to invoke application objects. Service invocation is largely automatic.

The resulting product (originally code named Iceberg) is now called M3 and represents a layer of functionality that sits over the Tuxedo product, enabling a developer to use objects underpinned by the Tuxedo middleware. Support for three types of object is planned:

- **CORBA objects (where support is offered now)**

- **Java Bean objects**

- **ActiveX/COM objects (aided by BEA's recent purchase of WebLogic).**

This support can be provided because no service calls are issued. There is no need to translate service call invocations because the object has no need to make service calls — Tuxedo's underlying services are automatic. In effect, translation between interface types is feasible because we have reduced the problem back to the level we saw earlier in the basic inter-operability description.

What does such a solution give you?

You obtain inter-operability between objects of different types — so any objects you develop using Java or COM based tools will eventually work with M3. But you still have not reached the Holy Grail — the ability to use the resulting object with another middleware product. You are locked into M3.

Furthermore, if you take an object type that has been developed for CORBA or COM and use it with M3, you will probably have to strip away many of the unnecessary invocations. Similarly if you decide to use a Java object developed for use with M3, with the Java Enterprise Beans environment, you will have to change code in the move.

At the end of the day, your objects still cannot be re-used as they stand with different middleware products. And this does not only apply to M3. IBM's Component Broker, albeit wrapped into WebSphere, will suffer from similar difficulties.

Management conclusion

The only practical route currently available to an organization wanting inter-operability between objects of different types — Java Beans, COM objects and CORBA objects — is via products such as M3 from BEA or Component Broker/WebSphere from IBM. Even if you follow this route, your lock-in to a specific middleware approach/vendor is inevitable.

Given that every route is effectively a lock-in (to the middleware vendor), potential customers must choose whether they want to use objects exclusively or want a mix of paradigms:

- ***if the desire is for objects exclusively, then you must decide whether you will be happy to use a middleware product that only supports objects of one type***

- ***if the answer is that this is acceptable, adopt the Java or COM route***

- ***if not, go the M3-type route.***

If the choice is 'not to use objects exclusively', the M3-type route will still be an option, because Tuxedo enables a customer to use both normal programming paradigms and the object one with the same service support and infrastructure. Similarly, IBM's WebSphere (with TXSeries) also offers both object, Enterprise Java Beans and conventional programming support.

If the decision is that objects are 'not for us', then an alternative middleware product should be sought. The key here is to select a product with rich service support but a minimum number of developer APIs. This will then minimize the degree of inevitable lock-in.

Enabling EAI with MQIntegrator

Jay Lang
Chief Technologist
Distributed Computing Professionals

Management introduction

In virtually every technical publication or periodical in print today, you cannot help but read about 'Enterprise Application Integration' or EAI. Whether companies are currently involved in projects (like introducing ERP or other application suites) or are planning new projects, EAI is happening in every IT department. With companies continuing to merge — or acquire — other companies, this need for EAI will persist.

Asynchronous messaging is increasingly becoming the solution of choice to help companies ensure success with their EAI solutions. Whether a company wishes to expand its business channels with electronic commerce, merge with an acquired company's systems or try to gain better inter-system co-operation within its existing enterprise, messaging fills the need.

In this analysis, Jay Lang explores one of the options for introducing EAI — MQIntegrator — from a function and implementation perspective. As a practising consultant and implementor, he chose MQIntegrator as an example of a message broker because of its exploitation of MQSeries as the middleware base which enables enterprise applications to communicate across different operating systems running on different platforms using different network protocols. But, if it is MQSeries that provides the 'A' in EAI (by giving applications the ability to communicate across those disparate platforms), it is the use of the message broker's formatting and rules which delivers the 'I'.

All rights reserved; reproduction prohibited without prior written permission of the Publisher.
© 1999 Spectrum Reports Limited

Message brokers

Message brokers are applications that receive messages, determine where to route the message and, in some cases, reformat the message based on the requirements of the receiving service. Some message brokers can also provide facilities for encryption of message data along with security services to deny or authorize access to different services throughout an enterprise.

As was reviewed in the November 1998 **MIDDLEWARE**SPECTRA (page 2), there are already multiple commercial message brokers available on the market today. In spite of the range of choices out there, some companies have opted to develop their own 'home grown' message broker applications to meet the specific needs of their organization — as was discussed by David Callingham (also November 1998 **MIDDLEWARE**SPECTRA, page 10). Their reasoning is that they can ensure that they will obtain exactly what they require: and if you have the skills, this is undoubtedly practical (as Mr. Callingham demonstrates).

On the other hand, why build when purchasing is possible, especially when MQSeries is already broadly accepted as a messaging standard? MQIntegrator is a logical extension.

What is MQIntegrator ?

MQIntegrator was a product developed by New Era Of Networks Inc. (NEON) out of Englewood, Colorado. Its main function is to act as a 'message broker application' sitting above MQSeries (and it will be renamed MQSeries Integrator when IBM announced its 'blue' version in early 1999, which will also be sold by NEON).

This message broker offers organizations the ability to send all interface messages between systems to a single point (a 'Hub'). From this point, each message will be routed to a specific location within an organization based on the content of the message data. In addition to these features, MQIntegrator also offers the capability to re-format the message data into the specific format required by the receiving server application.

MQIntegrator comprises several components:

- **a message formatting engine**
- **a business rules engine**
- **an MQSeries server**
- **Application Programming Interfaces.**

The message formats and the business rules are stored in a relational database (DB2, SQL Server, Oracle or Sybase — depending on operating system). This data (the rules and formats) can be unloaded and then reloaded onto the platform (or platforms) where the MQIntegrator rules engine is in operation: the database and MQSeries Integrator do not have to run on the same system, only be able to talk to each other.

The message formatting engine

Message formatting consists of two separate parts — 'Incoming' and 'Outgoing' formatting. Incoming messages need to be 'parsed' or broken down into fields that the MQIntegrator rules engine and formatter can use to make decisions. These input formats are used to identify types of message as well as individual data elements in the message.

This is probably the most time consuming element of introducing MQIntegrator into an enterprise application integration initiative. To facilitate this function, MQIntegrator provides a graphical user interface to assist messaging administrators establish the input/output formats to be used by different applications. But before an organization can truly utilize the power of MQIntegrator, or any message broker for that matter, it is crucial to be able to break down all messages (being passed between server applications) into data elements.

Each message type has an input control that is used to identify its individual fields. These input controls describe how to identify a field within a message. They specify:

- **the data type**
- **length of the data**
- **any literals used**
- **whether or not there are tags that precede or follow the data, or the number of times a field might repeat itself within the content of a message.**

Output formats work in much the same way as input formats. Each output format contains the fields that will be placed in the resulting output message.

One major difference between input and output formats is the ability to modify data. Output formats have operations that can be defined and performed on individual fields. These operations can be math expressions, data substitutions, field justifications, character case changes or virtually any data transformation you want. Output operations can also be used to pad and trim data from fields, and also perform sub-string operations.

For data substitution, MQSeries Integrator provides the ability to create tables of data that can be used for lookup and textual replacement. For example, if a message contains state or country information, it can be transformed to or from its full name or abbreviation using these tables from Colorado to CO (or CO to Colorado).

For more complex transactions, output operations can be grouped into collections. Collections are used to perform multiple transformations to an output data field. For example, if you want to change a string of character data to UPPER CASE and also end the string with a delimiter that your EAI has chosen as a standard, you would group these two separate operations into a collection, and then associate this collection to the output data field.

Programming exits are also allowed as an output operation. These exits offer developers the capability to initiate an application program — passing to it the data field associated with the operation. Exits can be used to access data tables already existing in an enterprise's applications in order to make further determination of what needs to be done with the message data.

The business rules engine

Business rules are the foundation of any message broker's function. Rules provide the means by which a broker can interrogate a message for specific data or directives that can indicate to the message broker where this message needs to be sent within EAI.

In MQIntegrator these rules are defined using a graphical user interface application. Using this interface, application or messaging administrators define the rules that will be used to send messages to the correct application server. Since the message data has already been parsed into separate fields using the formatter, these same data fields are used in the logic used to route these messages.

Also provided in the rule creation graphical user interface is language syntax for logic statements to be used on the data fields. Once the logic statements have been created, and the rule completed, you can subscribe different application messages to these rules. Along with the subscription is the associated action that is requested. This action would instruct the rules engine to issue the MQPUT call, placing the message on the queue associated with the server application appropriate for this rule.

MQSeries

As previously mentioned, MQSeries is messaging middleware which enables applications running on the same or different platforms to communicate with each other in a synchronous or asynchronous fashion. Over 25 platforms, from MVS on the mainframe to NT on the desktop, are supported as MQSeries servers (plus numerous lighter weight clients). Regardless of the platform, applications communicate with the messaging system using a standard API that contains four key MQSeries verbs — OPEN, CLOSE, PUT and GET.

MQSeries uses a queue manager plus different kinds of queues, communications channels and triggers to accomplish its mission of assured message delivery. MQSeries stores messages in a transmission queue until a communications channel is active and can transmit the messages to a local queue managed by a different queue manager.

The queue manager is responsible for managing all objects (queues, channels, listeners, receivers, etc.) in its domain. Applications use those objects (and a queue manager is itself such an object). For example, applications can issue message 'gets' (MQGET), 'puts' (MQPUT) and 'inquiries' (MQINQ) against queues.

In the scenario of using MQIntegrator, inbound messages are put on queues targeted to the inbound queue on the hub where the MQSeries server is running (and on top of which run the formatting and rules engines). In practice, MQIntegrator will use two of four separate sets of queues during the course of processing a message through the hub. These are:

- **an input queue: this queue is used to receive all of the incoming messages from source applications (and this is the queue that triggers the formatting and/or rules engine into action)**

- **a no hit queue: this queue is used by the rules engine to place messages that ended up having no rules applied to the message (all messages must have at least one rule apply in order for them to be routed)**

- **a failure queue: this queue is used by the rules engine to place messages that experience errors during the formatting and/or rules stages of processing**

- **outbound queues: these are where messages are placed once processing has been completed by the formatting and rules engines; messages placed on outbound queues are then delivered to the destination systems/applications matched to the outbound queue(s).**

Application Programming Interfaces

Along with the GUI interface to define and manage message formats and routing rules, MQIntegrator provides programming APIs which enable access to all of the functions provided by the GUI and the rules engine. These APIs can enhance the messaging environment. Message administrators can:

- **write applications that access the input and output formats directly**

- **apply additional data manipulation if the need exists**

- **dynamically create input and output formats, along with routing rules.**

By using these APIs, companies can write their own message broker applications, extending the features currently provided by MQIntegrator. This allows for specific formatting and rule applications that an enterprise might need, while still taking advantage of the standard functions found in the basic product.

Developers of products try to cover all functions that they believe will be needed by the users of their products. IBM and NEON with MQSeries Integrator are no exception. The formatting/rules engines provided by MQIntegrator should cover most of the needs of most organizations.

However, in instances where your organization has specific requirements for message validation, parsing or routing, the MQIntegrator APIs are there to help. By allowing access to the low level functions in the product, IBM/NEON have given Message Queuing Administrators the ability to add their own message broker functions and exits — to be used when the packaged functions do not suffice.

Why use a message broker?

As you start to lay out an enterprise application integration 'network' (and all the related queue managers, servers and associated objects to support the integration) you quickly realize that you are going to need a great many connections to obtain the necessary connectivity. This is the 'spaghetti nightmare', especially if it has to be implemented on a point-to-point basis.

It is now that the appeal of a message (possibly better called an application) broker becomes clear. It can save you a great deal of effort as you move towards EAI.

In the example of an MQSeries-only middleware set up, each server platform in your enterprise will have a MQSeries queue manager (or, possibly, a set of client libraries) which will provide the access by applications to the messaging (and queuing) function. As your MQSeries-only environment grows along with your business, so do the number of servers (and associated objects) which need linking if integration is to occur.

Manually configuring these, and the need for application developers to have intimate knowledge of your configuration (queue names, server locations, service locations, etc.) swiftly produces an unmanageable morass. There is simply too much detail which must be tracked and available as new servers are added.

For example, each remote queue providing access to a service on another platform will need a transmission queue as well as channel pairs for storage and transport. Any change in the location of any part of this service — whether it is due to capacity upgrades or server relocation or renaming of any object — will require changes which can ripple throughout the enterprise.

In a practical situation, assume that you have a system which involves twenty different queue managers. In order for all of your queue managers to have access to each other, you will need a minimum of 190 pairs of sender/receiver channels, that is 380 channel definitions (the formula here is

n (n-1) = c — where n is equal to the total number of queue managers and c is equal to the number of channels needed).

Put simply, the number of MQSeries objects spirals upwards all too readily. Even if your enterprise has purchased a tool to help with the configuration of your MQSeries environment, the challenge of managing all of these channel connections can be overwhelming.

This is where the primary benefit of a message broker comes into play. In the example above, all 20 server platforms would be connected to an MQIntegrator hub. All messages will now go via the message broker server (hub) platform. This means that (for example) the number of channels needed to connect these 20 platforms is reduced to one pair per server (to and from the hub) — or 20 pairs or 40 channels. If a new queue manager is needed, only one additional pair of sender/receiver channels will need to be established (in this instance, adding a new queue manager server would involve defining 2(nw -1) channel definitions, where nw is the new number of queue managers).

This is just one of the significant benefits of using MQIntegrator as a message broker. It helps you contain and manage your MQSeries messaging environment.

But there are many additional benefits, once this basic broker environment has been established. These are the ones which apply most directly to EAI — and include the following:

- **non-invasive connection to existing applications and/or application suites**
- **assured, asynchronous processing between servers**
- **formatting and reformatting between unlike systems (including merging or event explosion)**
- **long running transactions, spanning both time and systems**
- **publish and subscribe (one application may generate one ‘event’ which can be distributed to multiple other applications without the publishing applications needing to know who or what is subscribing)**
- **intelligent routing**
- **introduction of aspects of automated work or event flow management**
- **adding new logic outside, and/or between, applications.**

All of these simplify EAI. They deliver the tools for accomplishing what has, in the past, been both difficult and complex. In business terms, these are the core justifications for message brokers. But, in IT terms, the simplification of the infrastructure is what counts.

Management conclusion

Message brokers are coming into their own in EAI. By using MQIntegrator as an example, I hope that you have gained a better understanding of both this product and how Enterprise Application Integration can be delivered with one particular message broker. Indeed I will go further. I believe that it is imperative that organizations starting an EAI endeavor, or finding themselves in an ever expanding application enterprise, will benefit greatly by looking at the power of message brokers supported by asynchronous messaging and the message driven model (if you disagree, you can always argue with me via jay@tpmq-experts.com).

MQSeries has long proven itself as a leader in asynchronous mission critical enterprise messaging. With the additional benefits provided by MQIntegrator (message formatting to message routing and data validation as well as simplification of the middleware infrastructure), organizations can deliver applications to support their business activities faster — all the while utilizing existing systems and services currently in place. As your applications grow in numbers, so will your messaging environment. Message brokers are there to make EAI work.

Financial MIDDLEWARESPECTRA
Enhanced Intranet Subscription

UPGRADE TO THE ENHANCED INTRANET SUBSCRIPTION

The Enhanced Intranet Subscription provides you with multiple copies of each Report, for one year, plus additional middleware resources to put on your Intranet. It includes the following:

- ✓ 5 printed copies of each quarterly Report, published in February, May, August and November (these copies can be mailed to separate addresses)
- ✓ 52 pages of analyses and case studies in each Report, including diagrams
- ✓ a quarterly CD with a 12 month licence to publish the information on it internally on your Intranet containing:
  - the Reports of the Calendar year to date
  - the Reports of the previous Calendar year
  - the latest copy of MIDDLEWARESPECTRA's Vendor Database (>200 vendors listed)
  - the latest versions of the 10+ volumes of the *Issues in Middleware Collections* (collectively priced at US$3500 if all bought separately from an Enhanced Internet Subscription)
    - Middleware In Action Collections (4 Volumes: US$47.50 each; US$100 for all four)
    - Middleware Architecture Collection (426 pages, $795)
    - Strategic Issues in Middleware Collection (531 pages, $895)
    - Distributed Systems and Middleware Collection (637 pages, $795)
    - Middleware in Distributed On-Line Transaction Processing Collection (437 pages, $995)
    - Middleware in Data Warehouse, Database and Database Access Collection (170 pages, $375)
    - Messaging and RPC Collection (361 pages, $475)
    - Queuing Middleware Collection (319 pages, $495)
    - OO and Middleware Issues Collection (311 pages, $395)
    - Middleware and Application Development Collection (405 pages, $475)
    - NEW Enterprise Application Integration and Middleware Collection (91 pages, $395)
    - NEW Internet and Middleware Issues Collection (228 pages, $495)
    - NEW Middleware and Message Broker Collection (96 pages, $395)

**To order your Upgrade
to the Financial MIDDLEWARESPECTRA's
Enhanced Intranet Subscription
call:**

**(in the USA/Canada) 1-800-933-5997
(Europe/Rest of the World) +44 1962 878333**

Members of the International Advisory Board

Charles C.C. Brett
President, C3B Consulting Limited & President, Spectrum Reports

William Donner
Chief Architect, Reuters

Kathryn Dzubeck
Executive Vice President, Communications Network Architects, Inc.

Ellen M. Hancock
President
Exodus Communications, Inc.

Paul Hessinger
Vision UnlimITed

Pierre Hessler
Deputy General Manager
Cap Gemini

H. William Howard
Vice President, Inland Steel Industries, Inc.

Michael Killen
President, Killen & Associates, Inc.

Dale Kutnick
President, Meta Group, Inc.

Norris van den Berg
General Partner, JMI Equity Fund, LP

Fiona A. Winn
Managing Editor & Publisher
Spectrum Reports

Philip Manchester
Consulting Editor

Additional contributors include:

Francis X. Dzubeck
Communications Network Architects, Inc.

James V. Franch
System Software Associates

Keith Jones
IBM

David McGoveran
Alternative Technologies

John Tibbetts & Barbara Bernstein
Kinexis

Amy Wohl
Wohl Associates

Martin Healey
Technology Concepts Limited

Allan Lees
Softwire

Aurel Kleinerman
MITEM

Les Yeamans
North American Systems Group

Ian Hugo
Year 2000 Taskforce

David Sutherland & June Hacker
Carleton University

Rosemary Rock-Evans
Consultant

Jonathan van den Berg
Premier Software Technologies

Tom Heywood
University of Southampton

Eric Leach
ELM

Glen Macko & John Parodi
Digital Equipment Corporation

Randy Rhodes & Troy Terrell
Black & Veatch

John Carter
IBM UK Laboratories

Roy Schulte
Gartner Group

Jim Johnson
Standish Group

Tom Curran
TC Management

Alfred Spector
IBM Corporation

Max Dolgicer
International Systems Group, Inc.

David Baer
Consultant

Jerrold M. Grochow
American Management Systems, Inc.

Ken Orr
The Ken Orr Institute

Peter Houston
Microsoft Corporation

Jeff Tash
Database Decisions

Ed Cobb
BEA Systems

Bernard Abramson
Merck & Co.

Mirion Bearman and Kerry Raymond
CRC for Distributed Systems Technology

Oliver Sims
Integrated Object Systems

Jim Gray
Microsoft Research

Pierre Jouanny
Thomson Software Products

Wayne Duquaine
Grandview DB/DC Systems

Steve Craggs
Candle Corporation

Colin White
DataBase Associates International

Gustavo Alonso
Swiss Federal Inst. of Technology

Peter Mark
Lotus Corporation

MIDDLEWARESPECTRA is published and distributed worldwide by:

USA and Canada:
Spectrum Reports, Inc.

Subscription Center
PO Box 301368,
Escondido, CA 92030, USA
Telephone: 1-800-933-5997
Fax: 760 432 6560

UK and Rest of the World:
Spectrum Reports Limited

Research and Editorial Office
St Swithun's Gate, Kingsgate Road
Winchester SO23 9QQ
England
Telephone: +44 1962 878333
Fax: +44 1962 878334

Subscription Centre
St Swithun's Gate
Kingsgate Road
Winchester SO23 9QQ
England
Telephone: +44 1962 878333
Fax: +44 1962 878334

Email and Internet

Email:
spectrum@middlewarespectra.com

World Wide Web:
www.middlewarespectra.com

ISSN 1460-7220