MIDDLEWARESPECTRA

incorporating Enterprise Middleware

Contents

November 2001

Volume 15 Report 4

JMS and SonicMQ at Syntegra

Finlay Fraser
Technical Design Architect
Syntegra

Management introduction

Finlay Fraser is a Technical Design Architect at Syntegra, BT's system integration business. Syntegra is split into a number of vertical sectors, one of which focuses on capital markets, particularly dealer boards and trading room systems. (Other Syntegra markets include transport and logistics, government and defense). Of some 5000 staff, about 200 support the trading systems business, located mainly in the UK, USA, Germany — with additional locations in the Far East and Australia.

Mr Fraser is currently working on ITS Myriad — which is an integration framework for collaborative working involving clients, colleagues and trading partners. While his focus is primarily in the finance sector, the significance of the Syntegra work applies beyond finance — particularly in the context of customer relationship management (CRM) and how voice and data can be brought together in collaborative environments.

ITS Myriad uses SonicMQ as the message transmission base. In this case study he discusses:

- ***how SonicMQ is used and why it was chosen***
- ***Java, applications servers and their relevance***
- ***the differences between CRM and a trading floor.***

Setting the Syntegra scene

I have been at Syntegra for about 7 years, and I have worked all that time in the trading systems business. About two years ago we started a research project into what new and emerging technologies might affect our line of business as both our customers and we, Syntegra, moved forward.

At that time the primary business of Syntegra's trading systems group was, and indeed still is, centered around a voice trading platform, ITS, which we manufactured and sold around the world. We competed with a number of players to provide our custom built voice trading platform, which was specifically designed for the rigors of the trading room.

Yet our experience showed that the voice-trading platform needed to link to the trade capture platforms and other information sources in use in the data world. Initially this was conceived of via straightforward CTI gateways and developing systems integration around them. But we found that our customers would, in future, need us to lift our business application onto an open platform to enable comprehensive integration.

In essence the value proposition was, and is, moving out of our hardware into software. We could see that the hardware was being commoditized, especially with the development of VoIP (Voice over IP). This was clearly going to impact our business as financial traders sought greater integrated access to the data associated with the phone calls that were coming into their trading desks.

Let me put this another way. Think of traders sitting at their trading desks and talking to everybody left, right and center (inside their own organization and to counter-parties outside). Not only were they talking but they had quite separate sets of screens providing:

- **data facilities — like market data feeds and trade capture forms**
- **voice facilities — their trading turret.**

At the desktop the voice and data were separate. There was only a minimal level of interaction — with any connection made by the intelligent interpretation of the trader. In effect, the trader was the middleware.

In the past, trading systems were primarily focused on order entry. Traders placed and received orders by phone. These orders were entered into a data processing system. Voice calls were recorded so that, in the case of any dispute at a later date, there was a legal record of the transaction.

Then electronic trading took off — traders started order matching on-screen. What we did not know was the likely impact on our business. We created a research project — in an effort to find out:

- **what the effects might be**
- **what new opportunities might be generated in a much more converged market**
- **what technologies would be needed.**

The result was an integration framework — ITS Myriad.

Research purpose

Much of our research was devoted to VoIP and its implications. However, in parallel, we focused on designing an event handling middleware architecture that sought to converge many of the channels (both voice and data) by which traders access the market.

Our original premise was that in the future — meaning today — clients of financial institutions will 'touch' businesses by any, or all, of the following:

- **phones**
- **web sites**
- **email**
- **SMS (simple message service) and other wireless technology**
- **direct data feeds**
- **video**
- **etc.**

We felt that traders needed to be aware of all the mechanisms that might be used. In our research we linked various communications channels into a middleware framework in order to present — to the user, whether sales person or trader — a prioritized view of what clients were doing and what channels they were using to make contact. In addition, we introduced business rules to filter information and even clients: on certain days you may want to speak to a particular client (because a deal is going through) — while on others you may have other priorities.

Part of our design purpose was that these rules should be capable of manipulation by:

- **an organization**
- **a management layer**
- **individual business units**
- **individual traders or sales people (in some circumstances).**

The intention was that we should transcend both data and voice. We wanted to assist financial institutions (and their staff) to determine if a particular client was visiting (say) a specific Web page with identifiable research material. If a client account manager or trader could be informed of this, an otherwise unobtainable business opportunity might result.

The same could apply in other ways. A client, whose call might not always be a priority, might require urgent attention this week. Equally, if a specific financial stock reached a certain level, an alert should appear.

In delivering this flexibility we were quite clear that we were not trying to replace information feeds (like those from Reuters or Bloomberg). What we were seeking to do was produce a form of real time context-sensitive integration framework where the events displayed mattered 'now' — and probably only for the next five seconds (after that they probably did not matter):

- **if you had not picked up that call, your competition probably had**
- **if you had not noticed the client on your web site, he or she was probably not there any more.**

The same applies to email. We were not seeking to replace an email client or to display email. On the other hand we were trying to say 'of all the incoming mail, here is a relevant email that is so important you must read it now'.

Our focus, therefore, was on displaying events and routing — filtering real time events where the nature of the event is not longevity; rather it is the shortness of its relevant life. What we were trying to do was enhance the quality of information presented — in whatever form — to traders and sales people so that they might take faster actions on the most time critical events occurring.

Here I should make a key point. We are the opposite of a Call Center. What many fail to understand about trading and Call Centers is that they are inherently different. In a Call Center the aim is that a generically trained set of operators be able to respond as quickly as possible to an arriving event (typically a phone call).

It does not necessarily matter which operator picks up, only that when an operator does pick up the call they have the necessary skills and information. In the Call Center scenario, the call arrives and is processed so that when an operator picks up the call the appropriate data is presented at the same time.

In a trading environment, our intention is to empower traders and sales people. We want to offer alerts or pointers to those events which the trader has already flagged as interesting. The integration framework, our method of delivery, alerts the trader to those events occurring in the market which may cause a trader or sales person to take action. We recognized that it is the staff handling the event that adds the value. Unlike CRM, we did not intend to build business logic to make decisions for Call Center operators. Rather we want to assist the trader and sales people to make good decisions and more of them.

In essence our research objective was to prioritize the information that a trader wants in a meaningful way. That, at least, was our initial concept.

Delivery, application servers and asynchronous event handling

The project team evaluated a number of ways to tackle the requirement. One emerging technology we felt had the most promise, and relevance, was what is now called an 'application server'. The attractions were various:

- **J2EE (Java 2 Enterprise Edition) was supported by many manufacturers and was, thereby, largely open**
- **we could write nearly everything in J2EE Enterprise Java Beans (EJBs)**
- **these 'EJB' components would then run on application server run times**
- **there would be no need to worry about scalability or fault tolerance or load balancing — because application servers (and their components) could be replicated as needed.**

Looking back, what we did not appreciate was that these application servers — especially those that were available two years ago when we started — did not really possess an asynchronous event handling mechanism. They were good for synchronous processing, for circumstances when you did not mind 'blocking' as you waited for a response. But this was not the nature of the trading environment.

The nature (of what we intended to process) meant our needs were almost totally asynchronous. People do not warn you that they are going to phone you. You receive a phone call, as it happens. You receive an email or SMS message, as it happens.

Similarly, movements in the price of US$ to Yen or the Euro cannot be predicted or anticipated. They happen, through natural trading fluctuations in response to market volatility.

In such an environment, an asynchronous event handling mechanism is a necessity. But the application server we had selected lacked this capability. The result was that we soon had to face the fact that:

- **the bulk of our code would have to exist outside the application server**
- **we needed a messaging infrastructure (which we did not want to write for ourselves).**

Messaging

At this point we started to research the market to find out what messaging we should be using. The principle influence here derived from our original decision to write the entire ITS Myriad architecture in Java — so that we would not be tied to any specific platform. We wanted portability and flexibility, and J2EE looked promising as an accepted standard.

Although we had worked, in other parts of the business, with both TIBCO's TIB and IBM's MQSeries messaging (both of which are strong in the financial community), everything we had written thus far had been in Java. It made sense, therefore, to adopt JMS (the Java Messaging Service) — which is the messaging specification that is part of J2EE.

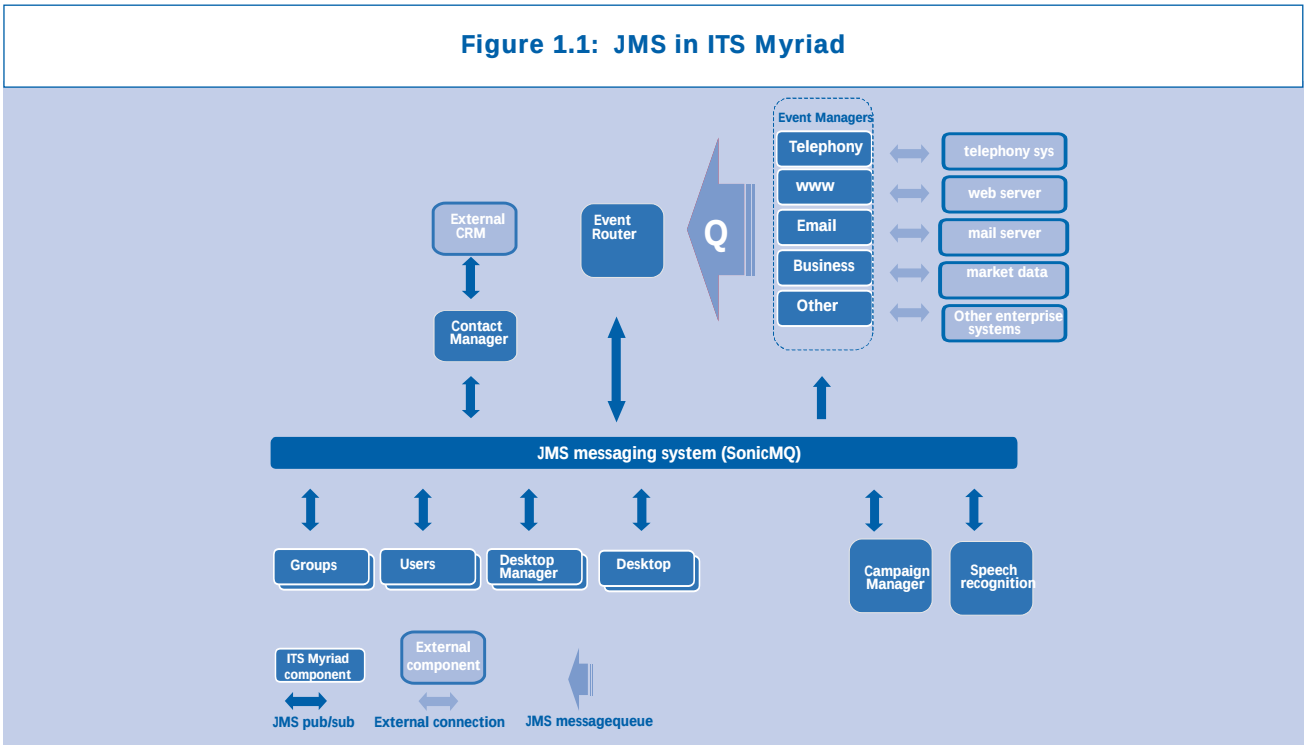
This had its own complications as, at the time we were originally looking, there was little more than the odd reference JMS implementation. Fortunately, by the time the research team understood the need for asynchronous capabilities, there were several products available — and these have firmed up significantly.

In the traditional way, we went out to look at all these — using our own criteria. Of these criteria the most important was that our messaging had to be fast, very fast. Telephony phone calls, from our trading switch, can generate in excess of 40 or 50 messages a second. In addition the typical deployment of ITS Myriad involves not only the trading system but also:

- **the corporate PBX**
- **every email coming into a company**
- **every web hit landing on the corporate web site.**

This demands an architecture capable of handling heavy message loads. My point is that our view of messaging was that it would run throughout ITS Myriad — and all its associated functions (Figure 1.1). It was not restricted to the trading desk. We needed the messaging to hook everything together — from trading desk to PBX, to emails, etc.

Figure 1.1: JMS in ITS Myriad



Independence

Another criterion was independence. A point to make here, and this assisted us greatly, was that originally we were not building a product. We were in research mode.

This meant that we were not bound to a formal specification which, in turn, enabled us to be flexible about which route forward to take.

This mattered because, in our experience, it is difficult to make network-based architectural decisions until you are close to deployment of a product and we were very keen that any object we might use should have no dependency on any other object.

That was another attraction of JMS. Our idea was that, if we really needed to scale and process major loads, we would be able to add the extra processing as needed. In addition, this could be located either locally or geographically elsewhere.

By using JMS as the interface between our component objects (for example, EJBs), we obtained independence. Whenever we come to a network decision, we know that we would have the flexibility to architect this in almost any way that suited a particular customer. Clearly some ways are better than others. But pragmatism is what counts with most customers.

Let me offer an example. One of our customers in the US has main offices in New York and Chicago, with satellite offices elsewhere. It can use ITS Myriad across these sites — really as one logical system, which we have termed the virtual trading organization. To support this it has a comprehensive broadband network.

It does not matter where it locates processing components in the 'network cloud' — Chicago or New York or elsewhere. Indeed, other factors can assume greater importance — like availability of support options for different parts of the operation.

Ease of installation and configuration

Our third selection criterion was ease of installation and configuration. This always matters to us. Often the first products to market may work but they are not easy to install and configure.

For Syntegra, with large customers located across the globe, straightforward installation and configuration is vital. We do not want to have to send design engineers on-site. We want to simplify installation, configuration and subsequent support activities.

A major consideration was, therefore, the degree to which the various JMS product vendors could enable straightforward:

- **installation**
- **configuration**
- **maintenance**
- **resilience.**

Publish and subscribe, queuing and ActiveX

The fourth criterion was a publishing and subscribing capability — in addition to point to point queuing. This was needed because certain types of event would need to be broadcast, say to all dealers — although only some traders or sales people might actually subscribe to particular events. Decoupling the publishing activity from the decision to subscribe is a powerful efficiency tool.

At the time of our evaluation, not all message brokers offered both point to point queuing and 'pub/sub'. Usually they supported one well and one rather less well — or their implementation of one was strong and the other noticeably weak.

A related factor was ActiveX. Research showed that customers were quite happy with J2EE and JMS in principle — but they also knew that they had Windows on the desktop. Any solution from the Java world had to provide ActiveX support to be credible.

In effect customers were happy with the Java-based server-side architecture of ITS Myriad, but this only applied so long as existing applications (particularly Microsoft ones, as Windows is dominant on the desktop on trading floors) could be connected in some way. For this we needed bridges or something similar. We did not want to have to write these ourselves.

Another selection criterion became, therefore, the ability of JMS vendors to offer us some strategy or scheme for interworking with ActiveX, MQSeries, the TIB and the other messaging forms in common use in the financial sector.

Selection

Initially TIBCO was a strong candidate — not least because we had extensive experience of using the TIB and it had a leading position for supplying financial market data. When

we started the research two years ago, however, it did not provide JMS support and so had to drop off our consideration list. IBM's MQSeries, also used extensively by the finance sector (albeit for different purposes than the TIB), lacked the performance and depth of JMS support that we needed.

The principal JMS contenders were, therefore:

- **SpiritWave (from SpiritSoft)**
- **FioranoMQ (from Fiorano)**
- **JMQ (from Sun)**
- **SonicMQ (from Sonic Software, part of Progress Software).**

In the end, speed, performance and ease of installation, configuration and maintenance prevailed as the most important issues. After substantial investigation, SonicMQ 'won the beauty contest' in benchmarking each proposition. It possessed the speed — it was the fastest of all. It was also easy to install.

There was one other significant plus point about SonicMQ, although this had not initially been one of our criteria. SonicMQ worked with the Net Dynamics application server (which has now been superseded by the iPlanet Application Server) — and seemed to be the only one that could do this with any competence.

Balancing this, there was one other governing factor. This was, and is, that we should be able to preserve independence by finding it relatively easy to replace one JMS vendor with another — if we had to do so. To ensure this capability was not destroyed has meant that we have been careful to observe the JMS specification and not use any additional features that might only be available within one particular JMS product. [With JMS, it being an API specification, vendors can build their own delivery vehicles as they wish, adding extra functions as they see fit.]

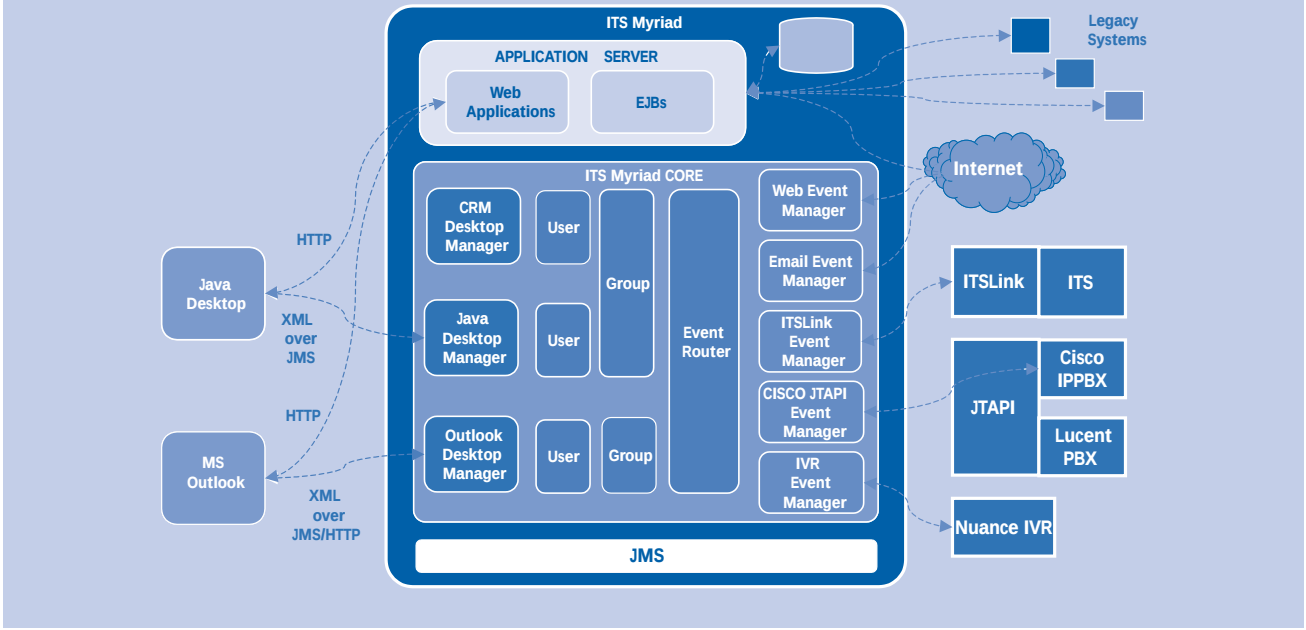
Usage

In the architecture we developed we have an inbound event queue (Figure 1.1). This is the 'sink' into which information from the various external systems — the sources of events like the ITS phone system, the PBX, Web servers, email, etc. — is directed. That queue is then read by a component, which we call an event router.

In practice there may be multiple such event routers, usually for load balancing purposes. These would all receive events off the same inbound queue — although each event could only be read once (we did not want the next message being read by more than one event router).

The event router component is responsible for working out which topics are relevant to each event. It then publishes

Figure 1.2: ITS Myriad



the information (and those who have chosen to subscribe then receive the event).

Originally we did discuss the possibility of using point to point queues — until we realized the size of the problem that we were creating for ourselves. Just think about a trading room of 1000 people, with events coming in at the rate of 40-50 per second and the definition as well as processing complexity that would be needed to ensure that each event reached each designated queue (and person) on a point to point basis.

Remember that most events have more than one potential consumer (an aspect that the 'point to point brigade' often overlook). Too many queues are a nightmare to set up and manage — which is why 'pub/sub' appeals.

Another aspect I should mention is that we foresaw all the messaging running outside the application server (Figure 1.2). The primary reason was that we realized that we needed to make less use of the application server's capabilities than we originally expected (mostly due to the technical constraints mentioned earlier). Instead, the message system identifies callers, as these arrive at the desktop. What ITS Myriad does is add data to the raw events.

The raw event might be your phone number. We would perform a translation of that into your name — and your company — to display this. If the sales traders are interested (in that call), they can fire off a request for more information.

The event router

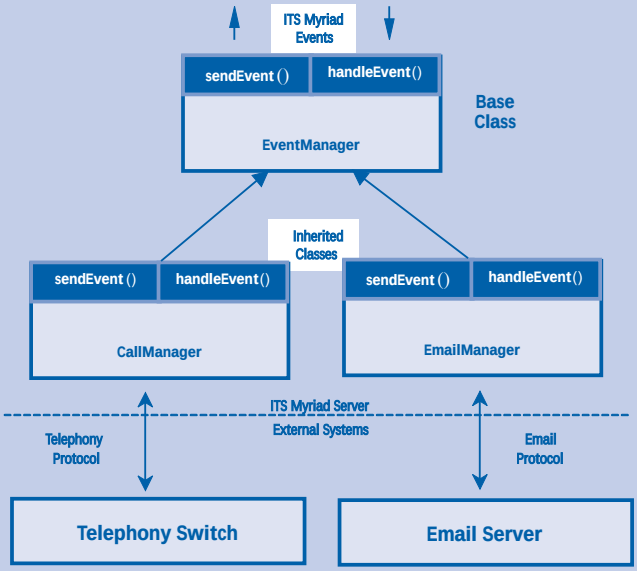
The event router is responsible for processing and deciding upon a destination for incoming events from the event managers (Figures 1.3 and 1.4). The business event managers:

- **provide the interface between ITS Myriad and market data or news systems — conforming to standard interfaces (like that of TIBCO's TIB for market data)**
- **monitor the arrival of data events and then pass these onto the JMS queue.**

It is the event router which receives events — and data — from the JMS queue. It is written in Java. The advantage of this is that we can locate it anywhere that suits the customer, which usually turns out to be determined by the expected loading.

As I described earlier, one of the advantages of Java (and JMS) is that we can deploy multiple event routers, all consuming the same queue. Our event routers are like worker objects — you can have a pool of worker objects and,

Figure 1.3: ITS Myriad to external communication channel



depending on the load, you increase or decrease the pool.

Typically, in the deployments we have done to date, they are running on Sun's Solaris, and we run the event router and handler. But that is not essential. The SonicMQ messaging provides the backbone so that the components on their own do not do anything other than communicate — receiving from the queue, as well as publishing or subscribing.

The event router is, in effect, a large rules engine (Figures 1.3 and 1.4). It holds the rules, which determine which events, coming in off the SonicMQ queues, are published to which topics.

Connections with CRM

One question we are asked a lot about is whether users can define the rules. Users can decide which topics to subscribe to, but it is at the discretion of the business as to the extent to which rules are determined by the business or by the individual. There is a further dimension. Increasingly we are integrating ITS Myriad with CRM systems.

In practice, CRM applications are collections of business rules. The CRM system, therefore, holds the rules about how certain processes should behave.

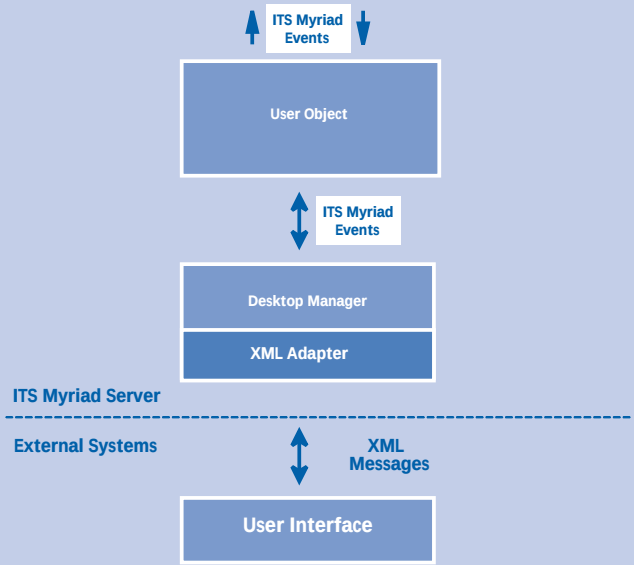
Given that this is the case, I think we shall see any particular user (who is interested in particular topics) obtaining the information by subscription. Whether that subscription is from ITS Myriad or the CRM will be determined by the process and choice of the business users. Syntegra's ITS Myriad is not designed to compete with any CRM system — it complements them specifically for the wholesale trading environment while extending the concept of relationship management across multiple channels of communication.

There are, however, some key differences between the way we exploit messaging through SonicMQ and the way that CRM handles apparently similar processes. For example, many people think that we look like a traditional Automated Call Distribution (ACD) function, where we are doing all the main call distribution — very like that found in Call Centers.

The difference is that we do not intercept calls, or do preprocessing before handing the call over to the operator in a typical Call Center. With ITS Myriad the phone rings, irrespective of any processing we may add (like identifying who is calling, presenting an account history, bringing up other relevant information, etc.).

The ringing at the sales person or trader's desk is irrespective of whether any additional processing has been accom-

Figure 1.4: ITS Myriad to user interface connection



plished. Furthermore, calls will (typically) be presented to several traders at the same time.

This is the converse of Call Center processing — where presenting a call to an operator without information largely incapacitates the Call Center operator's ability to handle the call. Traditionally the CRM application:

- **receives all calls**
- **makes a skills-based routing decision**
- **directs the call to a particular agent**
- **marks that agent as busy.**

In a trading room environment, or even with a normal PBX user, there is no option for interception. The call will already be ringing. All you can try to do is to improve the pertinence of the information presented with the call.

In the Call Center world the CRM system is in control of the call and will route it to a single operator. The trading world is different. In a trading room there is group working where one team may look after a particular client, region or instrument type. Anyone in the team can take the call.

The way we accommodate this is to use SonicMQ, the event router and the concept of publish and subscribe. The paradigm we have to work with is one where the call has already gone through and the sales trader may 'receive' multiple events at once (say four phone calls, three web hits, some emails, ...); it is the sales trader who prioritizes which event to choose to deal with.

Likes and dislikes

I would like to go back to SonicMQ. As you will have appreciated, I have not discussed its use or capabilities at great length. There is a reason. It works. Once we had it running, typically we have been able to forget it — in the sense that what works does not require much attention.

Having said that, now that we have our first operational projects installed, we are going back to look at what else we might exploit in addition to the ability to connect to multiple event sources, as well as routing them through to any desktop or trading position.

One aspect that already looks particularly attractive is the clustering that SonicMQ possesses. The 'clusters of clusters' capability looks promising for building global message backbones.

In addition SonicMQ's security has improved. It can support certificate-based access. This is relevant because we have architected the application as manageable modules that can be combined in innovative ways to suit Syntegra's customers.

Anything, like messaging, that assists this, will help us become more dynamic. Ultimately we need to offer complete flexibility for, say, a head trader to go in and drive a chosen trading strategy by reconfiguring certain users in response to market events. At that point, I think having certificate-based access across a global network will be vital — if only so that someone in a Hong Kong office cannot bring down half the London or New York trading floors.

My final thought on Sonic Software, as a user of SonicMQ, is that we have received good support — even at the research stage of the project. That impressed us — as did the access to development staff. They responded well, which made our research that much easier and more fruitful.

Lessons learned

In our business you have to 'take the pain' to gain from emerging technologies. If you do not do this early on you are disadvantaged. Our customers buy our products because we 'take the pain' on their behalf — so that they can take advantage of that pain as incorporated in our knowledge and skills.

So, although I might have given the impression that there was a certain amount of suffering, that learning is how we gain our expertise, which is the basis of our business today. Syntegra is always seeking to be at the forefront of understanding emerging technologies.

J2EE is a largely open standard. We have proved this. It matters because our customers have different technical environments. No project is the same. The more that middleware can be seen to be — and is — flexible, the easier it is to deliver.

We think the use of open standards — and quickly being able to adapt and evolve — often shapes a project's make or break point. We think that having open standards — like J2EE, JMS and Java — allows competitive advantage to be gained.

Earlier, I mentioned the desire for flexibility. This is even more significant where skills are involved. Using software based on J2EE improves our skills leverage. ITS Myriad has proved that — with Java, with J2EE and with JMS. This has been a good lesson and has definitely provided a productivity bonus.

Finally, and retrospectively, we would say that the ability to manage a product in the field turns out to be a bigger issue than performance. Performance is increasingly a given. Most customers expect it. Now they judge you on your ability to manage what you have built and installed.

Management conclusion

While Mr. Fraser may specialize in supporting the wholesale finance sector, what he says about Java, Java Messaging, J2EE and application servers has relevance to most organizations. In addition, the experience in building ITS Myriad has thrown up intriguing practical differences between the traditional Call Centers and trading environments.

Such differences are valuable. But of equal value to many large organizations will be the knowledge that companies like Syntegra do believe that their value lies in trying and proving technologies early — so that they have the experience to deliver. If this had not occurred with ITS Myriad, JMS/SonicMQ would have gone unresearched.

Brittle messages make for brittle transaction processing systems

Wayne Duquaine
Principal
Grandview DB/DC Systems

Management introduction

*Wayne Duquaine, of Sebastopol, CA, is a consultant and developer. In previous contributions to **MIDDLEWARESPECTRA** he has critiqued the continuing deficiencies of communications (and interoperability) in current Transaction Processing (TP) Monitors, particularly high end ones like CICS and IMS.*

This analysis continues in the same vein, albeit it in a Web and business to business (B2B) context. He examines where data communications, in a TP environment, are failing — especially when compared to databases. He assesses message broking and application servers (like WebSphere) — and locates their utility. Finally he explores which of two models will likely win — either the traditional TP monitor being natively equipped with Web capabilities or application servers 'acquiring' transaction processing functionality.

A real world example: burning compile cycles in the Midwest

An example of the type of problem produced by today's TP data communications support is evident at a major Midwestern insurance company. Its current operation is set up as follows:

- **Windows PC clients (using PowerBuilder) link to CICS on AIX — acting as the middle-tier (primarily for routing and fail-over); PowerBuilder supplies everything in COMMAREA format to CICS**
- **the CICS on AIX Distributed Program Link (also known as ECI) connects to CICS/MVS — where the CICS/MVS component runs the actual transactions and performs the updates to DB2.**

From an application development standpoint, every time this company moves a field in the COMMAREA, several applications break (whether PowerBuilder ones or CICS ones or both). Every time a field is added to the COMMAREA, a number of either PowerBuilder or CICS applications break. To resolve the consequent problems — arising from adding or moving a field — requires re-compilation of what seems like the universe.

This company is not unique. Many customers have gone down the DPL/ECI rat hole. Others have jumped on the CORBA (or Java RMI) bandwagon and found an equivalent rat hole. It does not matter that these are 'newer'. You end up in a similar fix, just as with CICS or Tuxedo. Add or move a field in a distributed CORBA or RMI application, and you have to re-compile everything that touches it.

The success of transaction processing monitors and managing data

Traditionally, TP monitors provide two key roles in an enterprise:

- **the multiplexing of large numbers of users who require access to databases or other critical files**
- **the provision of application level, data communications support for large networks of users.**

The TP world has made great strides forward with data management and relational databases. This 'data side' has successfully incorporated:

- **logical versus physical data independence**
- **self-description of data (via system catalogs)**
- **tools for automating the extraction of data.**

For example, new fields can be added to a table, or viewed, with no impact on existing applications. Coercion between different data types in the database, and application data fields, is automatic.

What has happened here is that applications have been logically and physically separated from the underlying representation of the data on the DBMS's disk files. In contrast, the data communications side of the house has not appreciably evolved past an elderly VSAM or flat file mentality.

The lack of evolution with data communications

The data communications side, by and large, remains mired in rigid fixed format messages. It still resembles the 1970s. While IMS provides some limited logical versus physical message independence, virtually all other solutions require a 1:1 mapping between data message format and the associated application data. Notorious examples include the COBOL COPY books used in products such as CICS External Communication Interface (ECI/EXCI) and CICS/WebSphere Business Logic Interface (BLI).

Self-description of data — via standard (catalog) facilities — is almost non-existent. Adding new fields or moving fields around in a message is impossible — without breaking existing CICS/ECI or CORBA/COM applications. At a minimum such applications must be re-compiled before they will work with an altered message structure. In addition, they frequently require code changes to handle the new fields or changed order of the fields.

Coercion between different data types in the data message format and the application data fields is, therefore, extremely limited — or not allowed at all. The net result is that the data management aspects of TP work well but the data communications aspects continue severely to impact both application:

- **development**
- **migration.**

Rigid formats: networking in a strait jacket

Nearly all of the current major transaction processing data communications paradigms utilize fixed format messages, many of which depend upon application-embedded

knowledge. The exact structure of the data sent to and fro is usually described in COBOL COPY books (or C structures) and compiled (embedded) within one or more applications.

Common examples of this approach include:

- **the CICS Distributed Program Link (DPL) — otherwise known as the External Call Interface (ECI) — which uses a fixed format 'COMMAREA' data structure**
- **Microsoft's Host Integration Server**
- **fossilized 3270 screen applications, with their associated fixed screen formats.**

Newer technologies coming on stream include (or have included) CORBA, DCE and DCOM. While such relatively newer technologies solve data conversion issues (ASCII to EBCDIC, little endian to big endian, etc.), they require strict one-for-one correspondence — of parameters and their data types — between client and server. If a parameter is added or moved around, or the data types between client and server do not exactly match, the call is rejected. The programmer then has to modify and re-compile the program.

There are presently no data communications equivalents of ALTER TABLE (where you can add a new field to a table without forcing all applications that use that table to be re-compiled). Instead, data communications is still locked into a tight coupling of the logical and physical message layout together (or, in RPC, the logical and physical ordering of parameters in an RPC message). The kind of the data independence that you routinely use in RDBMSs is utterly lacking. Contrast the following:

- **when you write data serially onto a disk (as with DBMS), you know you obtain data independence**
- **when you write data serially to a wire (as in data communications), we completely forget what the concept of data independence means.**

The consequence, expensively for almost all organizations, is that application embedded knowledge reigns supreme. Whether it be ECI, CORBA, DCOM or DCE, all of the current buzzword technologies are too rigid and too brittle for the loosely-coupled style of computing that defines:

- **Web-based intranet or extranet applications**
- **the promised world of Web Services applications.**

Poor adaptability: dinosaur paradigms, dinosaur applications

Most existing TP applications were built using 1970s paradigms that are now obstacles rather than advantages. These paradigms were rooted in 1970 state of the art technologies that have increasingly fallen behind the needs of today's Web-based approaches. For example, ECI (IBM) and Host Integration Server (Microsoft) are both based upon mid-1970s SNA LU0 where distributed program support was built into the old 3790 and 3600 controllers.

These approaches critically depended upon:

- **'application embedded knowledge' of the message data structure**
- **a very tight coupling of data between remote modules (the bane of structured programming).**

The days of 'application embedded knowledge' in a world of open systems and Web-based automated business-to-business (B2B) computing simply is not adequate. It will not suffice. Programmers today expect that making changes to the underlying DBMS tables (adding columns, etc.) should have no impact on existing applications. Yet data communications applications still continuously break whenever an equivalent change occurs in a message.

As we move to integrate and create loosely coupled or 'federated' systems, having an application break every time a field is added or moved around in a message is dysfunctional. We must:

- **move away from 1970s data communications models**
- **instead build in the same kind of logical/physical 'adaptive coupling' or loose-coupling which already exists in DBMS applications.**

Message brokers are no more than band-aids

The latest TP middleware vendors frequently advertise 'message brokers' (or 'integration brokers' or 'application brokers') as the cure-all for the difficulties of getting different types of TP applications to interoperate. While message brokers try to come to the rescue, they are in reality gigantic band-aids which do little more than paper over the fundamental design problems described above:

- **interoperability falls apart because there is no independence of the physical data from the logical data in the messages**

- **the lack of self-description of the data makes program changes to data area or parameter layouts brittle — and breaks everything any time a field is moved or added.**

Message brokers do nothing, therefore, to solve the underlying data communications paradigm problems. They only provide a gloss, which appears to hide existing, and basic, design problems.

Know thyself: self-describing data

Arguably the key lesson learned in the migration from VSAM and other flat files (or indexed files) into RDBMS technology was the notion that the data itself (tables, fields/attributes, constraints) is describable, via the use of 'metadata'. In DBMSs, this is kept in standard system catalogs that can be programmatically extracted at any time by any authorized client.

A similar phenomena needs to be introduced for the data communications portion of TP. Message formats need to be externally accessible and programmatically extractable. These should be built-in functions, just as system catalogs are built-in functions for a DBMS.

The rise of XML and XML message descriptor languages — such as Web Services Description Language (WSDL) — offer one possibility for providing a meta data structure to data communications messages. Using Web-based registries (like UDDI with WSDL), the entire logical content of messages can be externally described, retrieved and used to process (XML-based) messages being sent or received.

On this basis, organizations could build their own UDDI/WSDL registries. These would initially act as repositories for in-house (intranet) based messages — those used internally. Such registries, and associated WSDL descriptors, would become the data communications equivalent of system catalogs. An additional attraction is that they would apply on a network-wide basis, not merely on the local system.

Challenges raised by integrating the Web with TP

The latest industry efforts involve integrating the Web into TP environments — even though most of today's high-end TP data communications components continue to suffer from the constraints described above:

- **rigid fixed format, and request/reply, formats persist; they have minimal data independence**
- **most of today's messages lack any accompanying self-description information; they rely upon fixed format COBOL COPY books or rigid IDL structures, both of which are 'compiled in' to applications**
- **existing applications often poorly adapt to the Web (put bluntly, 3270 BMS may be acceptable for some simple Web server support but it is quite insufficient for delivering Web-based B2B applications; ECI is no different)**
- **nearly everything offers poor Web security support.**

If we do not start to focus on fixing and re-thinking the way we design data communications messaging infrastructures, then any OLTP related Web-based application implementations will continue to be plagued by the same recurring issues. The net effect will be significantly to reduce application and data inter-operability over Web-based infrastructures. The consequence will be even more complex band-aids.

Yet e-business (B2B) delivers its biggest payoff when automating different business processes (ordering, re-stocking, etc.). This type of processing requires a program-to-program paradigm — such as a Remote Procedure Call (RPC) or messaging. This is very different from a human sitting at a 3270 or at a Browser. Screen-scraping (an already dysfunctional technology) becomes even more dysfunctional when moved to the Web.

Such albatross technologies only delay forward movement. Yet they consume scads of designer and developer time as highly paid professionals seek clever fixes in attempts to make a quart fit into a square (a severe muddling of analogies that is entirely intended).

Web-based infrastructures are increasingly being used for both intra-nets (in-house networks) as well as extra-nets (external networks — usually selected customers and partners) and even to public internets. Gradually these networks are replacing existing proprietary networks (SNA, 3270, DECNet, etc.) — because of the lower cost and simpler operation of TCP, Ethernet and browsers.

This transition will eventually require that HTTP (the protocol that is ubiquitous to browsers and Web technology) be integrated into TP monitors. The result will be that the TP monitor will become a Web Server on steroids.

B2B applications will not only have to interact with the corporate TP monitor, but will eventually have to be integrated into and run inside the corporate TP monitor, since it is the

corporate TP Monitor that runs most of a business's mission critical business processes.

TP Web security: squeezing cats through keyholes

As previously mentioned, a key aspect of transaction processing over the Web involves program to program RPC or messaging. But, presently, most of the security mechanisms used to access high-end transaction systems (like CICS or IMS) are deficient or inappropriate:

- **DPL/ECI sends passwords in the clear (LU6.2 or TCP62)**
- **CORBA and IIOP do not co-exist well with firewalls**
- **DCOM does not work well with firewalls or mainframes.**

To be fair, all of the above were designed for in-house or closed networks where security was a controlled, and known, commodity. But the Internet and the Web impose change — and none of the above natively work with HTTP (the lingua franca of the Web). Trying to connect unlike mechanisms most often produces a kludge. It is not unusual to find organizations which have:

- **created 'tunneling' support (gateways) — to convert the data to/from HTTP**
- **define special firewall ports to which protocols (like CORBA or RMI) connect.**

Unfortunately, such approaches too often lead to security being compromised (or, at the least, being badly mangled). One security evangelist from Sun Microsystems aptly described today's approaches as being like "locking the door, then greasing the cat to try to squeeze it through the keyhole".

Building blocks for a more rational future

The future IT world will be composed of loosely coupled systems, operating over a Web-based infrastructure. It will use combinations of RPCs and messaging, with self-describing messages incorporating some form of logical/physical data independence. The current rigid message formats, and application embedded knowledge, present in today's TP systems are anathema to this. They will ultimately 'evolve' into oblivion.

While much is made of RPCs in today's marketing, an RPC is merely a fancy term for function that TP monitors (CICS, IMS, Tuxedo, etc.) have been providing for the last 30 years. For example, a CICS transaction is functionally a subroutine call (essentially an RPC) which invokes a sub-program that performs a set of actions against persistent data, based upon a set of data passed in with the transaction invocation.

The primary change is that these RPCs (transaction invocations) will be completely automated using program to program capabilities that operate across a Web-based infrastructure. This is very different to the obsolete 'human at a 3270' paradigm.

Transaction processing support for Web-based computing in the future will, therefore, require:

- **self-describing data: XML is a good solution for this as it is self-describing, is already highly ubiquitous and runs natively over HTTP**
- **built-in support for automatic transformation of messages: XSLT (eXtensible Stylesheet Language Transformation) — used in tandem with XML — is a good solution for this requirement as well as significantly reducing the need for today's special purpose message brokers (when messages need to be mediated); in effect, the message broker becomes part of the TP monitor and is relatively simple for customers to build (they just need to create an appropriate suite of XLST transform definitions)**
- **generic program to program support over the Web which encompasses both RPC and messaging modes of operation; SOAP (Simple Object Access Protocol) becomes a big win here — since it is built upon XML, can run either as an RPC (over HTTP) or as a messaging application (over SMTP, MSMQ, etc.); however, in order to play in this game for real, SOAP will have to mature much further — for example to include extensions for TP semantics.**

Successful future TP monitors will also need to incorporate Web Server-like functionality directly into the TP monitor. In particular, HTTP support will need to be integrated directly into the TP monitor.

The alternative to this scenario is to upgrade the Web Server with TP monitor-like capabilities, as some vendors are attempting. IBM's WebSphere is one example of this. It is a transactional application server that sits on top of a Web Server which includes support for transactional Enterprise Java Beans.

While such an approach is feasible, it will be interesting to discover which approach succeeds in winning the Web infrastructure race first:

- **if extending the application server (WebSphere) wins, CICS will begin a long-term decline: its use will be dedicated to running old ECI and 3270 applications**
- **if extending the TP Monitor approach wins, WebSphere and other application servers will essentially become redundant (after all CICS has been serving VSAM flat files a lot longer than WebSphere has, and CICS front ends far more DB2 installations than WebSphere does).**

If you think that this is excessively binary, remember that most customers will not run essentially redundant functions (transactional servers with Web support) forever in parallel next to each other.

The world of the future will see browser-based XML clients issuing transaction requests over HTTP to TP monitors, operating over intra-nets, extra-nets and the public Internet. Fat clients — such as PowerBuilder and possibly even Visual Basic — will also begin a long-term decline. They will be displaced by thin client, smart browsers supporting XML.

On the server side, too much of the so-called new technology being promoted by vendors is just adding extra tiers — Web application servers and Java — to front end old applications. The long term problem is that this does little to solve the fundamental design issues imposed by different generations of technology.

Management conclusion

Customers need to demand that their TP vendors — whether IBM, BEA or otherwise — step up to the plate and address the fundamental TP data communications application design issues that exist. Vendors must not be allowed to hide behind band-aids.

Instead, TP monitors need:

- ***natively to understand HTTP and XML***
- ***provide the support tools to integrate HTTP and XML with TP applications***
- ***use the self-describing capabilities of XML to isolate the TP applications from message format details.***

Anything else is just the recycling of existing, out of date tools by introducing snazzy-looking GUI and Java front ends to hide inadequate back ends. This can no longer be tolerated, if automated B2B (internally and externally) is the objective. Instead, if the TP monitors do not adjust, application servers will require TP monitor functions — with the result that CICS, Tuxedo, IMS/DC and the like will accelerate their long decline into oblivion.

Addressing pitfalls with EJB development and deployment across different J2EE application servers

Dan Rolnick
Senior Consultant
Cacheon

Management introduction

J2EE is the great technological hope for those who do not wish to follow the Microsoft .NET path. Application servers are all the rage. Over 50 different products are available. Competition is intense. The common assumption is that, if you deploy Enterprise Java Beans (EJBs) built using two or more J2EE application servers and development tools, they will work happily together.

Sadly, this is not necessarily the case. Indeed, J2EE-based application servers have — as most pessimists anticipated — specific incompatibilities. What you build for deployment on (say) iPlanet will not necessarily run on WebSphere or WebLogic or Oracle 9iAS ... and vice versa.

In this analysis, Dan Rolnick explores the dependencies that arise when developing an EJB for deployment on a J2EE compliant application server. He examines how vendor-specific implementations of low level services and entity bean persistence, as well as proprietary deployment descriptors, can irrevocably tie an EJB to a particular vendor's application server. He then looks at possible solutions.

The practical, business context

To remain competitive, enterprises must balance business and technology challenges. A primary challenge facing many enterprises today is the ability to develop rapidly and then deploy new business applications and services in response to changing customer needs.

The need for this — and the fact that not all Enterprise Java Beans are created equal — was brought home to us on a implementation project at a leading international financial institution. This institution wanted to create a new business service in response to its customer demands.

Previously, its customers had to log into:

- **one account application to access a current account balance and bill payment information**
- **a separate brokerage application, in order to access portfolio balance and loan availability information.**

But these same customers wanted to log into one application that would provide them with a consolidated view of both their bank and brokerage account information. To satisfy this, the financial institution needed to do something if it was to avoid alienating, and losing, customers.

In response, the financial institution accepted that it needed to create a new business service. For operational and financial, as well as common sense, reasons the preferred approach was to combine functionality that was already offered in the existing bank and brokerage applications in order to deliver a consolidated view of account information.

What the financial institution knew was that, by leveraging existing application components (to deliver the new service to its customers), it would obtain a competitive advantage (however briefly) because it would:

- **decrease the time to market for the new business service**
- **increase revenue generation as a result of the new business service**
- **improve customer satisfaction and retention.**

However desirable these were, as a set of justifications, the financial institution also knew that it faced a daunting prospect — the integration of components that had been built on two (or more) application servers and/or legacy production environments. Furthermore, these applications had been developed by different vendors and different teams at different times using different technologies. Integration, especially of the application server elements (on which I will focus) was not going to be easy.

Setting the EJB/J2EE stage

In the financial institution, the 'opportunity' involved:

- **a banking application deployed on an Oracle 9iAS application server**
- **a brokerage application deployed on a BEA WebLogic application server.**

As will be discussed, the problems arose because the applications and services ran on heterogeneous platforms. Despite the fact that both application servers were hosting J2EE-compliant EJBs, they were not easily integrated.

The reality is that EJBs are developed for deployment on a specific application server. These require modifications before they can be deployed on a different vendor's application server. This is because some of the code developed — the tools used for creating EJBs — establish a dependency on a specific vendor's application server.

Because of these dependencies, moving an EJB between different J2EE compliant application servers produces deployment problems. For example, deployment issues arise when trying to move an EJB developed for (say) a BEA WebLogic application server to an IBM WebSphere application server.

Additionally, EJB to EJB communication across J2EE compliant application servers can produce run time problems. For example, run time issues arise when an EJB deployed on (say) a WebLogic application server attempts to communicate with an EJB deployed on (say) a WebSphere application server.

Dependency details

Most of the functionality available through the Enterprise JavaBeans Application Programming Interface (API) is included in the javax.ejb package. This package primarily consists of interfaces. These interfaces are implemented by each application server vendor, each one of which is free to interpret and implement them as they think best.

Inevitably, differences occur. The EJB interfaces define the services provided by a particular container. They also define the client interface to an EJB.

For example, the container is responsible for managing the interactions between an EJB and the application server. Yet

each application server vendor implements the services provided by their own application server container. These include services such as:

- **security**
- **transactions**
- **multi-threading**
- **distributed programming**
- **connection resource pooling.**

The worthy objective of J2EE and EJBs is to free developers from having to be concerned about such low level services. If they do not have to be, they can focus instead on developing the EJBs which represent specific business logic. These EJBs can then make use of the functionality provided (to the EJBs) by the low level services.

The problem is not with the principles. An application developer can utilize the services provided by an application server container. The challenge lies in how these services are implemented by each vendor.

Difficulties arise because EJBs developed for deployment on one application server have dependencies on that application server's implementation of these low level services. Yet, in order to take advantage of the benefits provided by such services (not least, protection from the low level considerations), developers must implement EJBs in ways that make use of the services as implemented by that vendor.

The result is that EJBs are created which depend on the underlying vendor's implementation of the key low level services. Consequently, if you attempt to move EJBs between different J2EE compliant application servers — or even try to have them work together while running on different applications servers — incompatibility problems occur.

Other problems: client interfaces

Similarly, other problems occur with client interfaces to an EJB. The EJB architecture allows clients to invoke an EJB over a network via the home interface and the remote interface for the EJB. The home and remote objects handle the communication between a remote client and an EJB deployed in an application server container.

Although many container implementations handle communication using Java RMI, the EJB specification does not mandate which distributed object protocol should be used for communication between a client and the container managing the remote EJB. As a result, vendors have latitude in how they choose to implement the communication between a client and an application server container.

Furthermore, clients use the Java Naming and Directory Interface (JNDI) to locate and access a remote reference of an EJB on an application server. This requires the instantiation of an initial context object, which provides the starting point for the JNDI lookup. The initial context factory is part of the JNDI API and requires a developer to set properties that are specific to that vendor's implementation of JNDI. The type of initial context object which is instantiated varies, based on each application server vendor.

It turns out that each vendor has unique JNDI properties that must be populated. As a result, the initial context object used for locating an EJB deployed on one application server cannot necessarily be used for locating that EJB when deployed on an application server developed by a different vendor. Therefore, an EJB cannot call another EJB deployed on an application server developed by a different vendor without encountering run time errors.

Persistence issues and entity beans

Additionally, the tools available for configuring the persistence properties for entity beans which use container-managed persistence are almost invariably proprietary. They are included as part of the application server software.

These tools matter. They are used to:

- **generate tables from entity bean objects**
- **map container-managed entity bean fields defined in deployment descriptors to database fields**
- **provide support for complex object to relational mapping, such as a one-to-many relationship.**

These tools, however, can be used to edit entity bean 'finder' method expressions. Yet 'finder' methods are defined in the home interface for an EJB. Again, each application server vendor has a proprietary strategy for defining how the 'finder' methods are implemented.

With container-managed persistence, each finder method defined in the home interface must be explained to the container managing the entity bean. This explanation is provided to the container during deployment of the entity bean using vendor-specific deployment tools and syntax.

Being proprietary by nature, these tools bind the definition of an entity bean's persistence to the application server

software provided by that particular vendor — resulting in the development of entity beans that have dependencies on the tools that have been provided by the application server vendor. As a result, entity beans written to run on a specific application server cannot be moved to other application servers developed by different vendors without encountering deployment errors.

Packaging into JAR files

In addition, each EJB is packaged into a JAR file for deployment onto an application server. This JAR file includes the class files and the deployment descriptor for the bean. The deployment descriptor:

- **defines the deployment properties for an EJB**
- **directs the deployment tool (on the chosen application server) as to how to deploy the EJB within the container.**

The Enterprise Java Beans specification version 1.1 requires that the deployment descriptor be saved as an XML file named ejb-jar.xml in the META-INF directory. Each application server includes a container that manages the interactions between the bean and the server. Deployment tools use the ejb-jar.xml file to add a bean to the container.

Most application server vendors require one or more proprietary deployment descriptor files for deployment of beans. While ejb-jar.xml defines common EJB properties, the vendor specific deployment descriptors define custom properties for deploying a bean within a specific application server container. The container then uses both the required and the proprietary deployment descriptors to apply primary services to the EJB at runtime.

Similarly, a deployment JAR file for an EJB includes both the vendor's proprietary deployment descriptors and the required deployment descriptor. The proprietary deployment descriptors are specific to the application server vendor and are therefore incompatible with an application server developed by another vendor. Once again, deployment errors will be encountered when attempting to move an EJB across J2EE compliant application servers.

Possible resolutions

From all the above it should be more than obvious that 'all EJBs do not necessarily talk to all other EJBs' — unless they were built on and run on the same application server, which was not the case at the financial institution. The question is: "Can such difficulties be solved?"

An EJB can require manual intervention — the rewriting of code in order to accommodate the service implementation of another application server vendor prior to moving the EJB. To promote the development of EJBs that can be more easily modified to accommodate another vendor's implementation of low-level services, a developer should identify service implementations that are specific to a particular vendor. These services might then be isolated into service factories in order to create a layer of abstraction between:

- **the application framework**
- **the services provided by the application server vendor.**

Similarly, an EJB that is dependent on a specific vendor's implementation of persistence for container-managed entity beans may require modification prior to being deployed on an application server developed by a different vendor. Although entity beans can be modified using the persistence tools provided by the new application server vendor, this often results in EJBs that are still dependent on a specific vendor's implementation of persistence. To overcome this, a developer may consider using bean-managed persistence rather than container-managed persistence to handle entity bean persistence.

Finally, an EJB that includes deployment descriptors that are proprietary to a specific vendor's application server cannot be deployed on another vendor's application server without modifications. One solution here is to develop deployment descriptors that are required for the new application server. But this solution also requires that a new deployment JAR file be created for the EJB that includes the new deployment descriptors.

Another approach

However, these are not the only approaches. For example, there are products which complement application server tools and processes in order to automate the conversion of deployment descriptors. [One such tool is the Cacheon one (Figure 3.1), which was used at the financial institution.]

This tool is able to read proprietary deployment descriptor files for several vendors and convert the contents of the files to the deployment descriptor representation appropriate to the new application server. Figure 3.2 shows the deployment of a JAR file — BankAccounts.jar — for a session bean.

In this, the BankAccounts.jar file includes:

- **the META-INF directory**

Figure 3.1:
The Cacheon Business Services Center (BSC)

The Cacheon BSC is built upon a dynamic component architecture which acts as a collaborative environment for discovering, assembling, deploying and maintaining business services at runtime. The main tools that comprise the Cacheon BSC include the following:

- **the Business Services Engine (the 'Engine')**
- **the Enterprise Warehouse (the 'Warehouse')**
- **the Enterprise Warehouse Console (the 'Warehouse Console')**
- **the Business Services Console (the 'Console').**

The BSC Engine, an extension of the Java Virtual Machine, supports a local warehouse and the loading of business services into that local warehouse. Additionally, it supports the loading of modules that facilitate communication across Engines using 'Voyager'. Voyager provides Object Request Broker functionality and facilitates communication between warehouses and multiple application servers (including JBOSS, IBM's WebSphere, BEA's WebLogic, Oracle's 9iAS, with others being added).

A module is a dynamically loaded class that augments the capabilities of an Engine on a particular machine. This enables an Engine either to acquire essential capabilities or to discard unneeded ones, as requirements evolve.

The result is a modular environment that provides a secure way of augmenting the run time functionality of the Engine. The modular nature of the Engine enables functionality to be added, including support for a range of networking technologies, protocols and standards.

Users of the Cacheon BSC can publish business services to the Warehouse and are stored there. Business services that have been updated in the Warehouse can be propagated to users, using 'pull' methods. With the pull method, a user can refresh a business service, or a set of business services, to the latest version. When retrieving a business service, a user selects the version of the business service they want to retrieve.

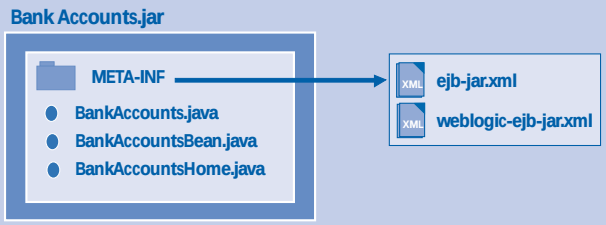
The real attraction of the Cacheon BSC, in a J2EE/EJB environment, is that EJBs that were built to run on different applications servers can be linked together — which is not always be possible. As the attraction of EJBs is supposed to be that they are building blocks, the BSC adds an otherwise missing piece to the J2EE equation.

- **the Java files that comprise the EJB.**

The META-INF directory includes:

- **ejb-jar.xml, which is required for EJB deployment**
- **the proprietary deployment descriptor required by — in this case — the WebLogic application server (weblogic-ejb-jar.xml).**

Figure 3.2: Deployment of a JAR file

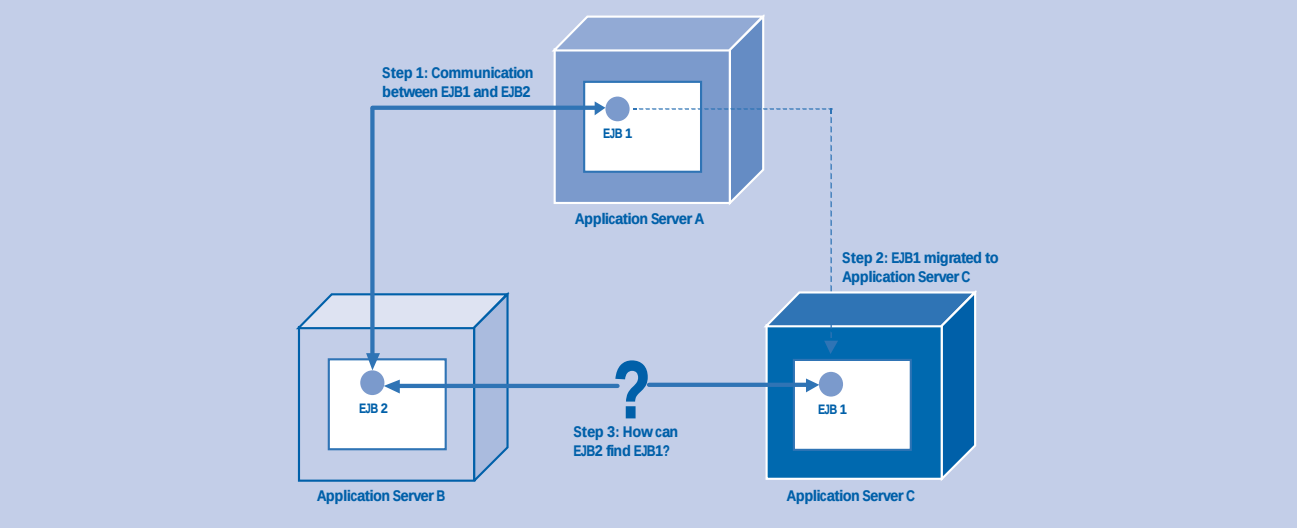


Assume that the session bean is currently deployed on a JBoss application server. This session bean could be moved from JBoss to WebLogic, using software to automate the conversion of deployment descriptor information from jboss.xml to the format required by weblogic-ejb-jar.xml. Furthermore, this session bean could later be moved from WebLogic to Oracle 9iAS, using the conversion software to automate the conversion of deployment descriptor information to the representation required by orion-ejb-jar.xml.

For all this to work, a developer must know about the issues which must be addressed if vendor specific implementations of context factories are used to facilitate communication between EJBs across application servers. Ideally, any such solution must include context factory implementations for multiple application servers.

In order to facilitate communication between EJBs on disparate application servers, context factories for one application server can be deployed onto another application server. This enables EJBs to communicate across application servers while eliminating problems that arise from vendor specific implementations of context factories.

Figure 3.3: Communicating EJBs



More complications, and solutions

However, a more complicated issue will arise when an EJB has already communicated with another EJB that has then been moved to an application server developed by a different vendor. This scenario is depicted in Figure 3.3.

Step 1 in Figure 3.3 illustrates the cross application server communication occurring between:

- **EJB1, an EJB deployed on Application Server A**
- **EJB2, an EJB deployed on Application Server B.**

If EJB1 is moved from Application Server A to Application Server C, as indicated by Step 2, the communication between EJB1 and EJB2 will no longer work. This is because the context factory for locating EJB1 is implemented for Application Server A and not Application Server C, where EJB1 has been moved.

In this scenario, a developer would need to utilize a solution which deploys the context factory implementation for Application Server C on Application Server B. This would enable Application Server B to locate EJB1.

Additionally, a developer could deploy the context factory implementation for Application Server B on Application Server C, thus enabling Application Server C to locate EJB2. Furthermore, using some form of automation software, both EJB1 and EJB2 can be recalled as services and modified in order to facilitate two way communication across application servers.

Management conclusion

To compete effectively, enterprises must respond quickly to changing customer, competitive and market demands. Further, they must be adept at functioning within a diverse and complex computing environment. This requires that existing enterprise resources be leveraged regardless of how these resources are developed and where these resources are deployed.

Unfortunately, despite the hopes for application servers, EJBs and J2EE, incompatibilities do exist between different application server and J2EE implementations. While not major, these issues are not simple to resolve — being located as they are in the implementation of low level services. Nevertheless, the result is a headache for enterprises trying to make maximum use of J2EE and EJBs.

Although there is a push to develop standards between the different application server vendors, the desire for competitive advantage inevitably takes precedence over the acceptance and introduction of standards. This, coupled with the fact that the Java standard is itself still evolving, makes the development of standards between application server vendors even more unlikely.

That is, however, not the whole story. As Dan Rolnick has described, alternate solutions (like the one from Cacheon) do exist and work. They do this by providing 'reference points' for EJBs. With these, different implementations of EJBs are able to work together — which was the original objective after all.

Application brokers or application servers: a 21st Century dilemma

Charles Brett
President, C3B Consulting Ltd. and President, International Advisory Board, MIDDLEWARESPECTRA

Management introduction

In today's software market place there are two broad categories of product which go under similar names but are, in practical implementation terms, very different. Both can deliver quite similar results, albeit via different paths. These categories are:

- ***application servers (iPlanet, WebSphere, WebLogic, 9iAS, etc.)***
- ***application brokers (MQSeries Integrator, e-Biz Integrator, e*Gate, etc.)***

Customers are constantly being invited to choose between these. There is, however, a lack of comprehension about how these are, at the same time, so different and so similar.

Application servers

In general terms an application server is a run time engine on which applications can be run. In this context, application servers can vary from the simple to the sophisticated. Arguably, an operating system, or even a database on an operating system, is an application server — you can run applications on such a server ... Yet application servers, in their more complex guises, are designed to be scalable, secure and transaction-capable (Figure 4.1 shows a list of 'typical' functions).

In today's market terms, an application server tends to be a specific run time engine which has a variety of technologies associated with it. For example, IBM, BEA and Sun have all built their application servers around Java. Java considerations (constraints) permeate their whole approach to application servers, although not necessarily to the exclusion of previous technologies (as both WebSphere and WebLogic demonstrate).

Unfortunately, the name 'application server' is not just associated with running applications. It is also heavily associated with application development in general (and Java in particular). One has only to look at the Oracle 9iAS Web site or IBM's WebSphere or Sybase's Enterprise Application Server product families to see that each of these comprises and gathers together multiple different varieties of run time engines plus multiple varieties of development tool (in IBM's case as varied as Visual Age for Java, C++, C, ... through a host of other tools).

There is, therefore, a common confusion about what an application server is. This applies as much to the vendor community as to customers. Neither seem quite sure whether an application server is about:

- **application development (where most people-time is spent)**
- **or application serving (when the ultimate value is delivered).**

Application brokers

Now think about application brokers (which are sometimes known as integration brokers or even message brokers). These are also software products and are available from a myriad of vendors, including:

- **IBM, with MQSeries Integrator (and even MQSeries Workflow)**
- **Sybase, with its e-Biz Integrator/Process Server family**
- **SeeBeyond, with its e*Gate collection**
- **webMethods, with its Integration Platform**
- **TIBCO, with its Integration Server**
- **Sun, with the iPlanet Integration Server**
- **some 30+ other organizations.**

Application brokers sit between existing applications and enable these applications to work together (Figure 4.2). An application broker 'brokers' the linkages between the

Figure 4.1: Application Server — typical components

J2EE support
Java Servlets
Enterprise JavaBeans (EJB)
Java Server Pages (JSP)
Java Database Connectivity (JDBC)
Java Naming and Directory (JNDI)
Java Transactions (JTA)
Java Messaging (JMS)
Java Security
RMI
HTTP
SQLJ support

Web Server to JSP/Servlet to EJB connectivity
HTTP and HTTPS tunneling
Load balancing
Availability
Connection re-routing
Application failover
Clustering
Session state replication (and failover)
IP multicast
Logging
+ copious and varied development tools

source and destination applications, thereby reducing the potential number of interfaces from ***n(n-1)*** — where ***n*** is the number of interfaces — to ***n*2***, a significantly more manageable number.

As such, application brokers are a mechanism for connecting applications — which explains their utility in EAI (Enterprise Application Integration). They reduce 'application connection spaghetti'. Furthermore, they can be extended to include process flow automation and work flow management, which introduces the capability to manage state and long running transactions. They are, inherently, asynchronous — in the sense that different application elements can be loosely coupled together.

Application brokers are, essentially, run time products. The focus is on the message and process flow, with little human interaction (until you reach full blooded work flow and its inclusion of people interactions).

But that is not to suggest that development is unnecessary: it is, except that development usually takes place externally to the application broker itself. In MQSeries Integrator, for example, nodes are built in C++ or Java. Working nodes are then assembled into flows on a palette which is then deployed onto the application broker run time engine.

A similar approach is delivered by e*Gate and e-Biz Integrator. In the latter, the rules and formats are created externally, and loaded into a database for execution.

Application brokers and servers are the same?

Ah ha, you may say. Does that not mean that application brokers and application servers accomplish the same only differently? The answer is 'Yes'. Both have run times engines. Both have development 'environments'.

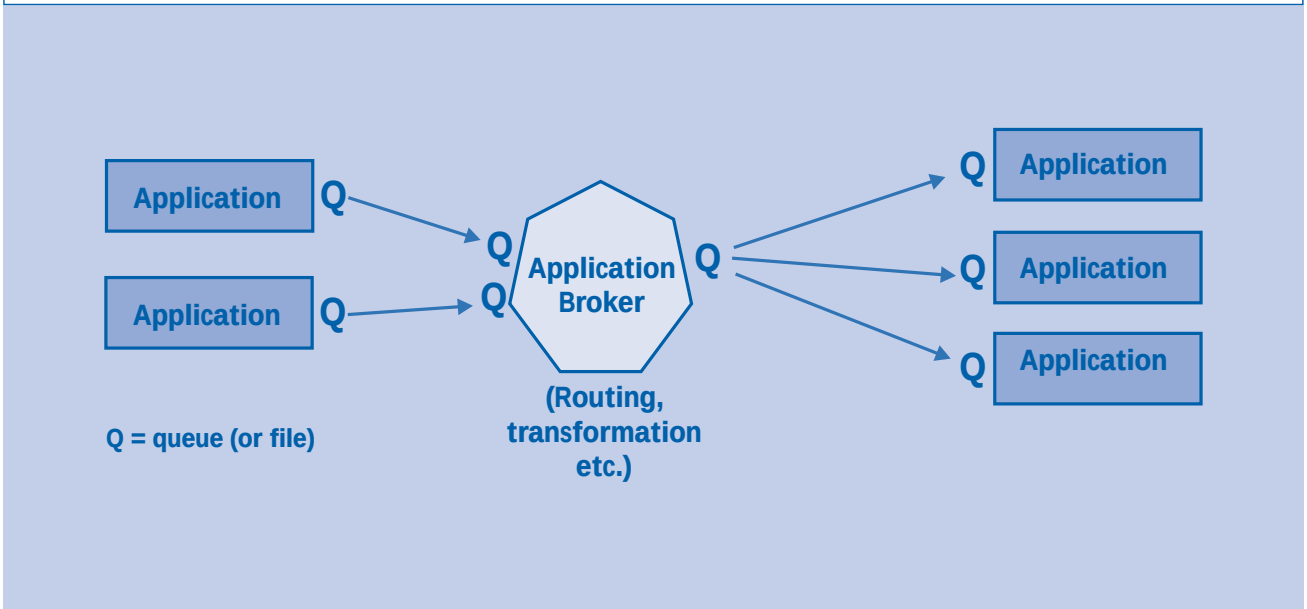
Indeed, the similarities go even further. You can use an application server to build what looks like an application broker. You can use an application broker, especially if process handling is added, to deliver what is to all intents and purposes an application server (a system running applications). Both can be used to provide application integration — which is what both are being sold to do.

Do the differences matter?

This raises the question of whether the differences — which are considerable in terms of approach, technical architecture and style — matter. We argue that the differences do matter. They probably matter most if you come from:

- **an application development background (when an application server will almost always look more understandable)**
- **a business process background, when the concept of an application broker will generally appeal more (working as it does between existing entities, applications).**

Figure 4.2: Application Broker — a typical configuration



If you come from neither background, the only prediction we will confidently make is that you will be confused — as you attempt to acquire both development and business process knowledge before sorting through the many possibilities.

Too many, too much

In one sense, therefore, IBM's initiative to place both its application broker and application server families under the one WebSphere name makes sense — if you know from where you are starting. If, on the other hand, you are new to the potential being offered, trying to sort out what you want and why will be a true challenge.

But perhaps the more meaningful, in architectural terms, is the approach taken by Sun. It has placed its application server (the run time) beneath its application broker. The application server is the engine on which the application broker function works.

While the Sun implementation is relatively unproven, it points to a simplification — of description, implementation and deployment — that we think is both desirable and clear:

- **the application server should be exactly that, a run time engine for applications (which could include application broking, process flow, work flow, etc.)**
- **the application broking (or other functions like process or work flow or whatever) is a specific implementation of application function which uses the underlying application server**
- **development tools, applicable to each functions, are kept separate**
- **transports, the mechanisms which bring payloads for processing, are also separate.**

This would be a much more attractive story for customers. It would be a more logical story for vendors. It would certainly simplify decision making. But it is not all. Now add PFA.

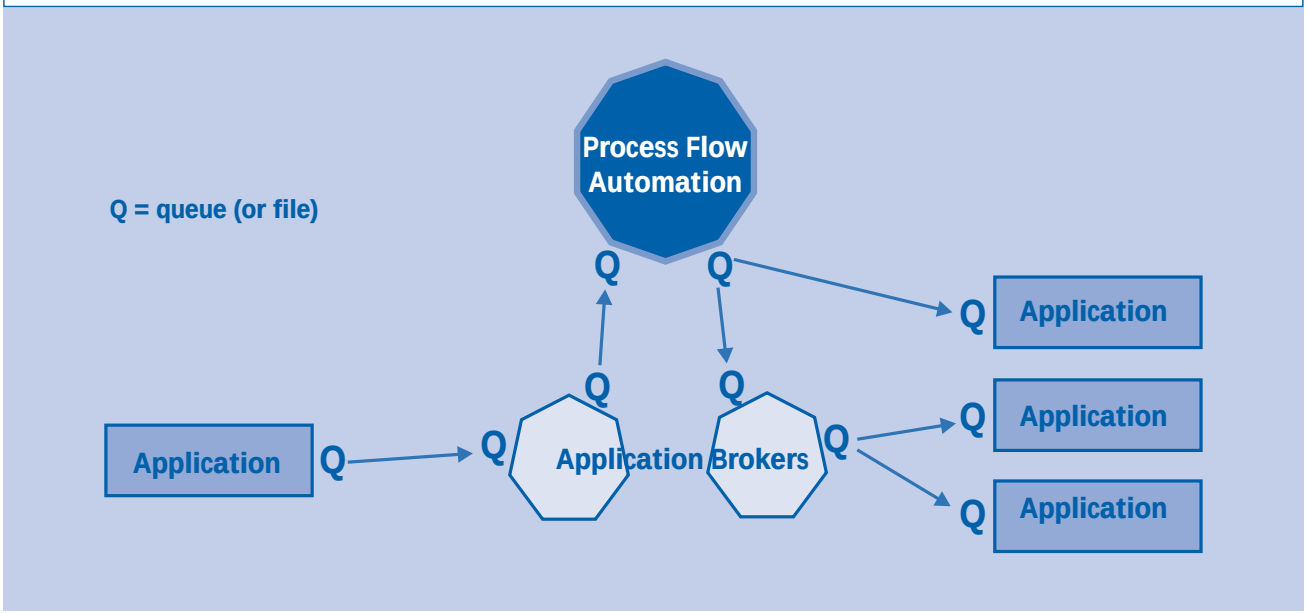
Process Flow Automation

As described above, both application servers and application brokers are relevant for integration, if in different ways. Traditionally, application brokers were associated with linking business applications inside an enterprise (intra-business application integration), which was often true for application servers as well.

With the arrival of the Internet, the need for linking selected business applications across the Internet (inter-business application integration or B2B) accelerated.

However, neither application servers nor application brokers are a total answer. They automate specific processes in

Figure 4.3: Process Flow Automation with an Application Broker



a relatively fixed manner. Once working, they run as needed.

In contrast, process flow automation (PFA) enables businesses to add an additional level of sophistication by which to direct whole business flows. To comprehend what PFA can do, think about two typical business processes:

- **a mortgage company making a house loan**
- **a business making a sale to another business over the Internet.**

In the first, the mortgage house will possess a formally defined process about how mortgage applications should be handled, from the customer completing the form through authorization through actual delivery of money to either the customer's account or the seller of the house. In practice, because of the way that systems have evolved, as islands of automation, it is likely that several different, and specialized, applications running on different existing physical systems will be used:

- **from the mortgage origination entry application**
- **through credit checking (which might involve external references to third party credit databases)**
- **through a payments system**
- **through to the mortgage house's own ledger system.**

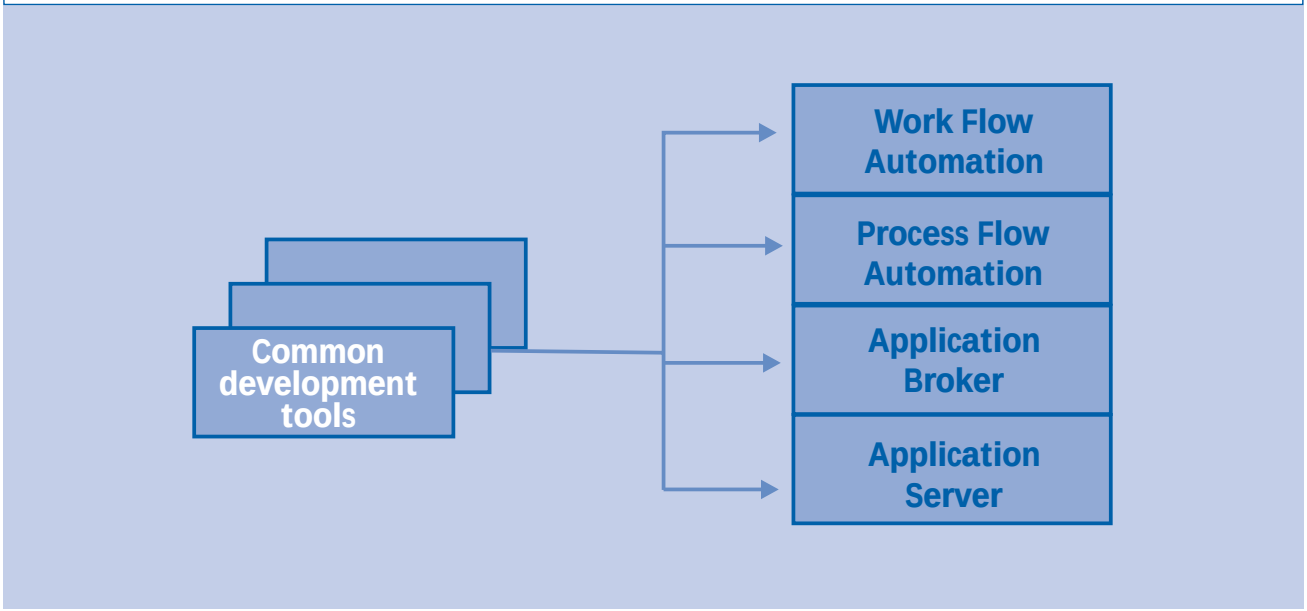
Using the traditional approach, even for the simplest mortgage, there is substantial human interaction — primarily supervising the application through the various sub-processes. Automating as much of this as possible has the potential to be faster and cheaper for the mortgage house (and customer), as well as require fewer people.

In the e-commerce instance, a business might have a Web site with an order form. A business customer decides what to buy (say) 1000 widgets. That order has now to be processed, and the widgets delivered. But to do this a sequence of processes — the order and delivery business process — has to be completed, from:

- **establishing whether the widgets are in stock (checking against inventory)**
- **making a credit check (possibly against external credit agencies)**
- **arranging for delivery (and notifying the customer)**
- **calculating discounts**
- **invoicing, once delivery is made**
- **posting the sales and general ledgers (which may be held on different systems)**
- **calculating the sales commissions to be paid.**

With the exception of the physical delivery, most of these processes could be automated. People involvement is not needed — if the various applications can work together.

Figure 4.4: Application Server + Application Broker + PFA + WFA



What PFA enables is the automation of a predictable sequence of events, but where different choices of actions are enabled — which can link two or more applications. This is instead of those events being triggered by one or more people.

In business terms, PFA is not the same as work flow automation (WFA). WFA originated with large paper-based organizations — like insurance companies and government — which looked for software to 'manage' the paper flows within their organizations. This frequently included long, complex processes — for example, the completion of an insurance claim, from submission right through to final payment or rejection.

In the terms of the mortgage application, or the sale of 1000 widgets example above, PFA can manage the process to process steps. Even more, it can manage (if sufficient attention is paid when it is set up) the automated catching and correction of errors or discrepancies. By reducing the involvement of people (where they are not really wanted or needed) processing can be further automated, and more accurately.

While PFA offers more than an application broker, it does not aspire to deliver the complexity (the human involvement) of WFA. The attraction of PFA is that it is:

- **simpler**
- **restricted to processes that can be automated without human involvement.**

PFA is, therefore, WFA without people and applied to processes or sub-processes which are already described in software. This is significantly easier to deliver than WFA. PFA bridges between processes that already exist in applications that are already running in organizations (or between organizations) — by applying a layer of logic which can direct and feed the applications with which it (PFA) has connections (Figure 4.3).

Layering a common sense solution

In some ways it (PFA) sits atop application servers and brokers and below the people-rich involvement of WFA. It does this by automating the co-ordination of activities between processes that are already 'captured' in existing or new applications and brokers without surfacing interactions to people (and all their associated complications).

In practical terms, think about the newly expanded WebSphere family or Sybase's portfolio — of Enterprise Application Server, e-Biz Integrator, Process Server and Portal Server. Envision a world where application brokers and PFA function are built using application development tools which produce function that can run on application servers. Go one stage further and appreciate the increased attraction of using those same development tools, which can be used for defining process and work flows. This would be coherent.

Most vendors' product stories would be more attractive if their application servers borrowed from the production, deployment and resiliency capabilities that application brokers possess. Similarly, most application brokers would benefit from being associated with modern run times, like those found in application servers and with commonly used development tools. And PFA would make even greater sense sitting above them both (Figure 4.4).

Management conclusion

This may be to ask too much. But, if vendors are bold enough, they might remove the fruitless 'angels on the head of a pin' debates about whether to buy an application server or application broker solution — when both accomplish the same broad objectives, only differently.

This would be a step forwards. Middleware is already complicated enough without having to wade through largely meaningless differences — that are mostly in the eye of the developer (vendor) rather than the end user.

The state of business rules

Martin West
Vice President of Research and Development
SpiritSoft

Management introduction

In this analysis, Martin West — the Vice President of Research and Development at SpiritSoft — reviews the evolution of business rules in the context of event driven systems. He introduces and positions reactive rules in a number of business areas, as expressed through the 'Event-Condition-Action' (ECA) paradigm.

In addition, Mr. West describes:

- ***the need for rules, by segmenting the problem domain into deductive rule processing and reactive rule processing***
- ***the concepts that underpin 'ECA' rules — and how these are instrumental in solving many of the business critical problems that arise in today's highly connected environment.***

Background

Up until the late 1980s, applications were generally procedural in nature. Business processes had to conform to the way the application that supported these processes was written.

A number of developments occurred towards the end of the 1980s that began to change the way applications could be written and operated:

- **Apple, IBM and Microsoft led the development of 'windows-based' systems; applications could now be written to include event driven user interfaces**
- **with the introduction of IBM's MQSeries, the development of message orientated middleware (MOM) came of age (it had existed before, but not really in self-standing product form)**
- **the introduction of triggering functions within database systems enabled applications to be called when data changed or was added to the database**
- **object orientated programming began to emerge from its evangelistic stage.**

However, none of these fundamentally altered the way that applications were written. The old, familiar procedural techniques were, and are, still fully in operation. Business rules were, and are, still deeply encoded in traditional applications.

As such, business rules were not visible to business people. To all intents and purposes, this meant that only programmers could change applications. But such changes were often costly as well as being prone to error.

Realization

By the 1980s there were those in the IT industry who recognized that these recurring problems were unacceptable — and that alternative solutions were required. One example of an effort to break free from traditional bounds was IBM's Insurance Application Architecture (IAA). This defined the concepts of business rules and business rule processing.

Since then, along with other developments, business rule technology has been following a typical 'technology adoption curve'. Today, IT professionals:

- **are seeing the emergence of business rules technology from the evangelical state**
- **can reasonably expect it to become 'matter of fact' in the next five years, in much the same way that object technology has been accepted over the past five years.**

Standards are, as so often, playing a critical role in its wider acceptance. JMS (Java Messaging System) and JAXM, in particular, offer definitions for the delivery of synchronous and asynchronous events. Equally, business related XML standards — such as ebXML and finXML — provide the semantics and choreography of industry specific business events and scenarios. Finally, Business Rules Standards are being defined by:

- **RuleML, which is part of the W3C Semantic Web Standards activities**
- **the Java Rule Engine activity, sponsored by Sun.**

In other words, rules have moved beyond the conceptualization stage. They are moving towards production.

Business rules

A business rule is a statement that defines or constrains some aspect of a business. It asserts structure or control which influences the behavior of the business. Traditionally, the dominant force in rules technology has involved 'deductive solutions'. These are rules that are inferred without the need to be told how to do this.

But this is not the only type of rule. As this analysis will describe, there is an alternative type of rule solution — known as a 'reactive' rule.

Today, many products support business rules. More often than not, however, products are specific to an industry.

The reason for this is that it has proven difficult to build a 'one size fits all' business rules system, not least because the terminology (and audiences) vary so differently between industries. For instance, an application that enables actuaries to define business rules for life and pension products has surprisingly little in common with that appropriate for the administration of a property and casualty system — and that is within the insurance industry. Between industry types, say, insurance and manufacturing or banking and pharmaceuticals, the differences are even greater.

Why do you need rules?

Business rules are, already, pervasive. They cannot be

avoided. Whether it is calculating the value of your life insurance premium or the cost of a mobile telephone call, it is a business rule which does the work. In this sense, business rules are the synthesis of one or more processes ('find out the number of minutes used, find out the cost per minute, multiply the two together and obtain an amount to be invoiced ...').

In the past, applications too often became unwieldy and inflexible as they grew or expanded. In so doing they began to fail to fulfill business needs. In addition, such concentration of logic and process created maintenance nightmares. These became ever worse as applications were modified in an attempt to address new business requirements.

The consequence was that organizations realized that they could not avoid the fundamental problem, which was that business requirements change faster than applications can either be created and/or modified. The attraction of rules is that they offer a way to:

- **encapsulate business semantics**
- **promote them to the surface in the same way that databases enable us to separate data from applications.**

We need rules so that we can withdraw the business semantics that currently are embedded in applications. In so doing we can provide improved support for the decision making processes that underpin the ways in which organizations operate. This is an essential requirement in today's operating environment where businesses need to remain flexible, whilst at the same time driving their IT infrastructure forward towards supporting complex real time decision making.

Having said all of that, business rules have not yet taken off — principally because of the lack of agreement on the standards and the technology that exists to implement the systems. However, this is changing as messaging, business process and business rule standards win ever greater acceptance and approval.

Different rules for different problems

Because rules enable us to capture business semantics as well as to support on-the-fly decision making, they clearly possess the potential to become part of IT's total offering. But not all rules, however, are created equal.

It is here I depart slightly from Barbara von Halle's view that all rules are about data, which she expresses in an excellent series of articles on Business Rule Systems. This was published recently in the DM Review (www.dmreview.com) . In my view, you should separate:

- **rules about data**
- **from rules about business events.**

Data-centric rules can be applied to large data sources to infer patterns and to answer questions. We might apply rules to an historical database to look for the reasons why something happened.

For instance, these might infer a relationship between two or more claims, and associate their risk, which then has an effect on the premium calculation for an insurance policy. These sorts of analytical problems are best solved using Rete-based deductive systems and data mining applications.

In contrast, business events form an essential part — the life blood — of any enterprise. Events codify the way in which a system responds, whether it is in procurement, insurance, finance, supply chain management or any one of the many forms of electronic data interchange that exist today.

For example, when an insurance product is sold, an insurance product administration system might generate a 'product sold event' which would be sent to the sales compensation system. The compensation system would process this event — according to its set of business rules; these rules might even generate additional events — that would eventually reach the payroll system (Figure 5.1).

Why business rules are different from rules about data

It is not sufficient to be able to expose the business rules and enable their rapid modification. A business rule system needs to support change management and version control. In this sense it differs from rules about data. Let me illustrate why.

Over time different versions of a rule will exist. A rule may have to be changed because the business itself has changed, or because the rule does not correctly model the business (a bug fix).

A further complication is that a rule may be part of a contractual agreement, say an insurance policy for instance. This may have an operational life of tens of years. Take the example of a sale person paid by the amount and longevity of a policy. He or she sells a policy, and the reward is a one-

off payment based on the assumption that the policy-holder continues to pay for ten years or more.

But what happens if the policy is canceled after six years? The sales person will have been overpaid. His or her compensation needs to be adjusted in terms of the compensation scheme in operation six years before. [Such long intervals are not that uncommon: just think of products with three or five year guarantees.]

So, in addition to the development version control and change management of business rules, there is an operational dimension as well. A business event will have a business date associated with it. When a rule is executed the business rule system must locate the appropriate version of the rule for that business date. Business rule systems today are immature in this respect.

ECA and rules

An event-centric rule is best expressed as a forward chaining set of Events, Conditions and Actions:

- **an event can be a message from a messaging service (say, JMS) (for example about a product sold or a 'request for quotation' message), a property change (a button press to enter a communication received) or a state transition (an insurance policy is about expire)**
- **conditions are predicates that are used to decide what actions should be executed; they are applied to the event (or events) that precede it and return true or false answers ('the beneficiary's age is within acceptable limits')**
- **actions represent what should be done; typically actions are built-in functions (sending a sales compensation update message or making a request for external communication to use a print service) or any method accessible to the rule.**

The benefit of a rule-based approach is that it develops and codifies the semantics of all business transactions that describe 'how things are done'. This reduces the cost of integration, training and operating a business over time.

At their simplest, ECA rules decouple application interfaces from business logic. They enable each to be optimized and, most critically, changed independently.

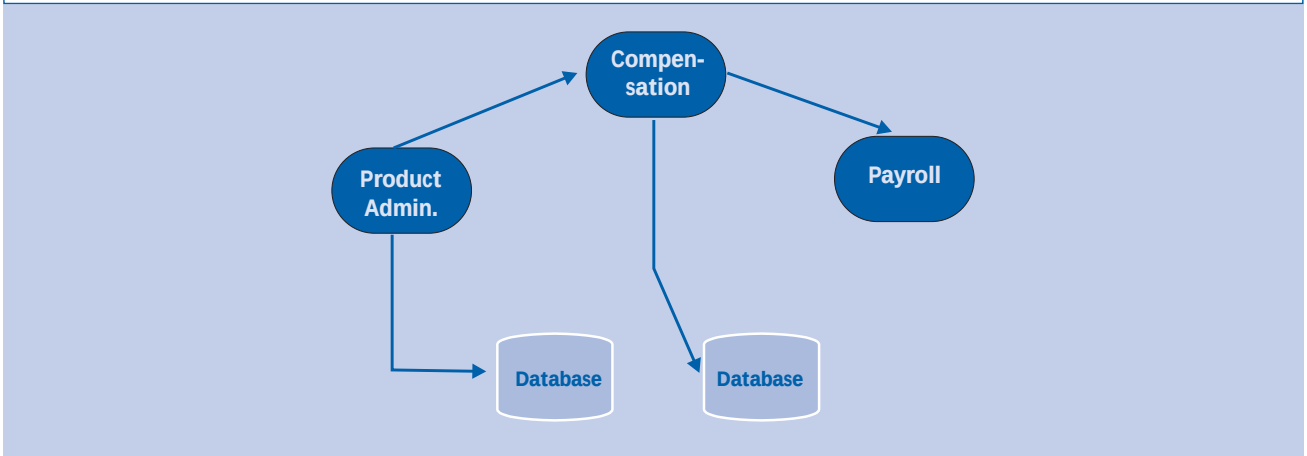
The ECA framework (multiple ECAs)

By looking at a simple conceptual model for describing re-active rules — in the form of Event, Condition and Action — it is possible to introduce ECA rules and an open ECA Framework. The advantage of this is that it forms the basis for collaboration and co-operation between:

- **different ECA rule implementations**
- **re-active rule and deductive rule implementations.**

In essence the framework enables the definition of business events — and the conditions and related actions that should be performed — to be expressed in an interchangeable format based on an XML schema. In the case of a sales compensation system:

Figure 5.1: Compensation and Payroll rules



- **the sale of a product would be a business event**
- **it would produce a result which sent another (new) business event to the payroll system.**

A Java-centric ECA Framework implementation is a set of interacting components that conform to the ECA Framework interface definitions, allowing multiple user interfaces to be constructed that fit the needs of the user community. The framework supports forward chaining ECA rules and, by so doing, produces implementations which are able to take a generative approach that, in a Java implementation, enables the rules to be generated (on-the-fly) into Java code. A C# based solution could conceivably do the same while still conforming to the ECA Framework interface definitions.

Components, embedded or stand-alone, are able to subscribe to rule changes delivered from a JMS message bus, and load them dynamically. For example, in the SpiritIntellect implementation with which I have been involved, a unique version-based class loader is used — without the need to stop the running rule.

In this way the framework is unique. It enables Java Virtual Machines to be used as rule engines, by exploiting a JMS compliant messaging bus for rule distribution as well as event delivery.

In this implementation, the ECA rules are geared to the problems inherent in event-based systems, the footprint is small and, because the rules are implemented as Java code, the rules can be embedded seamlessly into Java applications. This means that rules can be run on devices as small as PDAs — at least those that support Java Platform 2 Micro Edition (J2ME) with a Connected Device Configuration (CDC) — as well as on high end servers.

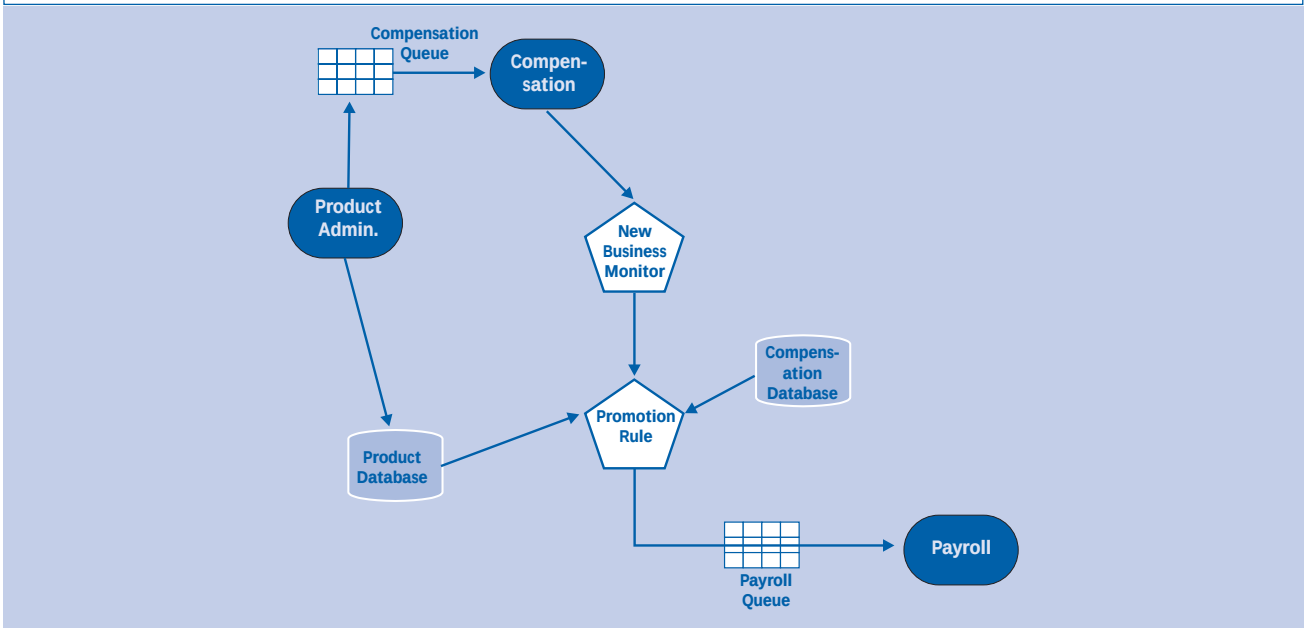
Rules can, therefore, be embedded into anything from a simple Java bean to an EJB session or entity bean. This makes ECA rules deployable in a wide range of environments and applications. It provides event-based flexibility wherever it is needed.

In practice

The sales person compensation system I referred to earlier could easily be implemented using an ECA framework. There would be a monitor to process incoming events that execute an appropriate business rule depending on the kind of the incoming event (Figure 5.2).

For example, the business could react to a new competitive product by introducing a sales promotion which offers the sales force an extra incentive to sell a particular product. To deliver this (without traditional coding), you would create a 'new business rule' which would in turn invoke a promo-

Figure 5.2: Compensation with ECA rules



tion rule whenever the product sold was the one being promoted.

This is an example of a short lifecycle rule. It could be implemented and deployed in a matter of days. It might exist only for a few weeks or months. Equally important, it could be discontinued — by removing the rule, with equivalent ease and speed.

At the opposite end of the spectrum, a life insurance premium payment event might trigger one or more rules that allocate portions of the premium to investment funds. The deployment and validation of this rule would be longer and more formal — because of the contractual nature of the business event.

In this instance, the rules might need to be operational for several years. Indeed, the rules might be upgraded over the lifetime of the investment agreement because of corrections, new requirements or regulatory changes. Whether subsequent payment events operate with the new version depends on the nature of the changes to the rule. One could even envision circumstances where meta-rules dictate which rules are applied to a particular business event and determine which rules should be used to police other rules.

Management conclusion

The technologies and standards for the successful development of business rules are just beginning to mature. The infrastructure to provide asynchronous messaging is well established and the JMS standard will increase the general acceptance of message oriented middleware.

XML standards — like ebXML and finXML — will enable message payloads to carry information against which ECA rules can be easily applied. Business rule standards will enable the development of componentized business rules, ones which can be assembled, shared and adapted by the wider community. Event driven execution, management and monitoring of business rules can be achieved in a truly flexible way to ensure that systems remain adaptive to business needs at all times.

As Mr. West has illustrated, ECA rules are a technology which enables the management and execution of business events. By providing an environment for the specification of complex rules which allow multiple events to participate in a business transaction, ECA rules can be constructed and deployed anywhere in a network.

Mr. West has also shown that:

- ***flexibility of deployment is an important requirement with the ability to run rules as stand-alone rule agents or as embedded rules within applications***
- ***an open ECA Framework exists — as a conceptual model that has the potential to allow interoperability of rule based solutions that includes both event and data driven rule technology***
- ***this obviates the need for a separate rule engine — thereby reducing the footprint of rules and ensuring an easier integration between the rules and their surrounding environment; it enables the choreography of existing components to be easily achieved.***

A reality check for .NET and J2EE Web Services

Mark Creamer
Consultant

Management introduction

More than a year ago, Microsoft announced its new vision for the future, the so-called .NET initiative. Broadly speaking, it was about three principle technology areas:

- ***first, there is common language runtime and C# ('C sharp')***
- ***second, there are a set of enterprise servers which are XML enabled***
- ***third, there are 'Web Services'.***

Each of the three principle areas has value independently of the others. Taken together, the technologies comprise a platform for distributed computing across enterprise boundaries.

Yet, of these, it is the Web Services area which has become the lead story — at least from an IT point of view. But it is not only Microsoft that has a story for Web Services; so do many other software vendors, from as diverse a collection as Sun, BEA, HP, Oracle, Sybase and many others — as Mark Creamer explores.

Web Services

In essence, Web Services are an old story with new provisions. Take established distributed component models (DCOM and CORBA), add support for Web protocols on an interface to a function and the browser is no longer the only potential client for HTTP-based services.

Web Services are significant because of the integration of the range of application components. These are intended to be implemented on different platforms using different programming models. Historically, however, such grand intentions required binding with proprietary middleware APIs, for example for messaging. Unfortunately, there has not been an industry accepted standard for wire-level messaging.

Now there is. Furthermore, that standard is within both .NET and the Java platform. It is the Simple Object Access Protocol (SOAP). It is SOAP that enables Web Services.

SOAP, and more

SOAP runs on top of HTTP, among other transports. This is crucial because HTTP is ubiquitous and avoids the need to punch undesirable holes in firewalls. A group which included IBM and Microsoft, two of the most significant forces in the software market (with products shipping that support Web Services today) authored SOAP.

In addition to SOAP, two other standards are fundamental to Web Services:

- **the Web Services Definition Language (WSDL)**
- **Universal Description, Discovery and Integration (UDDI).**

As with SOAP, these are XML-based. Unlike SOAP, they are not required to have a Web Service (although this is highly desirable).

WSDL is used to describe service capabilities and invocation procedures. It is roughly analogous to some early, and proprietary, EAI middleware connector functions.

UDDI is a registry specification. It is capable of storing WSDL, among other things.

Development

These are not the only standards relevant to the Web Services technology area. The others, however, are barely out of their discussion groups (so far). More important than these, by far, is the development tool set.

Development tools deserve attention immediately. Over past years, middleware — and platform suites in general — have focused mainly on the run time environment. Vendor answers to development questions were along the lines of:

- **'we support your favorite IDE'**
- **or 'we have an SDK you can use to create your own development tool set'.**

This was manageable, if barely. But it was certainly not desirable, or helpful, if you were planning a distributed systems development project or infrastructure.

In a Web Services scenario, where developers are in different organizations and potentially using hundreds of components, there is a real risk that scale itself becomes a problem. There must be support in development workbenches for:

- **data model aggregation**
- **automatic generation of interfaces**
- **discovery of services.**

In addition, there needs to be coherent (and consistent) support for team-based application component assembly. And there must be:

- **project management support**
- **independent budget monitoring**
- **accounting**
- **work flow management (for the whole engineering process).**

The back plane of the Web Services vision is the same as for distributed components in general — mass customization and rapid assembly of applications, albeit with a cross-firewall spin added. Given that, today, B2B trading, e-business and customer-facing applications represent the majority of new application initiatives, the battle among the major software vendors to win your business — as the Web Services platform of choice — has already begun. These three areas, by definition, are cross enterprise applications.

Needless to say, we are some years away from seeing all these concepts made available in development environments specifically for the purpose of supporting the peculiarities of Web Services projects. In this context, therefore, the Web Services vendors have yet to make a serious improvement on what their predecessors had failed to do.

.NET

The core product for .NET platform is Visual Studio.NET.

The focus, for Microsoft, is to support development of new, Internet-based, subscription services as a part of creative projects. One need only look as far as plans for Windows XP to observe that Microsoft sees applications becoming much more 'lively'.

While the Microsoft solution is intended for Windows only, at least as far as serving up the resulting services goes, many programming languages are supported for Web Services — and for consumption of them. In a time of scarce technical talent, this is a plus. Microsoft intends to be the first to offer a complete package that supports the Web Services concept.

Available now, or shipping this year, are:

- **development tools**
- **enterprise server software products**
- **a host of ready-to-use services (like an email Inbox, Wallet, Contacts, Filing, Device Properties and so forth)**
- **Microsoft applications, including Office, which are being Web Services-enabled.**

Sun

The Sun platform, ONE, is — as you might expect — based on J2EE. ONE is commercially oriented and offered primarily, although not exclusively, on the iPlanet Integration Servers.

J2EE — from multiple vendors, not just Sun — is the only alternative to the .NET/WinTel combination. That said, Sun's focus continues to be to try to prevent Microsoft from making inroads into the higher levels of computing.

Sun was later than most with its Web Services announcements. But it has picked up momentum recently, adding a Web Services Pack that includes XML-based Java APIs (JAX — Java APIs for XML) which is intended to foster integration between Web Services and J2EE such as:

- **JAXP (Processing), for DOM support, SAX, XSLT and schema processing support**
- **JAXM (Messaging)**
- **JAXB (Data Binding)**
- **JAXR (Registry)**
- **JAXRPC (Remote Procedure Calls), the mechanism for XML-based remote procedure calls in the Java programming language.**

Sun is taking the approach that developers need only write ordinary EJB components and the application server will do the rest. This is in stark contrast to the .NET approach, where Web Services support is found in the components themselves.

Sun's position on JAX and ebXML appear intended to slow adoption of Microsoft's C# and common language run time. However, the Sun story is not the only Web Services alternative in the Java market.

IBM

IBM has a strong Web Services platform for Java, aptly named the Web Services Architecture. Built around its WebSphere application server and WebSphere Studio tools (both IDE and downloadable SDKs), IBM offers a typically IBM-like infrastructure that supports all Web Services related standards — and not only the core ones (SOAP, WSDL and UDDI).

Unfortunately, 'typically IBM-like' means that the tools and installation are complex. This should not be surprising. In contrast to Microsoft, IBM's focus is on legacy integration with the Internet. This is readily explained: IBM has so many customers — with applications locked up behind firewalls in large data centers which are trying to escape — that any other approach would have alienated those customers. Indeed, many of these customers are also regular clients of the IBM Global Services division, for which Web Services offers a considerable upside in the way of future business.

The result is that IBM remains a clear choice whenever there are:

- **significant legacy integration requirements**
- **or complexity has to be addressed.**

On the down side, IBM:

- **is unlikely to appeal to those who want to start from scratch or to be 'lively'**
- **has committed to include Web Services support only in its Java/EJB tools and technologies.**

BEA

Another leading Web Services provider with good legacy capabilities is BEA. Its e-business platform is based on its WebLogic application server. BEA has recently announced Java Connector Architecture (JCA) support. JCA is significant because it permits EJBs to be populated from legacy artifacts, and EJBs are the likely components which will expose CORBA objects (and mainframe procedures) as Web Services.

BEA was later than one might expect with its Web Services announcements, having virtually nothing of significance to say until the second quarter of 2001. The JCA announcement is significant, given that WebLogic is the market leading J2EE certified application server.

Major vendors of packaged applications will be attracted to BEA because of this. It offers them a chance to portray a Web Services solution without cannibalizing their own monolithic suites and revenue models. JCA is the secret ingredient that makes the application server an integration solution. An integration platform that incorporates Web Services is a powerful construct.

If BEA has an Achilles heel in the Web Services arena, it lies in its lack of the equivalent, commercially accepted and comprehensive, development tools that Microsoft or IBM possess. Balancing this is that you can use most IDEs in association with BEA products. But, as discussed earlier, it appears that Web Services adds a sufficient number of distinctive requirements that most customers prefer tightly integrated Web Services development support to be available from their Web Services supplier.

HP

HP originally articulated the vision of Web Services — in 1999, as part of its then proprietary e-Speak architecture. Today, its *netaction* suite — based on the acquired Bluestone application server — is shipping. This is still based on the original e-Speak technology to some degree, and consequently is not as standards-based as many of the other competing vendors who started down the Web Services route that much later.

The HP architecture calls for SOAP and UDDI support, but the product today is simply XML-based. One benefit is that *netaction* supports C and C++ — in addition to Java. It can be used on Windows NT in addition to HP-UX and Linux. Furthermore, HP has added renewed life to HP Process Manager (the former Changengine product) — by casting this into the realm of Web Services.

Having made all these points in favor, HP does not have a software orientation or sympathy. One consequence is that it lacks the tradition of developer loyalty to its tools which Microsoft, IBM and BEA possess — which must limit HP's market appeal in the short and medium term.

Other players

Other well known players have announced their own, and different, Web Services initiatives. These include:

- **Borland**
- **Iona**
- **Oracle**
- **SilverStream**
- **Software AG**
- **Sybase.**

In addition a number of startups — like Asera, Bowstreet, Eltegra and WebCollage — continue to enter the market to exploit gaps and grab mind share. They do not, however, have hardware, operating systems or legacy software technology businesses to prop up. They are, thus, free to push the envelope with new concepts. On the other hand, few of these recent entrants have much in the way of revenues, which is not the case for Microsoft, IBM, BEA, Oracle, Sun, Sybase, etc.

Catches

Given all the hype and commitments from vendors of every ilk, there must be some catches. Do not expect any vendor at such an early date to have a complete architecture filled with all the constructs you need, or even a sufficient number to provide interoperability (see also page 18). A substantial investment in pre-purchase due diligence is prudent, if not mandatory.

We have had standards-based products for years, or so it has seemed. But remember how long it took for CORBA implementations of 'one specification' to become interoperable? This was not on account of pre-IIOP CORBA; it was due to the fact that vendors are competitive and the only air-tight specification is running code.

Three potential pitfalls or issues, which Web Services face, illustrate the difficulties:

- **UDDI pervasiveness: UDDI is a community process with over 200 members which is not likely to settle down and offer a tight specification anytime soon; people will have to start UDDI activity inside a company as a complement to DNS and LDAP, then move towards use within small, trusted, B2B communities before UDDI 'brokers' emerge — and you will have to decide if you want their content; balancing this, there is considerable potential for commercial exploitation (as per Yahoo)**
- **the volume of security/authentication requests skyrocket as finer grained functionality makes its way onto the Internet (PKI takes on a new level of importance, SAML (security profile exchange), XKMS (key management) and**

XTASS (authorization) are all under development); while security requirements are more complex (and certain to be based on policy management eventually), the ramifications of spurious purchase orders and wire transfers are worrying — EDI VANs or networks like S.W.I.F.T will be worth understanding for their lessons learned

- **service levels across a complex ecosystem of nodes on an inherently flaky Internet remain problematic; systems management vendors have been noticeably silent when it comes to Web Services — yet they are needed, not simply for use of Web Services in their platforms but to support failover, load balancing, alerting and instrumentation in Web Services enabled applications.**

Can .NET and/or J2EE stand up to Web Services requirements and expectations?

.NET can stand the load for smaller, non-critical projects today even though Microsoft has had to tune its architecture, and tools, more than J2EE needs to do. The tight integration of ASP/IIS/MTS/MSMQ and VB/VC/VJ — all basically COM+ centered — is producing a tightly coupled paradigm on top of (some argue within) the operating system itself; this is proving a wrenching challenge.

Yet history suggests that Microsoft is up to it. Microsoft's capability to deliver tools and infrastructure middleware is proven (if not often on time). This suggests that .NET will be a viable platform for serious projects in 2002 (it claims 2001, but that looks dubious).

The use of .NET will likely remain where Microsoft has always been strongest — in small to medium scale deployments or until proven otherwise. This is also where ease of use and fast development cycles are important. Host Integration Server will also play a pivotal role. This is Microsoft's entry point for its Web Services to link to legacy applications on legacy platforms.

J2EE would be a disappointment if not for IBM. A strong developer channel (a clear Microsoft strength) is needed for Web Services to fly. Where Sun has yet to tread, IBM has already trodden.

IBM already has a strong developer constituency, although this — with its legacy heritage — is very different to that owned by Microsoft. Despite this, it was IBM and Microsoft working together that produced several key standards for Web Services. Both IBM and Microsoft are committed to SOAP (along with many other vendors — with Sun only recently joining up).

IBM also supports all the key standards required for Web Services, though not always via J2EE directly. IBM has EAI middleware (MQSI/WebSphere Business Integrator) as well as wrappers for enterprise-level applications on CICS, IMS and Tuxedo.

BEA, with the marketing leading application server (in WebLogic), has stated its intent to implement JCA. This, potentially, will offer an alternative gateway for Web Services to link to legacy environments and commercial application packages.

Most other ISVs supporting J2EE have announced Web Services strategies for their product lines. Some platform vendors, like IONA with its iPortal XMLBus, have been shipping for some months. On this basis J2EE products for Web Services are ready for consideration.

Yet careful evaluation of product choices is still necessary since — with the clear exception of IBM — there really is no one-stop, multi-platform alternative yet. This is particularly true when comparatively large implementations, with many concurrent users, are the objective.

Management conclusion

A few months ago it appeared that Sun might fragment the Web Services initiative by supporting an alternative to SOAP (ebXML transport). However, Sun's J2EE application server partners were already shipping code that delivers Web Services on their application servers — using SOAP. Inevitably Sun had to support SOAP or see ONE shrivel — and arch rivals Microsoft and IBM walk away with the riches. BEA, IBM, Sybase, Oracle and IONA are among the other J2EE front-runners which, coming from a proven middleware heritage, provide decent offerings.

Microsoft is still ahead in terms of comprehensiveness — offering more tools, more discreet server products and more architectural skin in the Web Services game. But it is still a Windows-only scenario for developing and deploying your Web Services. Recent Linux porting announcements to the contrary, Microsoft has in the past made agreements (for example with Software AG) for large parts of Windows and DCOM/COM+ to be made available as third party platforms. None of these has had significant commercial impact on market acceptance or breadth of support.

While there are a number of specialists who have entered

the software market specifically to address Web Services, making a commercial success of distributing Internet-based components remains an unproven market — even though, logically, there is a demand for business processes, portals, and complementary tools, etc. Tracking partnership agreements with other vendors (especially the larger ones) will likely indicate which — if any — of these will survive.

That said, Web Services is no longer likely to be a passing fad — any more than distributed components are such a fad. Web Services are a natural evolutionary step for taking applications from behind the firewall to going through the firewall for business to business connections. On this basis, organizations should be preparing to select and experiment with Web Services development tools now — in preparation for tactical deployments of such projects in 2002.

In this context, first efforts should be directed inside the organization — where matters can be controlled. Subsequent steps should be through the firewall, in non-critical application scenarios.

The absence of standards for transaction management, security and service level management (for example, 'eventing' and management consoles) should not stand in the way. Opportunistic software vendors will plug gaps with solutions in due course.

On the other hand:

- ***strive to avoid lock-in to peripheral services***
- ***understand that the core to be protected is the fundamental Web Service itself***
- ***beware of Web Services components available on the market that are not based on SOAP, WSDL and UDDI.***

Even though useful and almost always attractive, proprietary Web Service components will defeat the intent of the 'open' vision. You will only be able to use them inside organizations deploying the same proprietary technology. But is that not where we started? Is that not what Web Services are trying to avoid?

Model Driven Architecture

Tom Welsh
Consultant

Management introduction

In March 2001 the Object Management Group (OMG) made what could be the most important announcement in its 12-year history. Unfortunately, the revolutionary nature of the new Model Driven Architecture (MDA) was thoroughly obscured by the statement's abstractions:

"In response to the growing and ever present challenge of enterprise interoperability," read the statement, "the MDA offers a full-lifecycle approach to solving the problems of developing, deploying and integrating existing distributed systems with emerging technology, assembling virtual enterprises that span multiple companies, implementing business intelligence solutions and enterprise information portals in a multi-vendor environment".

The automatic generation of application code from detailed models is an old dream. It goes back to the early days of Information Engineering,. It received a crushing setback with the collapse of IBM's AD/Cycle strategy in the early 1990s.

The question is whether software technology has now evolved to a point where MDA is a practical possibility. Surprisingly, as Tom Welsh explores, the answer appears to be a qualified 'Yes'.

Interoperability

Ever since its inception in 1989, the OMG has been dedicated to interoperability. For the past decade its slogans have been:

- **there will not be consensus on hardware platforms**
- **there will not be consensus on operating systems**
- **there will not be consensus on network protocols**
- **there will not be consensus on programming languages**
- **there must, therefore, be consensus on interfaces and interoperability.**

To this it has added one more affirmation. There will not be consensus on middleware platforms and component models.

Rather than trying to make disparate applications interoperable by imposing a standard format for the interfaces between them, however, the OMG has now become still more ambitious. It plans to make applications interoperable by deriving them from common design models — in its abstract technical terms, "distilling out middleware-independent semantics".

MDA specifications are based on a platform independent model (PIM) of business functions. The idea is that a single platform independent model can be translated to yield platform-specific models (PSMs) for a variety of middleware platforms — which in turn hide most of the underlying operating system and network details.

According to OMG, the benefits of MDA will include:

- **reduced costs throughout the application lifecycle**
- **reduced development time for new applications**
- **improved application quality**
- **increased return on technology investments**
- **rapid inclusion of emerging technology into existing systems.**

The chequered history of modeling

Ever since programs were first written, the programmer had to have both:

- **a clear overview of the problem to be solved**
- **the logical steps leading to a solution.**

These correspond, roughly speaking, to analysis and design. Analysis normally denotes the process of formulating the problem to be solved, without any indication of the methods to be used. Design follows on from analysis, and entails working out (in as much detail as necessary) exactly how the given problem can best be solved.

When problems were relatively simple, analysis and design could mostly be done with pencil and paper — or even in the programmer's head. Once the era of 'throwaway' programs had passed, however, it came to be recognized that software is a valuable asset.

But it is also perishable. Without someone who can understand its design, software cannot safely be modified. It then loses its most important characteristic. Complete, easily understood and up to date analysis and design documents are a vital safeguard against this eventuality.

In the late 1970s and 1980s, the principles of structured analysis and design were laid down by a number of 'methodologists'. By the mid-1980s, a few pioneers were starting to talk about object oriented analysis and design (OOAD).

Although each method differed from the others in details of notation, there was much common ground. Essentially, the idea was to draw standardized pictures of:

- **data structures**
- **data flows**
- **procedures**
- **and the like.**

If analysis and design could be carried out at this level, it made it much easier for everyone involved to participate. This included end users and project sponsors as well as systems analysts and programmers.

One serious problem that arose, however, was the diversity (and multiplicity) of methods. No consensus emerged, partly because new ideas were succeeding each other so rapidly.

Workstations and the CASE connection

The consequent fragmentation helped to constrain the use of analysis and design methods to a small minority. Instead, most programmers went on writing code straight from the requirements, sometimes with the help of flowcharts at least until Apple Macintoshes and IBM-compatible PCs started to appear on desks (soon to be followed by power-

ful UNIX workstations). All three delivered graphical software analysis and design as attractive applications, and so-called CASE tools soon proliferated.

The challenge was, nevertheless, to generate source code automatically from the design models. This idea became associated with Information Engineering and — according to James Martin, its best-known advocate — Information Engineering is "[t]he application of an interlocking set of formal techniques for the planning, analysis, design and construction of information systems on an enterprise-wide basis or across a major sector of the enterprise".

What could be called the 'first wave' of modeling technology culminated in IBM's AD/Cycle ambitions, which envisaged automated generation of applications. CASE tools from 'best of breed' vendors were to interface with IBM's Repository — which would be the central co-ordination point for all analysis and design metadata.

It was a great idea. Unfortunately it collapsed on account of several glaring weaknesses:

- **the process of agreeing standards and rolling out compliant products took too long**
- **AD/Cycle did not cater for emerging or non-IBM platforms (such as UNIX and Windows)**
- **development Repositories turned out not to be the 'modest extensions' of known database technologies that had been expected**
- **the whole apparatus, on top of everything else, was too complicated and too expensive.**

In retrospect, AD/Cycle — and, to be fair — Information Engineering shared one questionable assumption in particular. This was that modeling should be done on an enterprise-wide basis. The preparation of enterprise information models was a virtually impossible task for an organization of any reasonable size, unless it froze itself in time (not an action that most businesses were or are prepared to contemplate).

OOAD arises

In the early 1990s OOAD methods began to predominate. Again there were many of them, arguably too many.

The critical breakthrough came in 1996-1997, when three of the leading methodologists (Grady Booch, Ivar Jacobson and James Rumbaugh) agreed to merge their ideas. Subsequently their employer, Rational, submitted the resulting method to the OMG, calling it the Unified Modeling Language (UML).

While imperfect, like all compromises, UML was accepted by large numbers in the IT community — vendors and developers alike. For the first time the IT industry had a single, standard analysis and design method. By 1999-2000 all important modeling tools supported UML, allowing prospective purchasers to compare other aspects — such as:

- **reliability**
- **completeness**
- **performance**
- **price.**

OMG, MOF, UML and XML

At the same time, OMG members invested much time trying to tease out the implications of UML. For a start, they considered whether other meta models — in other words, modeling methods — might be needed in future.

To ensure their compatibility with UML, OMG adopted the Meta Object Facility (MOF). This provides a standard 'meta meta model' — a framework within which any number of consistent meta models like UML can be created and worked out.

Next on the agenda was a standard format for communication between modeling tools, repositories and other products that make use of meta data. Given the popularity (and power) of the eXtensible Markup Language (XML), this was a natural choice. The outcome was XMI Metadata Interchange (XMI), which encapsulates metadata within XML documents.

The Common Warehouse Meta model (CWM) was created using MOF and XMI — to provide a standard means of exchanging metadata which described data warehouses. This was extended, eventually, to embrace relational, multi-dimensional and record-based data definitions, followed by more and more types of meta data.

In September 2000 the Meta Data Coalition (MDC), to which Microsoft had entrusted the development of its Open Information Model (OIM), voted to merge with OMG. This decision left CWM, like UML, as the only broad-based industry accepted initiative in its space. [MOF, XMI and CWM are essential for MDA but they are not described further in this analysis — primarily for reasons of space.]

The architecture

The OMG has published a diagram representing the MDA (Figure 7.1). This picture accomplishes several things:

- **it replaces the older Object Management Architecture (OMA) diagram (Figure 7.2); this does not mean that the OMA is no longer valid, or that CORBA is being relegated to the background — rather that the OMG is building a new layer of abstraction which goes beyond CORBA**
- **the MDA diagram becomes an official symbol of the OMG's work**
- **the diagram can be used to explain some of the basic properties of the MDA.**

At the center of the diagram is the MDA itself, in close association with UML, MOF and CWM. This is the world of platform independent models. In the next ring are all the middleware environments for which platform-specific models can be supported. Currently, the list includes CORBA, XMI/XML, .NET, Java and Web Services but any other environment could be added, in principle.

The third ring features a partial list of pervasive services that can be standardized through the MDA — directory, transactions, events and security, to name but a few. Lastly, the 'spears' projecting from the sides of the diagram denote the numerous vertical industry domains within which the MDA can be of value.

On the face of it, creating a platform independent model is fairly easy. All that is required is to construct a UML model representing the problem to be solved, in such a way that no assumptions are made about what hardware or software will be used to implement the model.

The OMG has placed no limit on how many levels of platform independent models can exist (it is simplest to assume only one). However some profiles — such as the EDOC Profile for UML described in the following section on specifications — may require more. There may, for instance, be a business model and a platform independent component view. Having completed the business model, that would be mapped to a component view by selecting appropriate business components.

The next step (Figure 7.3) illustrates the heart of MDA. By applying specific mappings to the platform independent model, it is translated into one or more platform specific models for particular middleware platforms — for example:

- **CORBA**
- **Enterprise Java Beans (EJB)**
- **the Simple Object Access Protocol (SOAP).**

From independent to specific

At this point some interesting points of detail emerge. The mappings used to convert a platform independent model into a given platform specific model will, in the first

Figure 7.1: Model Driven Architecture (Source: OMG)

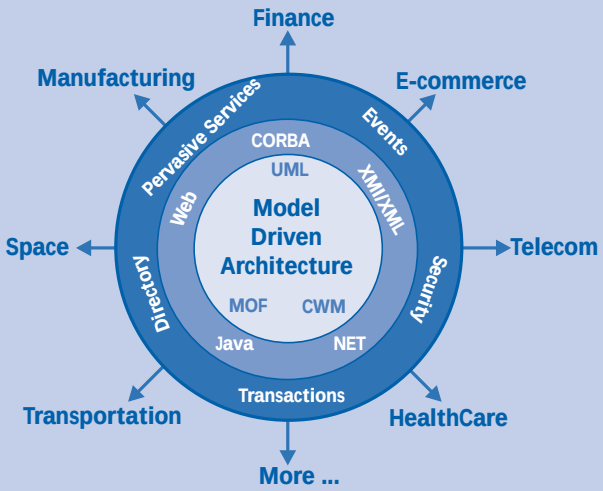


Figure 7.2: Object Management Architecture

The Object Management Architecture (from the OMG — the Object Management Group) envisioned a complete distributed environment. The foundation is built upon the concept of an Object Request Broker which:

- **manages communications between the OMA's components**
- **provides the basis by which objects interact in a heterogeneous and distributed environment.**

The desire is that objects should be independent of the platforms on which objects sit. In performing its task an Object Request Broker (ORB) relies on the OMG's definition of Object Services which, in turn, are responsible for:

- **creating objects**
- **providing access control**
- **keeping track of relocated objects**
- **etc.**

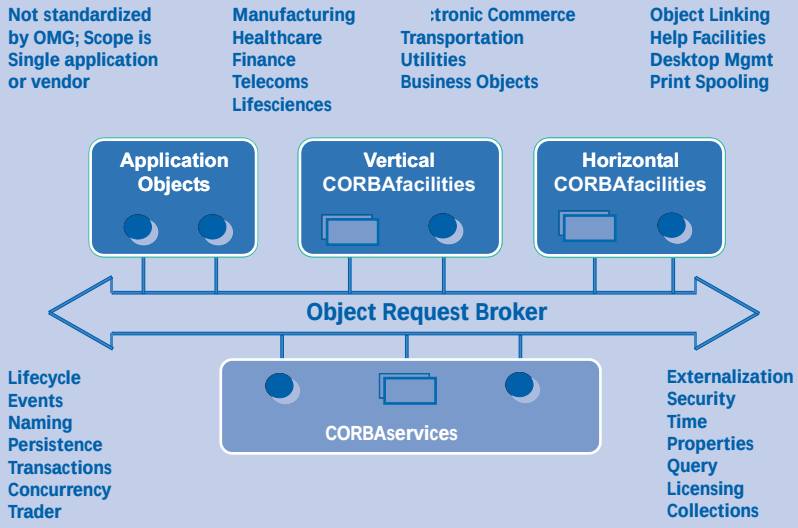
Common Facilities and Application Objects are the components closest to the end user. Their functions invoke services of the system components.

The Object Model defines common object semantics for specifying the externally visible characteristics of objects in an implementation-independent way. In this model, clients request services from objects through a well-defined interface. This interface is specified in the OMG's Interface Definition Language. Clients access an object by issuing a request to the object. The request is an event, and it carries information including:

- **an operation**
- **the object reference of the service provider**
- **parameters (if any).**

The Figure below shows the main components of the ORB architecture and its interconnections. The central component of CORBA is the Object Request Broker (ORB). It encompasses all of the communication infrastructure necessary to identify and locate objects, handle connection management and deliver data. In general, the ORB is not required to be a single component; it is defined by its interfaces. The ORB Core is the most crucial part of the Object Request Broker; it is responsible for communication of requests.

The basic functionality provided by an ORB consists of passing the requests from clients to the object implementations on which they are invoked. In order to make a request, the client can communicate with the ORB Core through the IDL stub or through the Dynamic Invocation Interface (DII). The stub represents the mapping between the language of implementation of the client and the ORB core. Thus the client can be written in any language as long as the implementation of the ORB supports this mapping. The ORB Core then transfers the request to the object implementation which receives the request as an up-call through either an IDL skeleton, or a dynamic skeleton.



instance, be written by experts in the target platform. Normally, quite large pieces of work will turn out to lend themselves to automation — for instance, setting up EJBs or when to use entity beans rather than session beans. Eventually, some mappings may be completely automated (or nearly so).

Also, provided that the platform independent model is detailed enough, there will be no need to make changes to the platform specific model, or the code that is generated from the platform specific model. Instead, all maintenance and application changes are performed at the platform independent model level.

The prospective benefits are enormous. Testing can be carried out at the earliest possible stage, saving huge amounts of rework. In addition, all the platform specific models derived from a single platform independent model — and all the applications generated from the platform specific models — should:

- **be functionally identical**
- **eliminate a whole class of software defect, especially those arising from the subtle inconsistencies that arise between different versions of a program.**

The next step

Th next step is generating source code from the platform specific models (Figure 7.4). This may not be as hard as would be expected — for a number of present-day modeling tools are already capable of generating code from UML models for CORBA, EJB and other types of distributed application.

The advantages of the MDA approach are now obvious:

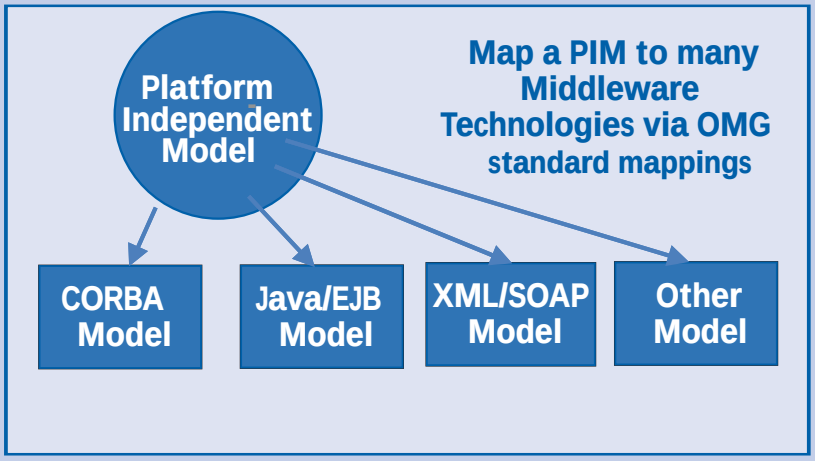
- **applications and data models need to be created once only, and can then be centrally maintained or enhanced with a minimum of additional effort**
- **changes to the core business logic (and data) are made in one place only, and are then automatically propagated to all deployed applications that rely on that logic**
- **once written, a single application can be semi-automatically deployed to CORBA, .NET, Java 2 Enterprise Edition (J2EE) and other environments.**

All deployed applications based on a single platform independent model are guaranteed now to share a single, consistent version of the business logic and data. Among other advantages, this means that they should interoperate.

Specifications and current UML evolutions

When trying to come to grips with the MDA, it is easy to

Figure 7.3: Mapping a PIM to multiple PSMs



feel that (as Gertrude Stein said about Oakland) "there's no 'there' there". At present about a dozen OMG specifications underpin the MDA — there will be many more in future — but not a single one of these was written with the MDA specifically in mind. That is because they are all responses to Requests for Proposal (RFPs) that were issued before the MDA announcement in March 2001.

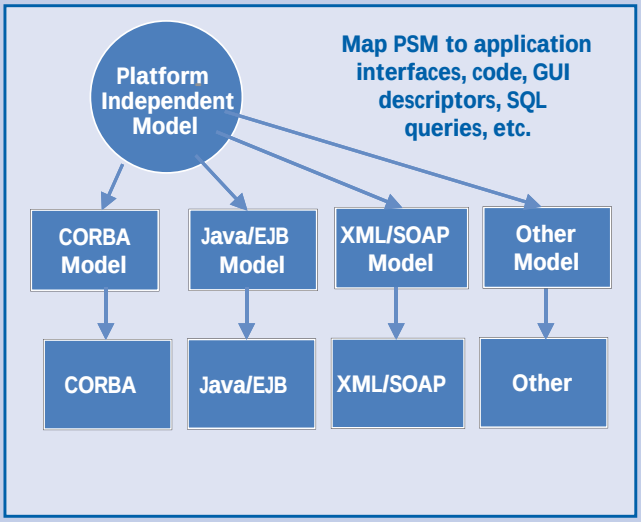
It cannot be over-emphasized that the MDA is a logical extension of the OMG's previous work. Indeed, it can be seen as a 're-factoring' of much the OMG has attempted, since the adoption of the UML specification in 1997.

- **diagram interchange.**

Work has been going on for some time in these areas — and is expected to continue throughout 2002. Support for the Model Driven Architecture is now one of the project's highest priorities.

UML evolved from a variety of methods, many of which had distinctly data-oriented pedigrees. So it should not be a surprise that it does an excellent job of describing data structures but is not 'computationally complete' — in other words, there are (many) programs that cannot be expressed in UML.

Figure 7.4: Generating source code from multiple PSMs



Other than press releases, presentations and an introductory white paper, the only document that concerns itself exclusively with MDA is the 31-page MDA Definition (OMG reference ormsc/2001-07-01). Written by the Architecture Board's Object Reference Model Sub-Committee (ORMSC), this is the authoritative account of what MDA is and is not.

UML has gone through several minor revisions since being adopted as a specification in 1997. Now it is going through a much larger set of changes which will eventually result in UML 2.0. In all, 37 letters of intent were received from organizations wishing to contribute their ideas, and UML 2.0 has been split into four pieces:

- **infrastructure**
- **superstructure**
- **Object Constraint Language (OCL)**

This is where Action Semantics are introduced. The submission on this, currently being evaluated, replaces all the UML constructs that relate to actions and computational behavior. It will allow the construction of both platform independent models as well as platform specific models — with platform specific models that can be executed (by special interpreters) to test their validity and completeness.

In addition, another submission, the Human Usable Textual Notation (HUTN), describes how a text language can be automatically generated from a MOF model. The objective is a mechanism by which developers and domain experts may read the contents of MOF models with relative ease.

UML Profiles

Throughout, the OMG has tried to keep UML itself as sim-

ple and uncluttered as possible. It has accomplished this by defining overlapping extensions which can be used when necessary. Such an extension is called a UML Profile, and consists of a subset of UML (which may be the whole) together with additional constraints, standard elements and semantics.

The first profile to be adopted (unsurprisingly) — and the only one so far — is the UML Profile for CORBA, which provides a way of expressing the semantics of CORBA IDL in UML notation. The specification is based on Rational's "Rose CORBA" (part of the Rose98i Enterprise Suite).

There are, however, other profiles currently working their way through the system, including:

- **a UML Profile for Enterprise Distributed Object Computing (EDOC)**
- **a UML Profile for Enterprise Application Integration (EAI)**
- **a UML Profile for Schedulability, Performance and Time.**

The EDOC profile is a massive specification, 447 pages long, which builds on the International Standards Organization (ISO) Reference Model of Open Distributed Computing (RM-ODP). Among other things, it defines mappings to EJB, Java and the Flow Composition Model (FCM). Profiles and mappings for the CORBA Component Model (CCM) and Web Services are planned.

The EAI profile addresses loosely-coupled distribution through products like:

- **IBM's WebSphere MQ Integrator**
- **Java Message Service (JMS)**
- **the C, C++ COBOL and PL/I languages.**

The Schedulability profile defines UML extensions for real time and embedded systems. This will include Quality of Service (QoS) concepts.

In July 2001 the Java Community Process (JCP) announced the completion of its UML Profile for EJB. What is notable about this is that it is the first such specification to be created outside the auspices of the OMG. It is not however, likely to be a diversion: most of the 11 companies primarily responsible for the work are also members of OMG.

Software Process Engineering Meta model (SPEM)

Unlike most OMG specifications, SPEM relates to the software development process. It deals with collaboration between roles that perform activities to create work products. Like UML and CWM, SPEM is defined as a metamodel; it is also a UML profile.

SPEM supports:

- **the Rational Unified Process (RUP), arguably the closest thing to a standard development process that exists today**
- **the DMR Macroscope**
- **IBM's Global Services Method.**

Future specifications

Two important kinds of MDA specifications are anticipated: Pervasive Services and Domain Facilities. These correspond quite closely to the present-day CORBAservices and CORBAfacilities.

Pervasive Services will include important optional pieces of functionality such as Directory, Security, Transactions and Persistence. If these can be defined once and for all at the platform independent model level, a great deal of time and effort can be saved (and scope for error avoided) at the specific implementation level.

Domain Facilities refer to vertical industry models, previously applicable only to CORBA, which under MDA will become available for EJB, Web Services, .NET and other important forms of middleware. The Healthcare Resource Access Decision Facility has already been implemented in Java and EJB as well as CORBA; other domain task forces are expected to follow suit.

Market implications

Over a dozen vendors (and several other organizations) have already committed to support the MDA. Several of these vendors are already marketing products that at least resemble the MDA in various ways.

Obviously, no product can yet comply fully with the MDA, as the specifications are not finished. It is, however, worth mentioning:

- **Adaptive, whose repository is MOF-compatible**
- **Interactive Objects Software; its ArcStyler tool closely mirrors MDA's lifecycle**
- **Kabira Technologies**
- **Rosch Consulting**
- **Secant**
- **Softeam.**

Project Technology and Kennedy Carter already offer modeling and code generation tools based on the methods worked out by Sally Shlaer and Stephen Mellor of Project Technology. These products perhaps come closest to the spirit of the MDA, although they were designed several years ago and omit the important platform specific model stage. They generate code directly from the platform independent model.

Although no announcements have yet been made, it is believed that some of the biggest players participating in the OMG — such as HP, IBM, Oracle and Sun — are working quietly to support the MDA in their product lines. The adoption of the MDA industry-wide seems the more likely because the new architecture is so firmly based on UML and MOF. As such, it represents an evolutionary rather than a step change.

Is the MDA good?

Clearly the OMG thinks so, as it did of CORBA. According to the OMG's technical director Andrew Watson, there are two fundamental business arguments in favor of the MDA — future proofing and the ability to roll out new forms of middleware without effort and disruption.

Future proofing matters. Today we can ask: 'five years ago, did Java or XML figure in your plans?' Of course not. What will occur in the next five years that you cannot possibly predict now? Lots. Being capable of accepting change is ever more important. The opportunity is to avoid, or reduce, 'technology dissonance'.

The ability to roll out new forms of middleware without effort and disruption will appeal to CTOs and IT directors. It has more to do with delivery than development. It does not take great understanding to accept that Java or XML or .NET are not the last waves of innovation that we will see.

It is high time that there was a standard notation for writing down business logic and data structures in a platform-independent way. After all, business logic is coming to be recognized as an important form of intellectual property. Yet, in most organizations, it is embedded in reams of incomprehensible low-level code.

Without hours of effort, even qualified programmers cannot figure it out. And every time a new system has to be deployed, or a merger or take-over or other business event imposes a new platform, everything has to be done over again. These are sufficient reasons to justify the MDA, before even considering the development impact.

Management conclusion

Ten years ago 'platform' meant a combination of hardware and operating system. Today there are fewer operating systems, but the term 'platform' has been extended to include middleware as well as operating system-neutral environments (like Java). One effect has been to increase the need for something like the MDA, which promises to generate platform specific models from platform independent ones that contain little more than business logic.

If OMG can deliver on the promise of the MDA, we could be on the verge of a new era in computer programming, as different from today's practices as Java is from assembly language. Better still, there will be no need to cut over sharply from the old way of writing software. With the help of 'reverse-engineering' tools, existing source code can be translated to UML models which can then be used as the basis for new designs.

Gradually, at their own pace, organizations should be able to enter the world of the MDA and focus their efforts on writing and maintaining business logic instead of coping with the myriad distractions of rival operating systems, databases and middleware that are so beloved of programmers (although not businesses). And herein lies the rub. The MDA is clearly a 'good idea'. So was AD/Cycle, at least conceptually — and the MDA has already gone far past that.

But the development community is conservative. It knows what it knows and does not like change. You only have to examine the extreme reluctance of synchronous programmers to think asynchronously to understand resistance. However virtuous the MDA is, its success will depend on it working ***and*** *being inexpensive (q.v. AD/Cycle)* ***and*** *being acceptable to those who will have to use it. Fail on any one of these and it will likely fail on all — irrespective of its other virtues.*

Yet, let us offer the last word to Microsoft. Ever since it attained any prominence in 1994-1995, CORBA has been the target of constant sniping from Microsoft and its allies — presumably because it was seen as competing with Microsoft's own Distributed Component Object Model (DCOM). In view of Microsoft's enormous influence with analysts and the media, this was a substantial handicap (for CORBA). The MDA is unlikely to meet with such overt resistance:

- ***partly because it is such a logical step forward from UML***
- ***partly because Microsoft has always been a firm supporter of UML.***

MIDDLEWARESPECTRA
Enhanced Intranet Subscription

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:

- ✓ 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 versions of the 14 volumes of the *Issues in Middleware Collections* (collectively priced at over US$7,500/Euros 8,000 if all bought individually)
  - ➢ 5 printed copies of each quarterly Report (these copies can be mailed to separate addresses)
- ✓ only US$4500 or Euros 5000 (saving over US$3,000/Euros 3500 if purchased as an Enhanced Internet Subscription)

Updated	■ Middleware In Action Collections (5 Volumes, 449 pages in all: US$95 each Volume; US$245 for all five)
Updated	■ Middleware Architecture Collection (637 pages, $795)
Updated	■ Strategic Issues in Middleware Collection (748 pages, $895)
Updated	■ Distributed Systems and Middleware Collection (791 pages, $795)
Updated	■ Middleware in Distributed On-Line Transaction Processing Collection (504 pages, $995)
Updated	■ Middleware in Data Warehouse, Database and Database Access Collection (225 pages, $375)
Updated	■ Messaging and RPC Collection (514 pages, $475)
Updated	■ Queuing Middleware Collection (436 pages, $495)
Updated	■ OO and Middleware Issues Collection (369 pages, $395)
Updated	■ Middleware and Application Development Collection (537 pages, $475)
Updated	■ Enterprise Application Integration and Middleware Collection (308 pages, $395)
Updated	■ Internet and Middleware Issues Collection (426 pages, $495)
Updated	■ Middleware and Message Broker Collection (240 pages, $395)
Updated	■ Middleware perspectives (46 pages, $150)

To order your MIDDLEWARESPECTRA Enhanced Intranet Subscription, call:

(in the USA/Canada) 1-763 502 8819
(Europe/Rest of the World) +44 1962 878333

or email:
spectrum@middlewarespectra.com

MIDDLEWARE**SPECTRA**
is published and distributed worldwide by:

USA and Canada:
Spectrum Reports, Inc.

Subscription Center
PO Box 32510,
Fridley, MN 55432, USA
Telephone: 763 502 8819
Fax: 763 571 8292

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 1356-9570