FINANCIAL MIDDLEWARE**SPECTRA**

incorporating Enterprise Middleware

Contents

February 2002

Volume 16 Report 1

JMS and SonicMQ at Syntegra

Finlay Fraser
Technical Design Architect
Syntegra

Management introduction

Finlay Fraser is a Technical Design Architect at Syntegra, BT's systems 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 a coherent 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 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 and 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 email, etc.

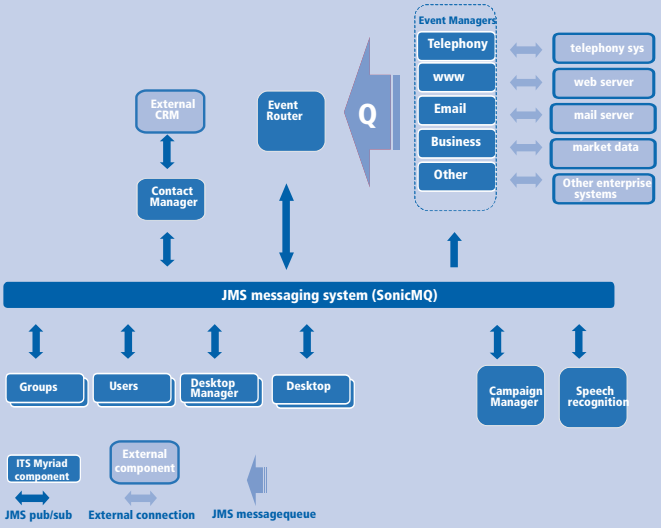
Independence

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 geographi-

Figure 1.1: JMS in ITS Myriad



cally 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 onsite. We want to simplify installation, configuration and subsequent support activities. A major consideration was, therefore, the degree to which the various JMS 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. 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.

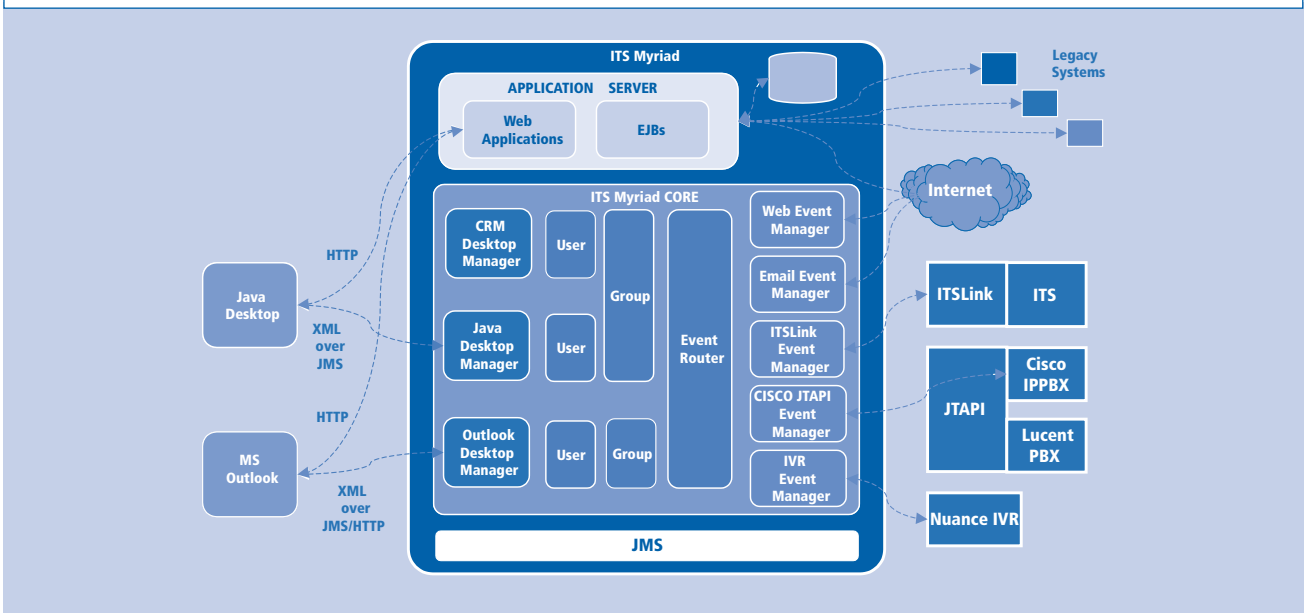
In practice there may be multiple such event routers, usually for load balancing purposes. These would all receive events off the same inbound queue — although each event could only be read once (we did not want the next message being read by more than one event router). The event router component is responsible for working out which topics are relevant to each event. It then publishes 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 the 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. The primary reason was that we realized 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 to add data to these raw events. The raw event might be your phone number. We would perform a trans-

Figure 1.2: ITS Myriad



lation 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. 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, 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. It is the place which holds the rules. These rules determine which events, as they come in off the various SonicMQ queues, should be published to which topics.

Connections with CRM

One question we are often 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 in wholesale trading environments while extending the relationship management concept 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 accomplished. 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 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.

‘Integration banking’ — making the voyage

Mark S. Allcock
Senior Partner and Chief Executive Officer
TMC Alliance

Management introduction

This is the last of a four part series commissioned by **FINANCIAL MIDDLEWARE-SPECTRA** *on the strategic use of middleware and integration technology in the finance and banking market place. Mark Allcock discusses how to put the pieces together to create the ‘big picture’ — by applying the techniques and approaches described in each of the three previous analyses:*

- ***Integration banking program management: charting the seas***
- ***Integration banking architecture: navigating the vision***
- ***Integration banking design: wind in the sails.***

In this last analysis, ‘making the voyage’, he consolidates the advice, guidance, experience and hints provided previously and summarizes what it takes to deliver a complex banking integration project. He is able to do this from a position of experience. Prior to setting up TMC Alliance, Mr. Allcock held senior management positions with some of the world’s largest investment, asset management and private banking organizations. At TMC Alliance he assists financial sector clients (in Europe, North America and the Middle East) to manage major change as well as introduce integration and middleware.

Resetting the scene

In earlier parts of this series, I described the planning and management actions that we (at TMC Alliance) believe all companies should follow prior to commencing any integration project. In our experience, taking the time to identify the key management, project planning and risk issues up front, prior to starting any project, vastly improves success.

In this context I identified particular key factors as being the difference between failure and success. The list is long and includes consideration of the scope, skills, budgets (money and time), plans, risks, rewards, capabilities, supply, demand, knowledge and dependencies.

Making the voyage

On a larger canvas, the baseline factors which we believe need to be considered as part of major business integration projects — especially those found within the banking and insurance sector — are:

- **a strategic business context that clearly defines and positions the products and services that the organization offers to customers**
- **an integrated business architecture that describes the various systems, people and processes to be integrated — and also the interfaces that will need to exist between each**
- **a defined 'integration program' which incorporates an assessment of the core competencies, abilities and risks within the internal, vendor and supplier organizations plus a plan that combines these in the most effective, efficient and extensible manner**
- **those integration, process management, middleware and technology platform tools and interface definitions which are appropriate and able to deliver the desired integrated business architecture.**

By considering these factors we find that success depends on both hard and soft factors plus subjective and objective decision making. In our experience, the creation of an integrated solution is only assured when all the above factors and the execution of the program plan are completed. The likelihood of successful delivery can be further improved by following a number of steps, as described below. These steps can, therefore, be used to reflect the key structure and strategy of a program plan to:

- **define your program goals and business driven measurements (including key performance indicators and success measures)**
- **define the tasks to be completed — based on the integration defined within the business architecture**
- **establish specific roles and responsibilities after assessing the skills, experience, knowledge and capability against the integration and architectural tasks to be undertaken**
- **define the program phases, work streams and the deliverables, together with the 'approach' to be adopted for each.**

In the last of these, depending on the size of the project, separate work streams are almost always beneficial. We advocate that they are created for selected key activities, particularly those related to business, system and enterprise integration which encompass:

- **the overall business integration architecture; this should describe precisely how the functional components are intended (together) to create an overall business solution**
- **the business process definitions; these should include business process, straight through processing (STP) and workflow as well as detailing the task definitions (concentrating on what *needs* to be done, not on *how* it should be done)**
- **the business interface definitions; these should define where are the points of interaction in the overall business solution architecture as well as how they need to work (as part of the business operation)**
- **the system interface definitions; these describe the system interface mechanisms, modes, technologies, data, protocols, communications, messaging formats, syntax and semantics supported; each definition should incorporate the expected as well as the exception conditions for each interface (plus any compensatory actions)**
- **the business process to system interface mapping; accomplished well, this defines the interfaces required to support the solution and identifies any gaps where new system interfaces may be required.**

Other steps include:

- **creating individual project plans for each work stream; these need to be synchronized within the overall program plan and management responsibilities need to be assigned for each project reporting to the program manager**

- **integrating work streams together so that when individual work streams are running on time, in scope and on budget there is a minimum of distraction; the objective is to minimize risk, whether to project, program or even the business**
- **conducting frequent independent 'improvement focused' project and program reviews; this is best done using resources that do not have day to day involvement and the object is to ensure that project and program goals are converging and meeting the business objectives and strategy**
- **ensuring the program plan is based on incremental phases that evolve to provide a progressive delivery of benefits; this boosts confidence over time, while balancing the risks typically found in major or complex projects**
- **initiating each project within the overall program at the earliest possible point while re-synchronizing project definitions (at each initiation as well as at pre-defined checkpoints) with the program goals; the idea is to track changing business needs and priorities.**

Each project within the overall program, therefore, must have clearly defined:

- **objectives, scope and deliverables**
- **detailed requirements, definitions and specification (if appropriate to this project stream)**
- **stated resources, costs, business benefits and timetable**
- **risks, mitigations, issues and query logs; these should be instituted upfront and maintained throughout the project**
- **progress and performance criteria**
- **success and failure criteria (when complete, not complete and when to abort); these should be created at both program and project levels**
- **acceleration, even deceleration, options — around time, resources and costs (for significant project work streams within very large programs); these keep a 'tally' on what could or should be done should a need arise**
- **project test plans, test cases and test cycle definitions; these have both technical and infrastructure perspectives as well as human resource implications.**

The most important

From a business, system and enterprise integration perspective, our experience is that, of the above considerations, the integration work stream activities are the most critical. They touch the most points. They have the potential to impact nearly every other work stream.

In analyzing successful projects that we have completed over many years, we have also determined that managing information on business and system interfaces is probably the single most important 'make or break' activity for any overall program. Effective integration information management, within the wider business and integration community and across IT, reduces the risk of failure. It supports the significant benefits that can be realized from adopting an enterprise wide approach for major integration initiatives.

To satisfy what we have seen financial integration projects to need, we developed a strategy for improving integration success. This centers on a repository by which we manage the information which is required to provide the business and system interfaces. To work, such a repository must include, at the very least:

- **integration information**
- **business process information**
- **business interface information**
- **system interface definition information**
- **business function to system interface mapping information.**

Each of these has its different aspects. These are as listed below.

Overall solution architecture: integration information

This information consists of:

- **the name of business functions**
- **the functions and their interfacing**
- **the type of interface information**
- **the frequency of interface and volumetric data**
- **the business data content to be exchanged across each interface.**

Business process information

This information consists of:

- **the name of the business processes**
- **the functions within business processes**
- **the work steps within business processes**
- **work step definitions**
- **work step time metrics**
- **business volume requirements — by time of day, day of week, month of year, etc.**

- **process, function and work step initiation (trigger) events**
- **process, function and work step completion conditions**
- **process, function and work step exceptions**
- **process, function and work step services, performance and quality metrics and rules**
- **work routing and allocation rules and methods.**

Business interface information

This consists of:

- **business processes, functions and work steps**
- **the names of business processes, function and work step interfaces**
- **the names of business messages exchanged across interfaces**
- **the business conditions, or events, requiring or triggering business message exchanges across interfaces (including volumetric and service level requirements for each event)**
- **the data formats, content and sources of data required by the interface events**
- **expected, and unexpected, event responses**
- **assessments of criticality, importance, reliability and resilience for each interface event**
- **predecessor and successor event dependencies and hierarchies.**

Interface definition information

System interface definitions are used to describe the system interface mechanisms, modes, technologies, data, protocols, communications and messaging formats plus the syntax and semantics supported by each instance or permutation of an interface. The key information to be recorded includes:

- **the name of the system interface**
- **the type of system interface (database, file, message queue, object, etc.)**
- **the mode of system interface (request/response, fire and forget)**
- **the methods of system interface (message passing, store and forward, pub/sub, etc.)**
- **the type of data interface (fixed, variable, delimited, self-describing)**
- **the domain of data interface (XML, SWIFT, ISITC, FIX, proprietary, etc.)**
- **the name of the data interface message format (MT523, ISO8583-12, etc.)**
- **mandatory and optional data attributes and value rules within the interface**
- **data interface types, domains, message formats and data attributes for expected and unexpected responses from the interface**
- **usage and volumetric capabilities and restrictions anticipated across the interface (for example, 5 simultaneous requests, 100 requests per second, etc.).**

Business function to system interface mapping information

These consist of:

- **the names of business interfaces**
- **the names of system interfaces to support each event from a business interface**
- **the grouping or sequence information which enables one business event to relate to multiple system interfaces, and vice versa**
- **predecessor and successor system interface information, for each business event**
- **any other conditions, contexts or rules.**

Management conclusion

Over the course of four analyses, Mr. Allcock has described how and why the success of major banking and insurance projects is dependent on much more than technology. He has shown how integration requires, fundamentally, a business led and business focussed approach if integration is to succeed. His views are based on experience. The key factors that he believes determine success are based on the effective combination of:

- ***a strategic business context that clearly defines and positions the various financial products and services that an organization offers, or wishes to provide, to customers***
- ***a business architecture that defines the various systems, people and processes to be integrated as well as the interfaces that will exist (both human and technological)***
- ***a defined Integration Program Plan which incorporates an assessment of the core competencies, abilities and risks within internal, vendor and supplier organizations***
- ***the selection of appropriate integration middleware (plus related technology platforms, tools and interface definitions) that are able to deliver the integrated business architecture***
- ***the introduction of a constant self-managing process, backed with appropriate data, so that decisions are anticipated and made on the basis of knowledge rather than unsubstantiated conjecture or wild hope.***

Sharpening the leading-edge: where next for middleware in time conscious organizations?

Colin Osborne
Chairman
The Tivyside Group

Management introduction

The application of middleware has produced real progress in business integration. Yet fundamental business challenges remain, especially for large organizations with major IT departments and multiple systems.

Today's organizational world is unstable. It demands improved solutions for survival, competitiveness and effectiveness. Operational managers want ever-more responsive IT systems in order to plan and manage profitability as well as effectiveness.

This multi-dimensional need has spawned terms like 'zero latency', STP (Straight Through Processing) and T+1 as well as cross function initiatives like those found in logistics and inventory management. September 11th then demonstrated that the need for comprehensive access to information was not specific to the private sector. It applies at least as much to the public sector, albeit with different implications.

In this analysis, Colin Osborne looks into the future at how and why middleware will matter in the forthcoming real time information environment. As he shows, timely data management is going to matter to leading edge technology innovators even more than it used to do.

What is driving change?

Currently there are at least four key drivers which are putting the pressure on for the delivery of major improvement to the real time assembly, processing and analysis of complex information. They are:

- **middleware evolution**
- **real time business management**
- **speed, speed and speed**
- **combining transactions and business intelligence.**

Middleware and beyond

Over the past decade middleware has become ever richer, to the point where you can scratch any analyst and receive a different version of what it includes. In my view we have seen at least four strong phases so far:

- **data centric middleware: typically this is used to cleanse, map and move data between disparate databases, usually where data is separated from the application environment**
- **messaging (whether synchronous or asynchronous): conventionally this is used for application to application transfers, often at a transactional level although it can also interface to any data source or target; typically it requires hard coding at the interface point**
- **Enterprise Application Integration (EAI): this has increased in importance as companies realize the amount of application integration they need in order to simplify (or reduce) the process of hard coding and data definitions**
- **Business Process Integration (BPI): although this is often combined with EAI (but can be standalone), it takes middleware into application development territory and adds value by enabling legacy extensions together with the creation of new applications.**

With so much being delivered one might think that there would not be so much more to do. But there is. New business pressures are dictating the priorities, and certain issues have yet fully to be resolved.

Towards real time business management

In the past decade, in which middleware became widespread, we have seen huge changes to both the business and technical environment. For example:

- **businesses are required more and more to interface with suppliers and customers**
- **the Internet has changed the whole nature of connectivity and customer information**
- **customer loyalty seems to be a thing of the past**
- **customer satisfaction matters above all.**

Described another way, it has been a long decade in which the large have become larger while niche players have had to discover innovative ways to create new markets or penetrate existing ones. In between, mid-sized companies have been squeezed as never before.

In the future, the middleware agenda will be set by the Global 2000 companies — because these have to operate in increasingly volatile industry sectors and markets where customers are able all too easily to switch allegiances. Protecting customer bases, maintaining the ability to make sense of market behavior and having the capability to deliver internal change have all become management imperatives.

Add to this concerns about the world economy in general and it should be no surprise that understanding the detail hidden in the complex data already held in most organizations matters. In times of growth, most were less anxious about market dynamics and causal relationships. With some 20 years of growth, most felt that profits were on the way, and there was cash in hand. That luxury has evaporated, and the need to exploit end to end cycles, from early planning through to post-activity analysis, has once again returned.

One consequence of this is that the traditional industries which gave middleware its initial push are again at its forefront. Financial services, transportation, manufacturing, telcos, retail and healthcare are all important. The one new entry will almost certainly be that of the public sector — especially driven by the aftermath of September 11th. Unsurprisingly this is being led by law enforcement agencies and financial regulators.

What these industries have in common is that they all possess key functions that increasingly demand:

- **improved integration of business processes**
- **data analysis delivered in shorter time frames.**

Financial services

The financial services industry often leads in the exploitation of new technology, not least because it exists in a

highly competitive world with an increasing ability (and willingness) for customers to switch providers. Innovation matters.

In the finance sector, the volumes of transactions are high, the value is frequently monetary and the business processes can be extremely complex. In addition, many regulatory barriers exist. Integrity and auditability are vital. Now add real time considerations to the high value of the transactions and the rationale for innovation is clear.

As a result, the financial services sector continues to possess one of the largest IT spends of any industry, often with significant building of custom applications in large internal IS shops. This combination of custom code, industry volatility and regulatory pressures creates a hugely challenging operational environment. Any technologies that might enable changing business requirements to be met faster are inherently attractive.

The industry's current pressure points include:

- **STP and moving settlement from T+3 to T+1**
- **risk management**
- **portfolio tracking**
- **profit and loss attribution**
- **fraud detection, including money laundering**
- **growth and handling mergers.**

That said, financial services are not unique. Other industries throw up different challenges, all created by the same combination of complex information management in a restricted time period.

Government

Governments worldwide are actively increasing their integration between what previously were stand-alone agencies or organizations. Globally there are examples where social security, health, pensions, labor, taxation and the like are being integrated. On the one hand this is justified as being for 'the citizen', so that he or she can obtain one single view of his or her interactions with 'their' government (local or national). Equally, that single view enables 'the government' to see one view of the 'whole' citizen ...

Beyond this there is a rising interest in improved investigation, tracking and analysis of individuals, both within countries and between countries. Many intelligence and criminal agencies are looking to share information so that they can improve detection rates based on real time analysis of shared data.

Insurance

Insurance providers may specialize in specific sub-sectors (vehicle, life, etc.) or may be multi-faceted. In either case they need to be ever more effective in monitoring and meeting claims, whilst also protecting themselves against fraud. They are also vulnerable to customer defection, and must be seen to be efficient and customer responsive.

Balancing this, many insurance companies have significant opportunities for cross-selling. They wish to share customer information internally, to increase their return per policyholder. All this contributes to the desire for faster, more comprehensive, integration.

Telcos

Telcos are unusual in that their transactions are very simple, with a minimum of data, but occur in massive numbers. This sheer volume creates intervals between the traffic occurring and its subsequent analysis.

At the same time, the telcos have serious problems with fraud, and would far prefer to identify traffic abnormalities in real time. If this were possible they would be able to minimize their losses. Additionally, more thorough analysis would enable more sophisticated payment programs to be offered, as well as bandwidth purchasing to be optimized still further.

Transportation

The airline industry is, arguably, the most extreme example of a real time industry that can all too easily be held hostage to external influences. Overheads are extremely high and are, in many cases, fixed for the short term.

Simultaneously, passengers are becoming more and more unpredictable as they are offered more options — and one of those is to choose not to travel at all in emergency situations. International airlines are involved in increasing numbers of commercial alliances, which require sophisticated integration of their disparate technologies.

Thus airlines need to exchange and manage large volumes of reservations, scheduling and administrative information. Much of it is time critical.

Ports

A variant on the airlines are ports. Increased surveillance by immigration and customs authorities worldwide creates the need for better profiling, plus more sharing of information between ports and transport operators.

Agencies, such as Customs, want to use their experience to determine a selective list of packages to be examined, often dependent on characteristics such as size, transshipment ports, descriptions of goods and consignees. Many of these packages will be within containers (by sea or air), and it is important for Customs to be able to intercept any targeted package before it receives automatic clearance through a port.

Manufacturing/Retail

The supply chain focus within consumer packaged goods/ distribution/ retail is well known. It has been extremely successful in optimizing inventory whilst ensuring sufficient availability to satisfy customer choice.

Both the manufacturing and retail industries have been pro-active over the last 20 years in the search for improvements to the value chain. They are always hunting for the next advance. Any ability to handle even more sophisticated business processes in real time will produce further gains in effectiveness. That is attractive.

So what middleware improvements are needed ?

The simple answer is 'many'. Businesses need systems that can more easily be shaped by business staff, rather than by scarce and expensive IT resources. This will be especially true in the next decade, which will be characterized by unprecedented change:

- **flexibility and responsiveness remain the key, but the expectation is that it will be achieved by ordinary mortals, not IT specialists**
- **hard coding will be anathema, as will be anything that does not produce easy and profitable re-use**
- **corporations will expect to mix'n'match to suit their problems, rather than be forced again into the single-sourcing route of 1990-style ERP monoliths**
- **pricing will be driven down, and many of today's middleware components will become commodities, including application adapters.**

The result will be that innovative middleware will be easily understood, with a value add that will create its own reward. In time, of course, even this will eventually become a commodity.

In this context, what variables will change? Not too many; the requests will just become much harder to meet and some of today's soft options will vanish. For example, you will see more volume, broader communities (of interest), more data sources and targets, Web Services, yet more putative standards, more objects and components and greater regulatory demands. In short, nothing fundamentally changes: we will all be under continual pressure to do more, more quickly.

This does not mean, however, that you will see universal APIs or XML solving the world's integration problems or causing the death of EDI or a comprehensive set of adapters. Nor will you be able to afford (retain) the ability to cripple data content in order to cram high volume processing into tight batch windows. Regulators, if not commercial managers, will see that short cut removed.

The result is going to be a more complex world in an economic climate that:

- **on the one hand, is increasingly hostile to new spending**
- **on the other, demands more precise information, more urgently, in order to improve organizational decision-making.**

That is the environment into which future middleware will fit.

Transaction processing and business intelligence — linking two worlds

The need to achieve an integrated real time business view will increasingly drive middleware and managers to integrate two worlds which have largely been unconnected:

- **transaction processing (or TP)**
- **business intelligence (or BI).**

The transaction processing world increasingly directs the delivery of products and services designed for markets of one — with all that this implies in terms of 'build to order' and inventory planning. Technically, this transaction processing exists in the world of entity relationships, of ACID properties, of status management and transaction sets. Typically it is handled in relatively small volumes but at high speed.

In large organizations, TP concerns itself with transactions in progress. These tend to be time sensitive. The transaction is complete when an individual transaction finishes and the audit records are written, almost certainly to an operational database and/or to some form of data warehouse.

As such, TP is middleware and is a major user of middleware. It bridges islands of computing and integrates applications within enterprises and even beyond. It knows about real time issues — short running transactions are its metier — but it will need to extend to embrace the state management sophistication demanded for long running transactions.

In contrast, BI is associated with:

- **aggregate data**
- **huge volumes**
- **multiple sources (and targets)**
- **the (largely unmet) need to handle massive ad hoc queries.**

In technical terms, the key words are dimensions, rows, facts and databases. Acronyms — such as ETL and OLAP — proliferate. (There are even different semantic meanings for the same expressions depending on whether it is TP or BI about which you are talking.)

While business intelligence has absorbed much investment, high volume users of today's BI approaches — and associated storage technologies — describe serious problems. Today BI focuses on historic data that may, at best, already be days old. BI struggles if asked to process high volumes of real time updates or analysis.

Users describe BI as 'stretched' when they need it to handle unstructured views of data and its variables. The business intelligence world requires tools that allow both predictive and reactive testing of data, to understand causal relationships and to test 'what-if' hypotheses. In practice the main use of BI middleware is for database extraction and reformulation, and for some cross-application communication.

This is not enough. Businesses and governments already demand more precise feedback. This is only going to increase, not least with:

- **the growth of 'markets of one'**
- **the need to 'build to order'**
- **the desire to understand causal relationships**
- **the requirement to optimize prices**
- **the wish to trace and track individuals (purchasers or products or suppliers or ...).**

All these focus the need for total information. This will need to be summed from ever more sources. Furthermore, the common BI shortcut, of using sub-set data, is already recognized to be insufficient at best, and potentially misleading at worst.

Today's TP and BI roles are vital. The difficult is that, in isolation, each is insufficient to enable the move from running complex organizations in batch to running in real time. Instead what users are looking for are new joint TP and BI functions that are designed to assist a transition.

Successful new architectures will, therefore, be:

- **optimized for speed and flexibility**
- **able to handle millions of transactions in seconds, for both TP and BI.**

To provide real time support they will need to be usable by business analysts. No longer will either TP or BI be constrained to programmers. Parameters and rules will be changeable in minutes, not the hours or days (or weeks or months) of the past.

Will IT improve its core infrastructure?

As usual, there is no single element to the solution. There is no fairy dust.

To succeed, users will have to tackle a combination of issues that act, often collectively, to limit their capability to derive real time feedback from high volume operational information. As such, solutions must be incremental and able to:

- **offer support to existing infrastructure components as well as legacy applications**
- **accommodate innovation.**

Today's constraints were established by the bottlenecks inherent in many of yesterday's deployed technologies, especially when these had to co-exist with restricted operational time windows. For example, the architectural environments which we still use for data storage and analysis are decades old in their design. Most stem from when computing architectures were more consistent than they are today, and when costs for hardware — such as memory and disk — were radically different.

The issues that need addressing include the limitations found in:

- **the RDBMS**
- **the spreadsheet**
- **data warehousing**
- **data mining.**

Each of these is explored below.

The limitations of the RDBMS

RDBMSs have done a great job since their introduction thirty years ago. They have enabled users to manage vast quantities of data reliably while underpinning an explosion in the number of applications.

However, their conceptual design principles are challenged by the realities of today's commercial and organizational environments some three decades later:

- **for cost reasons, most are optimized to reduce memory usage rather than to deliver speed**
- **in the late 1970s, data volumes were dramatically different from those deployed today, both in potential maxima and in actual use**
- **data construction has become ever more elaborate, yet most RDBMS products have serious performance limitations when confronted with the analysis of very large data sets**
- **most RDBMSs can only work in one axis at a time; this is a fundamental design restriction that requires much additional coding if 'workarounds' are to function**
- **the architectures of RDBMS are not tuned for complex interactions amongst data; this explains the growth of add-on tooling for data extraction, manipulation and warehousing, even though these tools often employ techniques that severely restrict their ability to handle large volumes in real time**
- **hard coding is possibly the most stubborn constraint to overcome (Figure 3.1).**

Handling complex queries in most RDBMSs requires a knowledge of SQL well above the norm. This demands scarce skill sets. Using programmers to produce complex code (that is not easily changed) is an expensive (and time consuming) way to reflect a different view of the same data, and it has low re-usability. In addition the flexibility to test derivatives and alternatives is poor, without the production of yet more complex hard code.

The limitations of spreadsheets

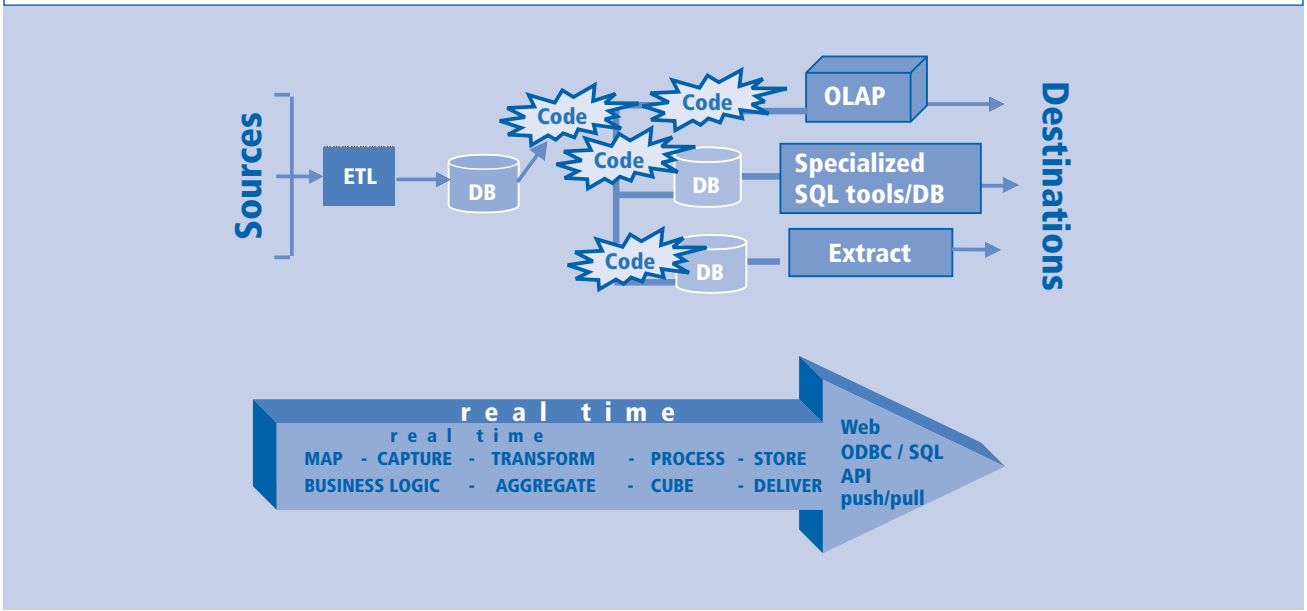
Interestingly, the Excel spreadsheet can outperform many RDBMSs for complex analysis of small data sets. Unfortunately, Excel (for example) does have a scalability limit of 65000 rows.

Despite the speed, this trivially low capacity (not unreasonable for an office application) is insufficient for the high demand worlds of financial services, transportation, utilities, insurance ... or any other sector which needs to handle large volumes.

The limitation of data warehousing

Operational practices for data warehousing normally call for production and data warehouse databases to be kept

Figure 3.1: Moving from the traditional to real time data



separate. While this is a delight for database vendors, it condemns most data warehouses to be composed of out of date information from which no real time feedback on queries — including the latest operational data — can be obtained.

An unintended result is duplication of staff costs — to manage both databases. Even with so much resource, the key requirement — of zero latency — is not achieved.

Additionally, the constraints of storage and processing time have often caused rich data sources to be stripped down between transaction source and storage. This reduction severely downgrades the relevance of subsequent analysis as granularity is lost and cannot be rebuilt — thus dramatically reducing the scope and value of the organizational feedback that is produced.

Data mining

Data mining tools are a specialist breed which add significant value to any RDBMS. They are often niche products, requiring several to be used together to obtain the maximum benefit. Yet, even then, possessing them can induce new structural constraints.

For example, on any sizeable data volume, pre-processing time will be extensive. In a recent public test — designed to show the efficiency of some database tools — much was made of the speed of processing of a large data set. What was not discussed was the '55 hours of pre-processing' required before this wondrously fast test could occur. Too often the cost in time and hardware and skills is forgotten in any data mining cost effectiveness calculation.

So what is needed?

We must build a robust bridge between the artificial separation of business intelligence and transaction processing. I say 'artificial' because no business dynamics separate these areas. The separation exists because we have not previously had the technical capability to bridge them in real time — not because we have been lacking the requirements so to do.

Middleware has done a great job in reducing the gap, but it has not yet closed it. The power of middleware will be enhanced by incremental capabilities to address a wider range of issues within the spaghetti jungle, and wring more payback from today's infrastructures.

Speed must be added to every process, to enable us to manage huge volumes of data in real time as well as make flexibility an integral design element. We must do all of this in a way that builds on today's investment in technology and staff training.

Above all, in a recession, payback must be immediately evident, and the total cost of ownership reduced. This latter point is non-intuitive, as most emerging technologies create premium prices for early adopters. For today's business realities, that is not good enough.

What will this look like?

A new software segment is emerging. This will enable real time decision making and provide organizational intelligence from massive volumes of data in seconds, rather than using a batch process taking many hours or days.

This segment will be optimized around major organizations needing to derive business feedback:

- **from huge amounts of data — potentially multiples of terabytes**
- **from many sources and destined for many targets.**

Much of this will need to be capable of being interpreted by business processes. Results will be wanted in real time — even for ad hoc enquiries made on in excess of a billion transactions or data items. Change and flexibility will be inherent and these systems will be used by business analysts (with minimal IS support).

Any solution to this challenge cannot be super-OLAP, or ETL on steroids. It is not EAI, RDBMS or memory resident databases all rolled together.

Rather any solution must leverage middleware, but will not itself be middleware. Instead it will be a comprehensive architecture, designed from first principles to be a power engine, optimized to deliver zero latency, yet able to work with installed investments in existing technologies, tools, and applications. Its key components will consist of:

- **rules-based environments**
- **speed — as in vector processing and storage**
- **integration tools.**

Rules-based environments

Think of information as being a fluid. Information is continually changing — in content, in source and in destination. This dynamism blends uneasily with today's systems, optimized as they are for semi-static data.

Rule-based applications are about to accommodate fluidity in a way that hard code and tables cannot. A flexible rules-based environment can bring the necessary simplicity and speed of change to business processes, data definitions and operating parameters (see also page 24 and Rule ML).

As with most things involving middleware, it is worth noting that rules are implemented with markedly different degrees of sophistication. As an example, consider a major international bank. It has hundreds of linkages which are required to drive a daily P&L statement which processes:

- **data from 70 countries via more than 40 sources and 700 fields**
- **multiple millions of transactions which occur in many currencies.**

In the SQL world, this calls for complex code involving multiple splits and joins. It is a challenge for even highly experienced coders.

But the bank's environment is dynamic. Sources, content and targets change constantly — with each change requiring significant rewrites in SQL. In contrast, in a rules-based environment, only simple changes are needed.

Speed, speed, and more speed

Business and computing share one common characteristic. Nothing is ever fast enough.

One potentially dangerous consequence that has occurred within complex information environments has been the trade-off between providing speed of business response at the cost of information quality. IT departments are often forced to reduce the level of detail carried forward from individual transactions in order to enable the volume of business transactions to be processed in the available time window (for example, an overnight batch run). The result has been, and is, frequently a dumbing-down of data plus the loss of potentially crucial detail.

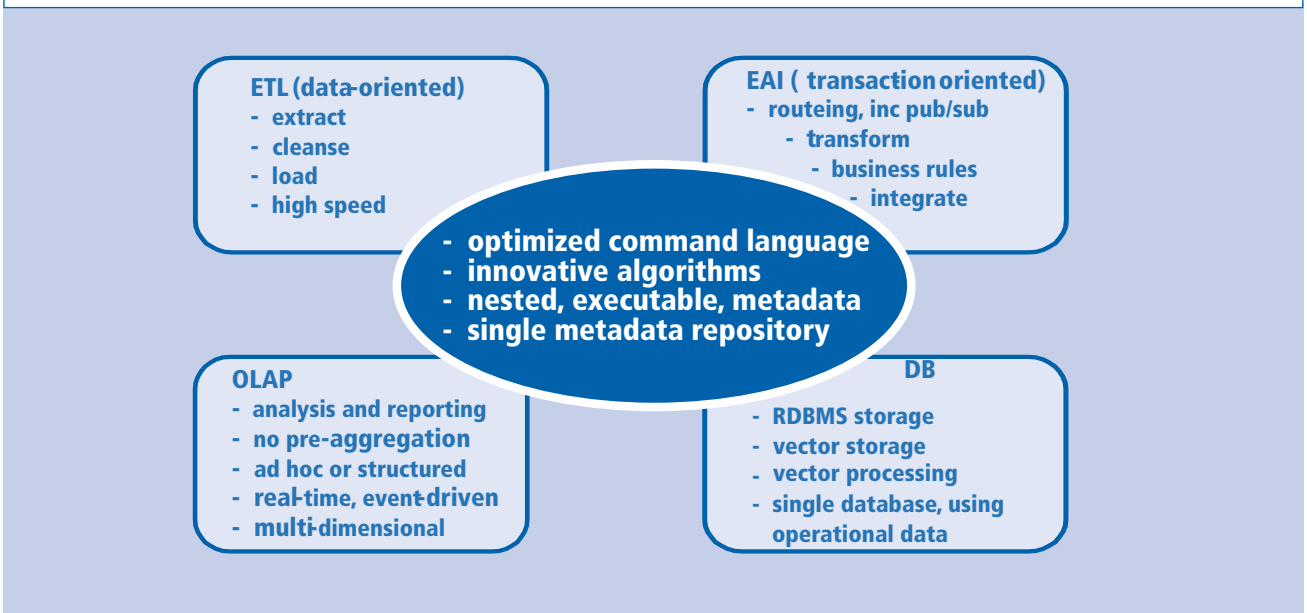
The need for speed adversely impacts the richness of the analysis possible — or produces duplicate processing. This is not an attractive scenario. Most managers are looking for this to be overcome.

Yet speed is not just about CPU power or run time execution. End to end speed includes many hidden delays — which are caused by definition, coding, pre-processing and preparation. Individually or in combination, addressing these can offer vastly greater reductions in time consumption than buying run time speed alone.

This matters because a lack of responsiveness may deliver frustration or inaccuracy. In turn this too readily becomes a war between business managers and IT, especially if it is perceived as being a barrier to corporate competitiveness.

The missing link for power users has been power itself. Even with rules-based applications, engines must exist

Figure 3.2: What the new world needs to encompass



which are able to provide the raw performance required to manipulate massive amounts of data. But such scalable engines have, in the past, not existed for the mass market.

This is changing. Today, engines are emerging that deliver the power to combine millions of transactions and billions of facts in near real time across disparate environments.

The significance of vector storage and processing

These powerful information engines are driven by vector processing and storage. They are magnitudes faster than traditional RDBMS-based infrastructures. They have been proven in specialist situations (often where budgets were available) such as weather forecasting, defense, space, mapping and life sciences. They are now ever more common in the commercial world, not least in forms like that well known Internet life-saver — the Google search engine.

In my view we are going to see two primary manifestations of such vector power:

- **the first will exploit vectors to take data temporarily from existing data sources, reprocess it and the return it to conventional RDBMS data structures**
- **the second will use vector arrays as the native storage medium.**

Initially the first manifestation will dominate. Organizations will seek the fast payback, while testing the concept holding data in vector databases and assuring themselves that integrity and auditability can be trusted.

The second, however, has the potential to have the more profound impact. It offers much greater advantage — especially for re-use, for real time management and for removing computing roadblocks. It may even become, if it lives up to its current promise, the way in which transactional data becomes the sole source for all operational and analytic functions. (BI would not disappear but data warehouses, as separate entities, might.)

The existence of such power is going to drive new middleware uses and benefits. Simultaneously it will release still more value from existing middleware installations as well as an improved RoI to those organizations which have already invested in middleware.

The international banking example I referred to earlier is one of the most highly-challenging operational environments I (personally) have ever seen. Examining the comparative performance of traditional and vendor-based computing is interesting. In this particular case, the batch process time needed to run the complex blend of transaction processing and business intelligence was reduced by 95% (of its conventional run time performance), with further potential for efficiencies being available via tuning.

Importantly, scalability is linear. This is critical in any environment possessing growth or operational spikes (not least after a merger or acquisition has been completed).

One additional point to note is that vector storage will not just hold raw data. It can include metadata, the descriptions which define data and how it is managed, plus business rules. In effect this (vector storage) will become a single repository for:

- **data**
- **transformation rules**
- **business process**
- **operating parameters (trading partner profiles, etc.).**

The second manifestation scenario will also take longer to establish itself. Yet, even now, progress is being made. For example, in the telco, at least one company has been constrained (by current technology) to seeing at most two days of operations at once. It needed more.

Using an approach based on vector storage, it expanded its view to cover a full month. Indeed initial tests show an analysis of a billion transaction rows taking less than 20 minutes. A comparison, with traditional ways, does not exist: examining a month of data or including a billion rows was but a dream.

Integration tools

The success of data vector processing demands integration tools, and middleware. If the data cannot 'reach the vectors' it cannot deliver the raw material on which business insights will be based.

It will, however, be some time before even leading-edge technology adopters are able to reach the stage of an enterprise-wide vector array database. Even when they do so they will need to continue to interact with external agencies — suppliers, customers and regulators. At least, however, the power of their database and rules environment will allow them to engage in integration across a range of mechanisms — EAI, Web Services, EDI, bilateral and ad hoc.

We will see continued payback from existing investment in middleware, whether brokers, message queuing, publish/subscribe mechanisms, ETL or OLAP tools. All will interact with vector systems via rules and published APIs, no doubt spawning another product line for adapter providers.

Why this matters even more: the challenges of recession

Recession reminds us that, above all, organizations need to manage customer satisfaction at every phase of the relationship. Organizations need the intelligence to understand what customers want — and how they then handle the customer (or partner or supplier). Many industries now build to order rather than to maintain stock, which only reinforces the need for accurate data. This, in turn, becomes the input into planning and can be shared (via more middleware) with suppliers.

Competitively most organizations need to glean predictive and production data in as close to real time as is practical. The zero latency world exists. The use of middleware with rules-based vector processing and storage will be a major thrust for companies desiring to exploit short-life markets as well as wanting rapid operational feedback.

Management conclusion

Change is constant. The past 12 months have shown us all extreme change, with the bursting of the 'dotcom' bubble, recession and appalling tragedy all affecting how organizations are run. Furthermore, it is now a given that organizations will themselves be monitored and measured, as individuals and as collective entities. In part this will be in response to market requirements, in part at the behest of other regulatory imperatives.

Those providing integration and information management will find themselves driving new changes. In this context, suggested actions for IT include:

- ***assume that increasingly you will handle real time push and pull event-driven information, whether to screens, to reports, to downstream applications or to data stores***
- ***work with business managers as they re-plan their priorities in the light of the past few months; few organizations will survive 2002 with the same key priorities and action plans that they foresaw at the beginning (or even the middle) of 2001***
- ***understand the hot spots where you will be asked for help to solve complex information problems***
- ***be proactive in thinking through how your own architectures can be made more flexible and responsive to meet these changes, and how to gain better value add from existing IT resources — whether via tools, such as middleware, or via people***
- ***test your development environments to see whether rule-based systems offer you the responsiveness and savings achieved elsewhere***
- ***understand what critical scalability issues are likely to emerge — whether through market changes, needs for new forms of analysis, merger or acquisition, organic growth or whatever: in effect, think through the unexpected.***

These benefits will be most important for power users. Short term benefits will be most beneficial to those organizations at the boundary of their abilities to manage key information in timely ways today. The more complex your organization, the more obscure the needles you search for within ever larger haystacks. The greater the volume of data you move, reshape, and interrogate, the greater your interaction with external bodies — and the more you will benefit from emerging technologies which exploit middleware.

Network computing and middleware for the next generation

Steve Ross-Talbot
Chief Technology Officer
SpiritSoft

Management introduction

The ability to communicate, to interact, is one of the principle differentiators between man and the rest of the animal kingdom. The successive technologies of the telegraph and telephone heralded unprecedented growth in the world economy. With the arrival of computer systems and integration of telecommunications — wired and wireless — the capability to communicate meaning between people, enterprises and systems has never been more important.

Simultaneously, the growth in the complexity of IT infrastructure has given rise to unprecedented growth in integration software. From databases in the 1980s to client/server in the early 1990s to the Web in the mid 1990s and now Web Services, integration tools play a major role. As application server vendors and messaging vendors start to consolidate, so network vendors are looking to add more value.

What do we have to look forward to? Predictions are dangerous. Divining the future is certainly less sure than explaining the past. Nevertheless, looking at the foci in academia, in JCP, in the W3C and in Web Services, certain patterns emerge.

In this analysis, Steve Ross-Talbot looks at the implications of the autonomic computing initiatives, rules, intelligent networking and Web Services. He shows that they are all interrelated parts of middleware. Autonomic computing may be a long way off; nevertheless, all the other technologies are playing their part in furthering the aims of autonomic computing — and some of them are available today.

Moving up the food chain

The recent announcement about autonomic computing by Paul Horn, the Head of Research at IBM, is more than an interesting turn of phrase. This move was led by observations on the most complex networked computer system of all — the human body. To quote Dr. Horn: autonomic computing is provided "by embedding the complexity in the system infrastructure itself — both hardware and software — then automating its management. For this approach we find inspiration in the massively complex systems of the human body.... Think for a moment about one such system at work in our bodies, one so seamlessly embedded we barely notice it: the autonomic nervous system."

Network and application providers are constantly looking to move up the food chain. In the case of network providers they are all trying to provide value added hosting services. Today, it is insufficient for a network provider just to provide the network and all that goes with it. What each network provider is trying to do is differentiate by providing additional value added services — from application hosting to 'edge of the network' services (such as JMS and caching) and so on.

In this context, for example, network vendors are forming alliances with:

- **mobile network providers — to provide end to end connectivity regardless of location**
- **middleware vendors — to provide the connectivity and integration tools as part of their offerings.**

Meanwhile, the application providers are also moving ever upwards. They are seeking to provide higher-level services, from business process management through to personalization.

What does this have to do with the next generation of intelligent networks? Simply stated, the challenge is to identify and deliver ways to exploit network resources intelligently. To do this you need to know about:

- **the network**
- **what it offers**
- **what applications require**
- **the intent (or goal) of its users.**

If this seems a mix of the confusing and the obvious, it is because so much stems from a blurring of the traditional boundaries that separate the fabric of the network and the business applications that use it. As Scott McNealy once said "the network is the computer".

This vision of an intelligent network is not a million miles away from the Semantic Web postulated by Tim Berners-Lee. His vision is that you ask the Web a question and obtain a reasonable answer. It is a marriage of intelligence within the network as well as encompassing an understanding of the user's intention.

Intelligence in a networking environment

The delivery of intelligent networks (IN) has been around for a long time. But what is meant by intelligence is much less than you might expect.

Intelligence, when applied to current networks, is more about the ability seamlessly to:

- **integrate circuit and packet-switched networks**
- **administer some sort of quality of service (QoS).**

In this instance, QoS is indicative of the level of service for which a network user pays. It has everything to do with bandwidth consumption — and nothing to do with intelligence (at least as it relates to the user).

Intelligence is the capacity to acquire and apply knowledge and to use thought and reason in pursuit of some goal. In applying this to the next generation of intelligent networks and their supporting middleware, we need to understand something of the shape and structure of intelligence as it might apply to networks.

In this context, we need to separate the application of intelligence into two areas:

- **self-awareness**
- **social-awareness.**

Self-awareness

To be intelligent, as opposed to possessing intelligence, is to have some understanding of self — to be self-aware. In a network this is akin to understanding the topology in terms of the connections between processing elements as well as what the processing elements are able to do.

This is valuable because it is only by being self-aware that a network can re-route information in the case of network element or connection outage and so promote self-healing (this is autonomic computing).

Social-awareness

Intelligence is also rooted in an ability to be socially aware:

- **in mankind this manifests itself in our ability to communicate and understand each other**
- **in a network this is akin to the network having the self-awareness of its topology, as well as an awareness of what its users of the network are trying to do.**

Autonomics

If intelligence is important then the next question that we need to ask is where we can find a good example of a distributed networked solution that does all 'its stuff' automatically and yet can accept user intervention when needed. The most obvious example of a complex, autonomic distributed system is, as already mentioned, the human body, [see also Mr. Ross-Talbot in **MIDDLEWARE-SPECTRA**, February 2001].

The human body is a complex system of processing elements that interact, for the most part, in a harmonious way to achieve a common goal. According to Dr. Bakewell of Addenbrooke's Hospital in Cambridge: "the nervous system is divided into the somatic nervous system which controls organs under voluntary control (mainly muscles) and the Autonomic Nervous System (ANS) which regulates individual organ function and homeostasis, and for the most part is not subject to voluntary control." The goal of bringing intelligence into the fabric of the network is to achieve the same level of voluntary control as well as to deliver homeostasis for:

- **the users of the network**
- **the applications that they use**
- **the providers of the network.**

When IBM announced its autonomic computing initiative, it defined eight key 'dimensions':

- **"to be autonomic, a computing system needs to 'know itself' — and comprise components that also possess a system identity**
- **"an autonomic computer system must (be able to) configure and reconfigure itself under varying and unpredictable conditions**
- **"an autonomic computer system should never settle for the status quo — it always looks for ways to optimize its workings**
- **"an autonomic computer system must know its environment and the context surrounding its activity, and act accordingly**
- **"an autonomic computer system must perform something akin to healing: it must be able to recover from routine and extraordinary events that might cause some parts to malfunction**
- **"a virtual world is no less dangerous than a physical one, so an autonomic computer system must be an expert in self-protection**
- **"an autonomic computer system cannot exist in a hermetic environment**
- **"an autonomic computer system will anticipate the optimized resources needed while its complexity is hidden."**

The last is the one that is, perhaps, most critical to a user.

IBM laid down a challenge to the rest of the industry. It stated that "this is bigger than any single IT company" and it called "on the entire IT industry to refocus its priorities on this essential goal". Grid computing forms a part of the puzzle. Intelligent agents form another part, as do initiatives like Project eLiza, IBM's self managing server technology and RuleML (see below).

Standards

While autonomic computing is in its infancy — it is still primarily a research initiative — much progress has already been achieved in moving up the stack, in particular in the area of nascent standards. The most notable, and relevant, of these are:

- **JAIN (Java Advanced Intelligent Networking)**
- **PARLAY**
- **RuleML**
- **the set of standards known as Web Services.**

All of these have or will have a role to play in the advancement of intelligent networks and intelligent applications and so will profoundly change our view of where the infrastructure ends and middleware begins. The continual advancement up the architectural stack will inevitably see provisioning technologies, such as these, enable application semantics to be hosted deep in the network — and not just at the edge.

JAIN

JAIN is here and now and is a 'happening' technology. It is a set of Java APIs designed to develop and deploy Service-driven network applications and services.

JAIN is also an industry framework designed and specified collaboratively by groups of industry partners and experts.

It is part of the Java Community Process (JCP) and its aim is to provide service portability across networks to realize the Java promise of 'write once, run anywhere'. It achieves this by specifying APIs that promote network convergence and thereby enable services to run across:

- **fixed wired networks**
- **satellite networks**
- **mobile wireless networks.**

JAIN supports the concept of a soft switched network in which the switches are written in software and are, therefore, configurable. This approach provides some of the flexibility required to meet the goals of self-awareness and self-healing within the fabric of the network. A more advanced technique is a rule based switch as found in project Rubin (see below).

At the higher end of the stack is the SLEE (Service Logic Execution Environment) in which services run. The aim of a JAIN compliant network is to be able to run services anywhere and the SLEE is the embodiment of this.

What the SLEE does is enable services to be created that can use the common facilities of the network while being independent of any network itself. By enabling services to run anywhere — and ensuring that the services can be managed — we can start to build networks that can host services or applications anywhere within their fabric, regardless of what sort of network it is or which vendor owns it.

Once this bridge is crossed we are able to start to explore the self-healing, self-optimizing approaches at a higher level. We can use the knowledge of user intent and map this to the computational resource on the network in a truly intelligent way.

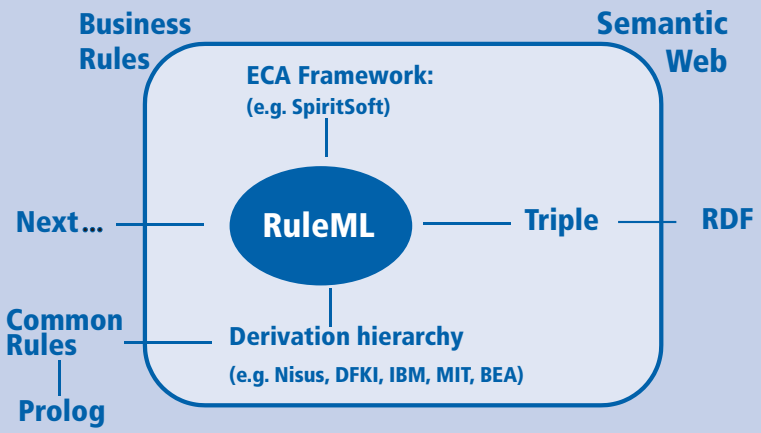
The SLEE encompasses management APIs based on JMX as well as providing services for timers, alarms, tracing and usage. These services are a requirement if we are to achieve the objectives of autonomic computing — namely to be both self-aware and socially aware. By capturing usage and trace information, rule based techniques can then be used to optimize the intelligent network and the applications running over it. The result will be improved services.

PARLAY

PARLAY parallels much of what JAIN is trying to achieve. The key differentiator is that PARLAY is technology independent. That is, it is technology agnostic where JAIN is clearly founded on Java. As such, JAIN and PARLAY are competitive and yet complementary approaches.

PARLAY's stated intention is to define a set of "open, technology independent APIs that enable network operators, independent software vendors and service providers to write applications across multiple networks." Most of the organizations which are supporting JAIN also support PARLAY. Indeed, one soothing positioning of JAIN is that it is a Java technology which is an implementation of PARLAY's APIs.

Figure 4.1: RuleML as an open standard



RuleML

RuleML was born out of the Semantic Web initiative at the W3C. Its main focus is to provide an open standard XML schema that enables rules to be described (Figure 4.1). This provides the basis for rule interchange (Figure 4.2) and interoperability.

Other standards exist in the rules domain (for example, JSR94 on Java Rules). What differentiates RuleML is its language independence and ability to encapsulate both reactive and deductive rules. There are two reasons why these are important differentiators. In the first case, language independence enables rules to play a role on any:

- **platform, from Sun's ONE to Microsoft's .NET**
- **device, from mobile ones through to the highest-end servers.**

In the second case, the ability to represent both rule types produces a runtime flexibility whereby rules can be used to solve many different problems:

- **reactive rules can be used to control flow and manage business transactions**
- **deductive rules enable inference processing to take place.**

In an intelligent self-aware network the use of deductive techniques to diagnose faults — or further optimize the network as usage patterns change — is an approach that has been used by most telecommunications companies (for example) over the past 10 years. The ability of RuleML to combine these rule types opens the possibility of deducing alternate flows that can be enacted as reactive rules.

The ability of RuleML-based solutions to exist as part of the network fabric, as well to encodify business transactions, suggests that RuleML is well suited to:

- **providing the intelligence required to deliver self-aware and self-healing capabilities within the network itself**
- **assisting the network to host agent based solutions that run and manage business goals on behalf of users.**

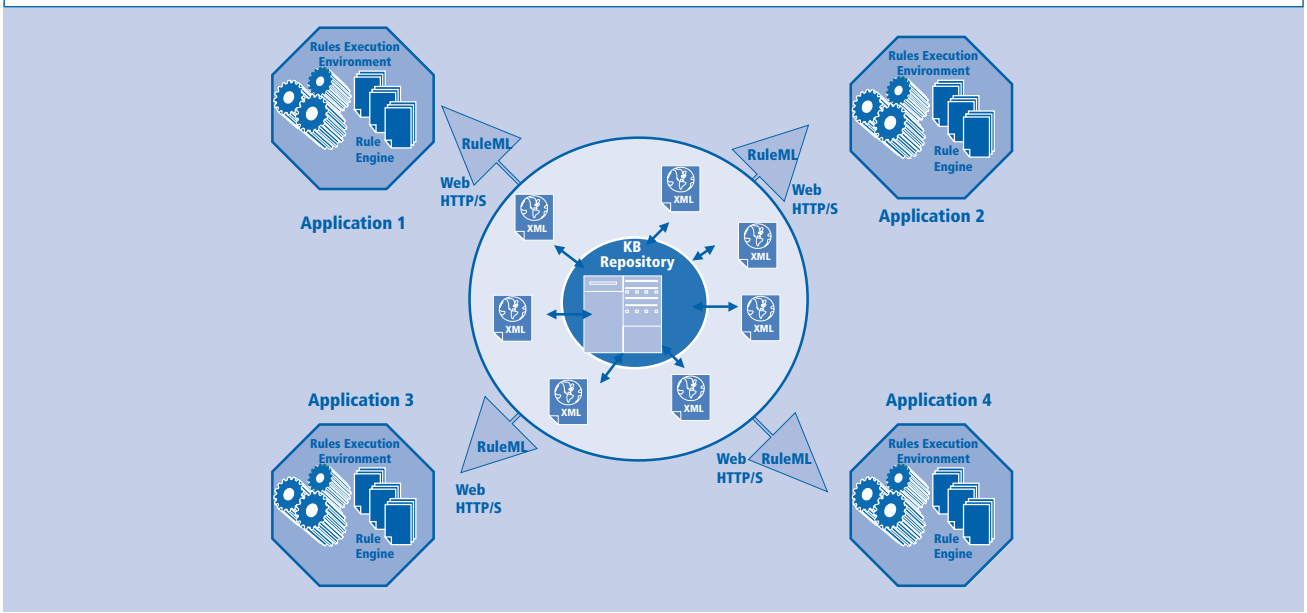
One project that has sought to combine several of these techniques is Project Rubin (Rule-Based Intelligent Networks). Rubin is a software router (or soft switch) that is designed to offer "a configurable and fast implementation for a wide variety of fault tolerant and adaptive routing algorithms".

It uses a rule-based representation of algorithms where the router's behavior can be completely exchanged by reloading the rule bases. The combination of Rubin's techniques and RuleML produces an exchange medium on which a flexible intelligent network infrastructure can be founded.

Web Services

Web Services embody the definitions of:

Figure 4.2: The RuleML interchange model



- **SOAP (Simple Object Access Protocol)**
- **WSDL (Web Services Description Language)**
- **UDDI (Universal Description Discovery and Integration) repositories.**

At first glance it is not obvious as to why these have any bearing on intelligent networks or even automatic computing. Think again.

Imagine writing an application or service that wants to run across any network. To do this you have to take account of the end points as well as the networks' inherent quality of service and variability. It is not enough to rely on the intelligent networks to figure it all out for you.

Suppose you are a trader, trading at your desk and you have an urgent requirement to move from one building to another while you are in the process of constructing a complex trade for a client. Such a scenario may:

- **last over many hours — even days**
- **require much monitoring of price movements and news events, interspersed with the buying and selling of financial instruments (for example, stocks and shares).**

If you have to go mobile and then lose the communications signal — when you enter a tunnel or go through a black spot — while in the middle of executing a bid or offer (a trade), the last thing you want is to be ignorant of your position. You do not want to come out and have executed the trade twice, or not at all ...

Figure 4.3: Useful references

RuleML	**http://www.dfki.uni-kl.de/ruleml/**
JAIN	**http://java.sun.com/products/jain/**
PARLAY	**http://www.parlay.org/**
Web Services	**http://www.webservices.org/**
UDDI	**http://www.uddi.org/**
Autonomic	**http://www.research.ibm.com/autonomic/**
Enigmatec	**http://www.enigmatec.net/**
RUBIN	**http://www.iti.mu-luebeck.de/Research/PC/Rbr/rbr-www/rubin_intro.html**

An intelligent application, backed by an intelligent network, can manage the bandwidth available plus the functions you can perform as a partnership (between the application and the network). The network may decide to throttle back on price changes and concentrate resources for transacting. Or the application may decide that performing the transaction is not possible: it might delegate actions to some server side agent which can then inform you, with a consolidated message, what the position is when you restore contact.

Such a partnership requires applications to have an awareness of network variability and to provide hints as to what to do when the QoS changes. One example of this approach being delivered can be found in Enigmatec Corporation's adaptive application environment. In the work it has undertaken with Sun (and others) it has combined:

- **the notion of an intelligent and adaptive application framework**
- **the RuleML reactive rule technology**
- **the QoS aware intelligent networks.**

Management conclusion

Autonomic computing has, rightly, been used by IBM and others to illustrate what future computing needs. In comparing it to what exists a number of technologies have been highlighted, each of which has a role to play in delivering autonomic solutions.

In this context, JAIN and PARLAY are moving in a direction that will ensure the networks are transparent. Mr Ross-Talbot also describes research in the use of soft switches that can be configured while they are running by using rule technology (RuleML). This, as he infers, is as a means of encodifying reactive and deductive behavior. Its influence, and the relevance of Web Services running on intelligent networks, has also been identified.

Whilst wholly autonomic solutions are some way into the future, the technologies inherent in JAIN and PARLAY provide a solid bedrock for network independence. RuleML then provides a clear way forward for the adoption of rules for soft switches as well as for orchestrating business logic and providing adaptive management of applications, many of which will be built around Web Services.

All in all, for those who thought that middleware was reaching a mid-life crisis, as the enthusiasm for EAI matures, the truth can be seen as very different. Middleware is growing up, flexing its wings and being extended into new process realms.

Making middleware happen: comments on middleware projects

Peter Bye
Senior Systems Architect
Unisys Systems and Technology

Management introduction

The IT industry now has a considerable amount of experience in using recognized middleware standards and products. Middleware has been used seriously for over 10 years (although the earliest middleware developments go back more than 30 years) as a platform for the development of applications, often distributed over several physical systems. It has also been used as a vehicle for an organized approach to systems integration: a middle tier provides a 'front-end' to applications which are combined in various ways to deliver new services. Applications may be written using the same middleware as the middle tier, using other middleware, or no middleware at all. Wrapping or encapsulation techniques can be used to create gateways between architectures.

In this analysis, Peter Bye looks at different ways of using middleware. The views expressed are his but they are based on involvement in many projects using middleware. The level of contact has been varied. In some cases, it was extensive; in others he had minimal direct involvement but was in a position to note points of interest. Necessarily, the examples include a mix of the successful and some that were less so. Projects experiencing difficulties are almost always more interesting, and not just from a sense of schadenfreude: they instruct because we should learn from mistakes (and it should never be forgotten that successful projects might have been even better).

This is not, however, an analysis about the criteria for success of software projects in general. It is selective but in the hope that it can throw light on middleware questions. In all cases, Mr. Bye has respected the confidentiality of the examples, unless there is a published description. Of course, in the vast majority of cases, only the successes are published. Only occasionally are there exceptions to this.

Software projects and middleware

In writing this analysis, I have tried to focus on issues specifically related to the choice or use of middleware. A complication lies in the fact that, while a software project may use middleware, what is of interest about it may not be middleware-related.

Take software project success, for example. The statistics on software project success — or rather the lack of it — make depressing reading. Analysts — such as those from The Standish Group — publish detailed reports showing that only a small percentage of projects are fully successful, just 16% in one Standish report.

The reasons for partial or complete failure are varied. There is no single dominant cause. So, while an unsuccessful project may be based on a well-known middleware platform or product, the reasons for the failure may have nothing to do with the ability of that middleware to support such an application or even the choice of that middleware in the first place. Factors such as unclear and changing requirements, insufficient end user involvement or inadequate management support — none of which have any connection with the middleware — are as likely to contribute to partial or complete failure as the middleware itself.

Such problems have afflicted all forms of software project for as long as we have had software. Indeed, drawing the net wider, the same applies to engineering projects as varied as aircraft or missiles, tunnels or bridges. The software industry does not have a monopoly of failed projects. On the other hand, the wrong choice of middleware — or indeed of any software product — almost invariably leads to difficulties in subsequent implementations, and one of my examples illustrates this very point.

I have therefore sought to select examples where it is the choice or use of middleware that is of interest, rather than some other set of factors. In order to give the analysis some shape, so that it avoids becoming just a list of projects, I have considered the following four different categories of middleware use:

- **choosing the middleware for a project**
- **middleware as an application development platform**
- **middleware as an integration tool, to expand overall application capability**
- **opportunistic use of middleware, to rescue failed projects.**

While this list is not exhaustive, it does I believe cover a representative sample of different middleware-related topics. The meaning of the fourth point, which may not immediately be clear to the reader, will be explained.

Choosing middleware

With any software project, the choice of the software and, increasingly, middleware platform is important. The wrong choice will certainly lead to difficulties, a fact which has long been recognized. On the other hand, the correct choice does not guarantee success, a point emphasized when one considers factors such as unclear requirements.

In an earlier analysis in **MIDDLEWARESPECTRA** (*Implementing middleware-based architectures: the importance of pragmatism* in Volume 15, Report 2, May 2001), I discussed the factors that should be considered in choosing middleware. In particular I considered some of the irrational attitudes which can interfere with a rational choice.

In theory the rational choice factors include:

- **a clear understanding of the functional and non functional requirements**
- **the current environment**
- **available expertise**
- **costs, all other factors being favorable.**

The factors interfering with rational choice were stated as being based on a combination of prejudice, ignorance and fashion — which seem invariably to lead to middleware religious wars. These have a pernicious effect on software projects. Consider two examples involving middleware choice.

Example 1

The first concerns a large European company, with a range of installed systems running its business. An independent consultant was engaged to define a technical architecture to support the deployment of new applications and delivery channels. Integration with the existing systems was considered important, in that they could not be replaced easily. Furthermore, other architectural projects were proceeding in parallel — and needed to be considered in the definition of the technical architecture. At the time of writing, this activity is still in an early stage; no implementation has yet started.

The resulting technical architecture was based on a conventional three tier model, comprising:

- **presentation**
- **business rules in a middle tier**

- **a database layer.**

A crucial point, therefore, was the choice of the middle tier middleware. J2EE, using EJBs, was selected.

In principle, there is nothing wrong with this choice. EJB technology is spreading rapidly and could technically do the job required. There are good products on the market.

However, various other factors pointed in a different direction. The organization concerned had (and still has) no Java systems or experience. On the other hand it does have experience of, and a number of systems written using, Microsoft technology (COM-based). Further, the tools available available from Microsoft for integration with the various existing systems were rather better than the Java alternatives.

Indeed, from an external vantage point, there is no doubt that COM+ could be used every bit as effectively as Java in the middle tier — with the likelihood that these would be more effective for that organization, given other factors. The selection made was, therefore, the wrong one in the circumstances. It appeared to be based on little more than a prejudice in favor of Java.

Example 2

The second example came to my attention when I attended a software conference. It was a case study described by one of the speakers. The presentation and the associated handout comprise my only contact with the project, but it seems to me to be an interesting case of the wrong choice of middleware for the task at hand.

The project was to provide a new system to manage commission payments to a group of retail outlets selling a particular product supplied by a European enterprise (referred to, here, as the 'Supplier'). At the end of each day, the various outlets, numbering several hundred, produced sales information which was collected, processed and recorded in a database. This was a batch process.

A small number of users in the Supplier's organization used the system interactively. Similarly, any of the retail outlets could access its own account with the Supplier, using the Internet. The Internet traffic was low by volume, since the retail outlets were more concerned with selling products than with looking at their accounts.

The system appeared to be straightforward. There are many examples of systems similar to this, and have been for years.

However, in this case, the choice of platform was surprising — at least to me. Java, using EJBs, was chosen to implement the application which was designed to reside in a middle tier. The database was implemented in a data tier, connected to the middle tier by a local area network.

Remember, however, that the great bulk of the work was batch, not interactive. The implementers therefore had to write a batch system in Java, using EJBs. The result, when first tested, was less than satisfactory. The daily run took more than 24 hours — caused in part by the separation of the database from the applications. After much optimization, the run time was reduced to something closer to a satisfactory level, although problems still appeared to exist with such functions such as checkpoint and restart.

It seems to me that, whatever the technological merits of EJB, writing a batch system using an EJB platform is not an ideal approach. The choice of the middleware environment seemed to be based on the preferences of the people concerned, apparently driven by a background in PCs and small systems.

Again, from the external vantage point, there were other environments available which supported:

- **all the batch functions required**
- **the necessary interactive capabilities.**

Any of these would have been a much better choice.

Middleware as an application development platform

There are currently many systems in production using middleware, and the number is constantly growing. Apart from early pioneering developments, recognized middleware products have been in use since the 1980s. Tuxedo, available today from BEA Systems, was one of the earliest forms — and it continues to support many applications including some very large ones. The introduction of object-based technologies and message queuing, especially IBM's MQSeries, provided yet more options for development of applications, often in combination.

Two examples should suffice to illustrate what can be done. The UK Employment Service developed and deployed a large-scale application using Tuxedo as its middleware platform. The system serves over 20000 users equipped with PCs. And anyone using the Internet to buy a PC from Dell interacts with a system using Microsoft technology, with COM+. These are just two large-scale systems out of many.

From a purely middleware perspective, the main concern here lies in the choice of middleware technology and product for a specific project. As I hope I have indicated, the wrong choice almost always leads to problems. Even where two products or technologies may be equally effective technically, other factors, such as the expertise available and the ability to integrate with other technologies, may point more to one product than another.

The religious warfare that can break out when choosing technologies is only counter-productive. Perhaps the most hotly debated choice today is between Microsoft, with COM+ leading to .NET, or J2EE. Each has its advocates (or, perhaps, disciples is a better description). There are those who believe that Microsoft technology is not mature and robust enough, while others believe the reverse. The reality is that there are large systems written using both approaches.

In the end, it comes down to making a rational, considered choice about the conditions obtaining for a specific project, since both approaches can handle most middle tier requirements. Equally, where one technology is the wrong choice, the apparent alternative may not be the right one. The Java batch system discussed above would have been no better if it had been written in COM+.

Middleware as an integration tool

Middleware is used as an integration vehicle, in addition to being a tool for developing new applications. Figure 5.1 shows a conceptual schematic of a typical integration architecture. As can be seen, this architecture is tiered:

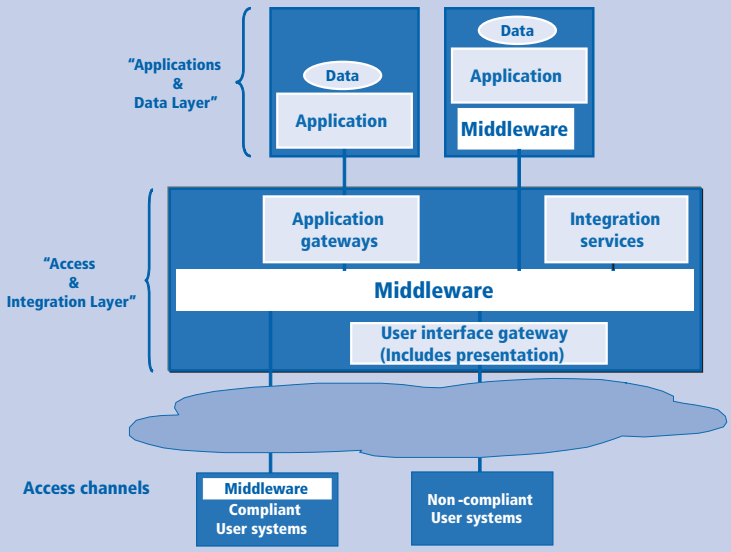
- **on the bottom, there are access channels into the environment**
- **on the top are the applications and data layers (the systems and databases collectively implement services for the users on the access channels)**
- **in the center is the access and integration layer, which is concerned with interfacing to the access channels and connecting to the systems and databases.**

Gateways are used to connect to systems that do not comply with the standards of the middleware. Additional application functions, shown in Figure 5.1 as Integration Services, may be implemented in the middle tier. These functions offer the user services on one side and contain the necessary logic to integrate the various applications and databases to provide the services.

Example 3

The following examples illustrate this use of middleware. The first comes from KPN, the Dutch telecommunications provider. The specific requirement was to reduce the time needed to add a new telephone subscriber — from around 2 weeks to 2 days (or less). The source of the problem was that adding each new subscriber required interaction with several different systems:

Figure 5.1: Middleware and gateways



- **the physical connection database**
- **directory**
- **accounts**
- **etc.**

Each of these had its own, different, terminal interface. There was no automated procedure for the whole process. This resulted in activities 'being lost' between the different interactions.

The solution was to implement a new automated process for the entire operation. It would:

- **offer services to the sales force in the telecom sales offices**
- **manage the whole business process**
- **use gateways to integrate the existing applications, which would remain unchanged.**

Tuxedo was selected as the middleware. In fact, at the time, it was the only realistic choice.

Figure 5.2 shows the configuration and the various elements used:

- **the clients are PCs on a network linked to the sales offices; while they do little more than provide a graphical interface, they are built on top of the /WS product from BEA, which provides a Tuxedo compliant client (it cannot be a service)**
- **Tuxedo sits in the middle tier and supports the new business process to manage the addition of the subscriber as well as the gateways to connect to the existing applications (screen scraping was the technology used); as this middle tier does not contain any database, a number of identical servers are used, one for each sales region so that any failed system can be immediately replaced with a backup (this is possible because there is no data to recover).**

This project was an immediate success, with a fast payback. The reason, in my opinion, is partly to do with the quality of the middleware and partly for other reasons:

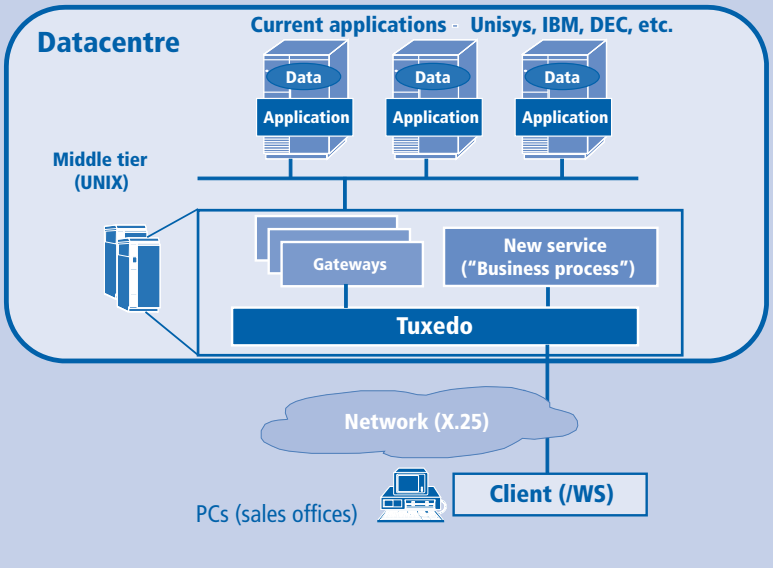
- **the goals were well understood and quite specific in scope**
- **the architecture was carefully thought out in advance, and was simple**
- **the middleware selected was proven, robust, and suited to the task.**

This project is interesting for other reasons. Although completed some years ago, it provided a model for a number of other projects.

One of these is EUCARIS, a network connecting a number of European vehicle and driver registration systems. Its purpose is to:

- **reduce insurance fraud**

Figure 5.2: Middleware as an integration tool



- **help find stolen vehicles — by providing customs, police and other authorized users with access to vehicle and driver information.**

The implementation takes the form of a distributed network, again based on Tuxedo. Users — police, customs or whomever — use a simple teletype interface to a native Tuxedo client in the nearest EUCARIS node. The integration services provided offer a menu of choices and then, depending on the input, connect to the EUCARIS node adjacent to the car registration system required for the particular country in question. A gateway service then connects to the car registration host, using an agreed technique to perform the transactions on the host. Many different host types are supported, including:

- **Unisys' OS2200 Systems**
- **IBM's OS/390**
- **Sun's Solaris.**

The network today covers a large part of Europe.

Example 4

A development of this idea, but using middleware to middleware gateways, is illustrated by a local government organization in the US. The environment includes a Unisys ClearPath system user, with applications running under the MCP/AS operating system.

Before discussing the details of this particular project it is relevant that Unisys had already worked on a project to provide public Web access to a health care application. This had been built on an MCP/AS application running under Open/DTP, the Unisys implementation of the Open Group DTP model.

The local government administration decided to add a Web service to its vehicle registration system, using what it saw as successful Open/DTP implementations (Figure 5.3 shows the configuration). The user interfaces to several information services. These were:

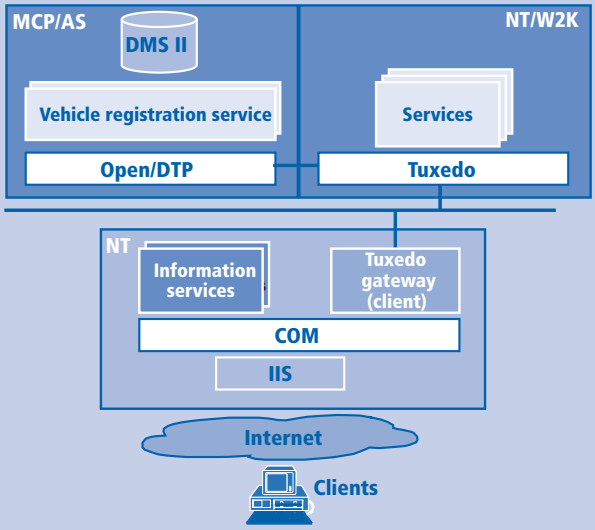
- **implemented using Microsoft's COM technology**
- **built with products from the Openshaw Media Group.**

Access to the vehicle registration application is provided by a client gateway into Tuxedo, which runs in a partition within the Unisys ClearPath system, and additional Tuxedo services can be accessed in this environment. The connection to the MCP/AS application uses eLink from BEA (not shown), which supports gateways to the Open Group DTP model.

What interests me about this implementation is that it:

- **illustrates the advantage of applications running in a middleware environment**
- **shows the power of gateway connections between different middleware standards.**

Figure 5.3: Middleware and providing a service to the Web



The MCP/AS application was easy to access because it had been adapted to run in the Open Group DTP environment, for which there are gateways available. In addition to BEA's eLink, there were several other off the shelf gateways available to connect to Open Group DTP. For example, OpenTI connects the COM+ environment to Open Group DTP implementations.

Opportunistic use of middleware

The point about the advantages of applications running in a middleware environment leads me to the final category of middleware use which I want to consider. It goes by the somewhat enigmatic name of 'opportunistic use of middleware' and is, as I will discuss, concerned primarily with rescuing failed projects.

Too often I have encountered enterprises that have decided (sometimes arbitrarily) to replace large, existing applications. Typically, such existing applications:

- **ran in mainframe environments**
- **had been developed over long periods**
- **supported large numbers of users**
- **were critical to that enterprise's business.**

In most cases, the proposed rationale for the rewrite of such an application was based on a perception that a more 'modern' environment, usually using middleware, was needed. The variety of reasons deployed to justify such decisions (to rewrite the application) included lower operational costs, more flexibility, new functions and so on.

In many cases, the rewrite failed — more often than not completely. Almost always it turned out to be much harder than expected, for many reasons:

- **the choice of an inappropriate middleware platform for the rewrite was the cause of failure in some cases; for example, technologies purchased on the 'bleeding edge' delivered unacceptable instability**
- **more common, however, were 'the other factors' — poorly defined and constantly changing requirements, insufficient understanding of the implications, new management, lack of backing and so on.**

Whatever the root cause, the result was always a serious problem for the owner of the system. Not only did the new system not work but the existing system, which ran the business, usually had not been upgraded to stay in step with requirements.

In this situation, where does the user turn? A new budget to rewrite yet again, with a different technology, is not likely to be forthcoming.

It may be, however, that there are some relatively simple options for improving the existing system, even if these were not originally considered in the enthusiasm to rewrite. In my experience, these can provide rescue vehicles to move forwards.

Example 5

Consider the case of a European public sector agency. This organization was running happily with a mainframe application running in a Unisys OS2200 environment. Some of the users were internal to the organization, and were connected via PCs and workstations. There were also several thousand external users, who used a dial-up interface with a terminal emulator.

The agency decided to rewrite the mainframe applications. It retained a software house to do the implementation. The purpose (of the rewrite) was to:

- **add new functions**
- **improve the user interface**
- **exploit the Internet**
- **introduce an 'open platform'.**

However, the project did not go according to plan. For reasons I will not discuss here, it failed — after a significant delay. Suffice it to say that the reasons for failure were not to do with the right or wrong choice of middleware, but 'those other' factors.

The result was that the existing system continued to run the business but had not been upgraded to include the new interfaces. Nevertheless, the existing system was running on a reasonably current level of software and had been fairly well written. Specifically, it used a display management system (from Unisys) designed to make applications independent of the device type accessing them.

Using middleware products that were built to take advantage of this display management system, it proved a simple task to make the original applications available via the Internet. This did not require any application changes. Indeed the 'rescue' project lasted only a few weeks and involved just three people in all.

New middleware extensions are now being discussed. The difference is that these are being built on the original application foundations.

I have included this example because it is representative of what can be done with middleware to extend the capabilities of applications with minimal risk. My experience is that these approaches should be considered before attempting any rewrite of applications. Doing this enables enterprises to evolve applications into what is required, rather than attempting to follow what is almost always a high risk strategy, namely an application replacement strategy.

Management conclusion

In this analysis, Mr. Bye uses examples of projects to illustrate different aspects of the application of middleware. In selecting these, he has focused on the middleware dimension rather than on the many other factors that can adversely affect all software (or engineering) projects.

In that context his primary conclusion is that, from a purely middleware point of view, the choice of the appropriate middleware for a project is the single most significant issue. Sometimes that choice depends on technical issues: will the middleware support the kind of application being developed? The example of the Java batch system illustrates what he considers to be an inappropriate technical choice.

On other occasions, different technologies and products can be used to implement the same requirements. This is true, for example, of COM+, EJB and also X/Open DTP. And, while a technology such as MQSeries can be used in conjunction with the above, it can also be used in place of some of the other technologies in other circumstances.

Where there is a technical choice, his view is that the final decision must be made based on 'those other factors' such as the current environment, developer experience and the experience of technologies in existing systems. The important point is to avoid dogma.

Finally, he argues that choosing the right middleware is only one step. There are many things that can go wrong in the implementation of a project, even when the right middleware has been chosen. Projects do not implement themselves.

How Enterprise Java makes use of CORBA

Tom Welsh
Consultant

Management introduction

Anyone who wants to understand the software market must take the entertainment effect into account, just as a skilled marksman 'aims off' for a crosswind. To judge from the press, 2001 witnessed the opening scenes of a battle royal between Microsoft's .NET and Sun's Java 2 Enterprise Edition (J2EE) — with all other standards being relegated to near irrelevance.

Among others, the Common Object Request Broker Architecture (CORBA) has been dismissed as too old, too complex, too expensive and too unpopular. Amid the general excitement about J2EE and .NET, it is easy to overlook the fact that CORBA is already a success, if you judge by the rate at which it has been deployed into production and its impact on J2EE.

There is a widespread misapprehension that CORBA is threatened by the increasing popularity of Enterprise Java. As Tom Welsh explores, this turns out not to be the case, for the following reasons among others:

- ***Java 2 Standard Edition (J2SE) and J2EE rely on, and actually mandate support for, a number of CORBA specifications and features***
- ***Java's greatest weakness — its language specificity — can most easily be compensated for by linking 'islands of Java' to the rest of the IT world through CORBA***
- ***many current products and production applications employ CORBA and Java together***

- ***several of the leading Java advocates and vendors — including Borland, IBM and Oracle — are CORBA advocates and vendors as well as being members of the Object Management Group (OMG) which owns the CORBA standard: they see the need to stick together against the threat from Microsoft, and this (more than anything) seems to encourage them to continue to co-operate.***

CORBA: a widely deployed standard

Unfortunately, no one seems to track actual CORBA usage. The OMG allows anyone to download and implement its specifications free of charge without seeking permission or making any acknowledgement. Who knows how many CORBA-based products/implementations are out there?

At least 50 or 60 ORBs are known to exist, but there could be many more. And there is at least one 'unknown' category — ORBs that have been written in-house, either as part of a product to be offered for sale or to support an internal project. Since writing a production quality ORB takes much effort and skill, it is probably reasonable to assume that the existence of 50-60 'commercial' ORBs implies the existence of credible demand — an assumption that is supported by other considerations. Some 800-plus organizations pay OMG membership fees, presumably in the expectation of obtaining a return on their investment. Of these, only about 100 are active CORBA vendors. The rest are actual or prospective CORBA users, although — often for reasons of competitive advantage — they are reluctant to talk about the CORBA-based systems they have built or are building.

Two other resources which suggest success are:

- **the list of CORBA applications at the OMG (www.corba.org)**
- **a list of over 1,300 organizations which have declared themselves committed to using CORBA (www.ericleach.com — under Links/CORBA Users).**

A third approach to verifying that there is still life in CORBA is to provide some contemporary examples:

- **the US Department of Defense, in its E-3 Sentry AWACS modernization program which replaces 1960s vintage IBM System/360 mainframes with multiple PowerPC single board computers linked by a LAN, uses Lockheed-Martin's RT-CORBA as its core backbone**
- **both Computer Associates and Tivoli, competing leaders in distributed systems management, have opted to build their flagship products around CORBA**
- **Hong Kong Telecom is investing $1 billion over the coming ten years in its Interactive Multimedia Service (IMS): the NETVIGATOR video on demand service, the first stage of IMS, has already been deployed and was, at the time, the largest CORBA and Java-based system in the world (IMS is expected to support tens of thousands of subscribers downloading gigabytes of data around the clock)**
- **AT&T, back in 1998, was already working on about 30 separate systems which relied on CORBA**
- **Spanish telco Telefonica has deployed a dozen CORBA applications, the largest of which serves 1,600 operators and handles 1 million calls a day**
- **Swiss Telecom handles subscriber administration from 17 regional offices, using a CORBA application with Windows/COM front ends**
- **CNN Interactive uses Iona's Orbix, OrbixWeb and the OrbixTalk event service to distribute news material — from hundreds of sources, in many formats, from many different platforms — to its editors**
- **Cable & Wireless Internet Exchange (CWIX) provides network monitoring through a Web-driven, CORBA-based application**
- **Boeing has integrated four 'best of breed' manufacturing applications into a comprehensive infrastructure that manages aircraft configuration — from ordering, through manufacture, to maintenance; this enormous distributed system, which is critical to Boeing's business, is based on a CORBA communications backbone**
- **McKesson Corporation, North America's largest healthcare supply management company, deployed an application called InfoLink based on CORBA and the Web; reports that once took weeks to prepare can now be delivered in seconds, while software development and maintenance have greatly been accelerated**
- **Wells Fargo Bank developed a Web front-end called the Customer Relationship System (CRS) which went live as early as May 1995; CRS encapsulates the bank's legacy systems as CORBA objects which are made available to the bank's service agents through a Web server tier**

- **Charles Schwab built a CORBA-based trading application for their 5,000 best customers; this routinely handles accounts worth billions of dollars**
- **aiming to offer more personalization and better service, American Airlines redesigned its online reservation Web site in 1997-98; now Iona's Orbix ORB handles connectivity to the SABRE travel reservation system**
- **the GNOME project, which built and continues to enhance the eponymous graphical user interface for Linux, chose CORBA as the backbone for its various modules; to avoid unnecessary overhead, the developers wrote their own lightweight, efficient CORBA implementation (known as ORBit) which has bindings for over a dozen languages**
- **the OpenSpirit Application Integration Framework, based on CORBA and Java, was developed for the OpenSpirit petrochemical consortium by PrismTech**
- **the Port of Singapore, one of the world's busiest shipping nexi, typically hosts 800 ships at any one time; to allow for future growth the Port is automating its cargo handling process using Silicomp's APPLI-BUS middleware, which is based on CORBA.**

CORBA is, therefore, in production and is used by leading airlines, banks, insurance companies, healthcare providers, manufacturers, petrochemical and pharmaceutical companies, telecommunications and transportation providers, utilities and government departments around the world. It is not as dead as J2EE or .NET proponents might wish.

CORBA and Java: kissin' cousins

Einstein's advice to "make everything as simple as possible, but no simpler" is routinely disregarded by the media. Comparisons between CORBA and Java are a case in point. While these two sets of standards have much in common, they are intrinsically different in nature.

CORBA is a language-neutral system for remote invocation of software services, to which optional extras — like security, transactions and persistence — have been added. Recently, it has been extended to include the language-neutral CORBA Component Model (CCM). This is interoperable with Enterprise JavaBeans (EJB) which it closely resembles (see also **MIDDLEWARESPECTRA**, August 2001, page 38).

Java, on the other hand, started as a programming language and has gradually been enhanced to include:

- **remote invocation**
- **two different component models**
- **virtually all of CORBA's optional features**
- **and much more besides.**

However, Java remains language-specific.

Nevertheless, there is much synergy between CORBA and Java — synergy amplified by the way in which supporters of the two standards have made common cause. As early as 1995 they realized that a Java ORB could be downloaded to a Web browser as an applet (ORBlet), simplifying distribution and neatly fitting in with the Web world of HTTP and HTML. Then interest waned in client-side Java, only to flare up again when EJBs re-introduced interest in Java on the server.

For a while in 1997 it looked as though the CORBA and Java communities might split over the issue of whether Java developers should use CORBA's RMI or Java's RMI (which was brand new at the time). Java resented CORBA's complexity, which many viewed as unnecessary — especially when they did not intend to program in anything but Java.

Eventually an agreement was reached whereby the Java 2 definition included both forms of RMI over the native Java RMI Protocol (JRMP) and RMI-IIOP. The latter guarantees interoperability with CORBA servers written (theoretically) in any of the supported CORBA languages. Starting with version 1.2, a minimal Java ORB shipped with the Java Developer's Kit (JDK).

In the OMG's words, "while the CORBA solution has embraced Java, it has not done so at the expense of other languages. In fact, Java is the only language for which CORBA supports binary portability. For all other languages, CORBA is portable only at the source code level."

The bindings for other languages give discretion to the ORB developer as to how much code is generated from the Interface Definition Language (IDL), and how much is tucked inside the ORB runtime. For this reason alone, code generated with one ORB's IDL compiler is incompatible with any other ORB. (The same source code, however, can be used, provided that it is processed by the IDL compiler of the ORB under which it is to run).

Uniquely, the IDL to Java binding generates code to a fixed format which guarantees that it can be run under any compliant Java ORB. This was essential because the code might

have to be downloaded to a range of computers with different ORB run times, and would have to run reliably on them all.

In an effort to open up the Java platform to industry input, Sun introduced the Java Community Process (JCP) in December 1998. Since then the JCP has taken over the design of new Java specifications, and has itself evolved to JCP 2.0. It is broadly similar to the OMG process, except that Sun owns the Java technology and thus retains the final say.

Sun acknowledges the contribution to the Java platform — to J2EE in particular — of the OMG's specifications like IIOP, Object Transaction Service (OTS) and Java-IDL. Furthermore, an unremarked fact is that Java and CORBA provide the foundation for nearly all Application Servers that are on the market today.

In addition. many organizations are members of both OMG and JCP. There are several ongoing co-operative projects, including:

- **the OMG's embedded security model (which is included in the Java security architecture)**
- **the JCP's Unified Modeling Language (UML) mapping for J2EE.**

Interestingly, inveterate rivals IBM and Sun co-operated to develop the CORBA features of J2SE, EJB and J2EE. George Paolini, Sun's ex-VP of Technology Advocacy and Community Development, stated that: "OMG is critical to the Java platform, to the JCP program, and to cross-platform industry-standard approvals. Sun considers them to be an essential part of our success and standards adherence". In turn the OMG's chairman, Richard Soley, said: "these two sets of standards fit together hand-in-glove... OMG's technologies provide developer independence and systems interoperability across all platforms, while the Java platform guarantees portability of their code. You could not ask for better."

Finally, there is a substantial overlap between the 800-plus members of the OMG and the 300-plus members of the Java Community Process. Equally to the point, OMG members predominate in the JCP's two executive committees:

- **the J2SE/J2EE committee includes 11 members from the OMG members (out of 16)**
- **the Java 2 Micro Edition (J2ME) committee has 9 members from the OMG (also out of 16).**

The role of CORBA in Java specifications

Although CORBA's continuing influence in the world of Java should now be evident, it is far from easy to nail down the exact relationship between the two sets of standards at any given moment. Each (Java and CORBA) comprises dozens of separate specifications, many of which are being progressively refined or (occasionally) replaced by new and better ones.

Major reworks — such as CORBA 3.0 or J2EE 1.3 — often appear monolithic on first glance. In practice, however, they are actually made up of many individual specifications — each with its own version number. Furthermore, there is a significant time lag before each new major version finds its way into commercial products.

CORBA's influence is most obvious in three of the most important Java specifications:

- **J2SE**
- **EJB**
- **J2EE.**

Java 2 Standard Edition (J2SE)

Although J2SE 1.3 includes full ORB functionality, developers are free to use it or leave it. This contrasts with EJB and J2EE, which require that IIOP (at least) be used in certain circumstances.

RMI over IIOP (often referred to as RMI-IIOP) is an integral part of J2SE 1.3 (and J2EE 1.3). In previous releases it was provided separately. When used with the included Java to IDL compiler, it enables Java programmers to write CORBA services without explicitly using the CORBA development process. An IDL to Java compiler is provided.

The idea of RMI-IIOP is to let Java developers exploit standard RMI interfaces, while taking advantage of IIOP and CORBA's language neutrality at runtime. Subject to RMI programming restrictions, CORBA services can also be exposed and invoked through RMI-IIOP.

In contrast, JavaIDL is an ORB that supports all the features of CORBA required when programming occurs exclusively in Java. According to Sun it is superseded by RMI-IIOP for most purposes, as Java developers prefer to avoid working explicitly with an IDL. On the other hand, JavaIDL is free from the RMI programming restrictions that apply to RMI-IIOP.

The upcoming J2SE 1.4 (Merlin) will support CORBA's

General Inter-ORB Protocol (GIOP) 1.2, with several new features and bug fixes (IIOP is simply GIOP running over TCP/IP). J2SE 1.4 will also support CORBA 2.3.1, including the Portable Object Adapter (POA), Portable Interceptors and the Interoperable Naming Service. This will make it possible to plug in any ORB that implements CORBA 2.3 or later. In particular, the POA allows developers to write a CORBA server and implement it on any compliant ORB.

Enterprise JavaBeans (EJBs)

In the long-awaited EJB 2.0, adopted in July 2001, there was an increased emphasis on standards-based interoperability. In practice, this interoperability was delivered through CORBA.

Chapter 19 of the EJB 2.0 specification explicitly replaces Sun's earlier EJB1.1-to-CORBA document. It states that:

- **both client and enterprise bean containers must support invocations over RMI-IIOP using IIOP 1.2, thus making this the default communication mechanism within EJB (implementations may support other remote invocation protocols as well as IIOP); one of the main purposes of this requirement is to guarantee out-of-the-box interoperability between different J2EE products while another benefit is that CORBA clients written in any supported language — such as C++ or COBOL — can invoke methods on enterprise beans**
- **Java Transaction Service (JTS) specifies the implementation of a Transaction Manager which supports the Java Transaction API (JTA) 1.0 at the high-level and implements the Java mapping of the OMG's OTS 1.1 at the low-level; JTS is simply a translation of OTS into Java and in so doing it enables enterprise beans to co-ordinate transactions with ORBs as well as with each other; EJB transactions may be controlled through JTS, although this is not mandatory (whereas JTA and the Connector APIs are); vendors can choose whether to provide transaction interoperability or not and, if they do, their EJB containers must support OTS 1.2 (directly, or in the shape of JTS) whereas if they do not, the containers must be able to decline participation in OTS terms**
- **EJB containers must be able to publish object references in the format of the CORBA Interoperable Naming Service; this is a logical consequence of using RMI-IIOP as the default communication mechanism**
- **interoperable security is based on Conformance Level 0 of OMG's Common Secure Interoperability version 2 (CSIv2), which must be supported by all EJB, Web and application client containers; both SSL 3.0 and TLS 1.0 must be available as security protocols for IIOP and IIOP/SSL (for which IANA has reserved port 684) ensures on the wire privacy; client authentication can be carried out using certificates at the SSL layer, or by sending user name and password over IIOP/SSL.**

On a slightly different note, the basic level of OMG's new CORBA Component Model (CCM) is almost identical to EJB — with the difference that multiple languages are supported instead of only Java. The CCM specification includes a comprehensive forward and reverse mapping to EJB. CCM components can be mixed and matched with EJBs.

Java 2 Enterprise Edition (J2EE) 1.3

CORBA's influence reaches a peak in J2EE, which includes J2SE and EJB as well as a number of other specifications. CORBA interoperability is a mandatory part of J2EE. This means that J2EE application servers can talk to clients and other servers using CORBA as well as RMI — a capability that comes in handy when talking to legacy systems.

The mandatory uses of CORBA within J2EE are:

- **support for IIOP, through RMI-IIOP and JavaIDL (as in J2SE 1.3).**
- **EJB interoperability, through RMI-IIOP, Interoperable Naming, OTS and CSIv2 (as in EJB 2.0).**

There are also optional CORBA features (in J2EE), such as:

- **JTS can be employed for transaction management**
- **JNDI, the Java Naming and Directory Interface, can use the JNDI COS Naming Service Provider, in which case JNDI lookups will be fulfilled through the CORBA Naming service; this arrangement also makes it possible for CORBA applications to store object references in an LDAP directory**
- **the Java Message Service (JMS) is, broadly speaking, compatible with the CORBA Notification Service — a more powerful version of the CORBA Event Service.**

The latter is convenient because JMS specifies only an API, whereas Notification also specifies a messaging engine. In

effect, commercial implementations of the Notification Service — for example those from Fujitsu, Hitachi, NEC and PrismTech — can be plugged in with few or no changes.

In consequence, a J2EE application server using a Notification engine in the JMS role can exchange messages with CORBA objects as well as other J2EE servers. Meanwhile, the OMG is working to tighten up the integration between CORBA Notification and JMS.

Management conclusion

CORBA is an extreme case of a standard that offers little sizzle — just lots of steak (with optional onions, mushrooms, tomatoes, etc.). For well-informed IT professionals, this is not necessarily a disadvantage. Indeed, it enables them to focus on providing robust, secure, extensible and scalable distributed systems without being overly concerned about external preoccupations.

'Legacy' has become a handy label with which to deprecate everything except the particular products a given vendor is anxious to push. But what is a legacy? The immediate connotation is with technologies that are no longer being actively extended. Implicitly this seems to mean that the labor or skills pool is limited. On the other hand, a few people have come to realize that what really distinguishes a legacy system is that it can be taken for granted. In other words, it works reliably, come what may — unlike many of the latest technologies, which always seem to possess teething troubles.

Java, for instance, has been generally available for five or six years. This is now just about long enough for it to have bedded down, except for the fact that it has been (and is being) continually enhanced and extended. CORBA, too, is still a work in progress. Although it is more than 10 years old, the current version (2.4) was only finalized quite recently and Version 3.0 is still emerging piecemeal.

That said, it is CORBA's relationship to the legacy issue which is unique. In itself, it is one kind of legacy — technology that works reliably. But it is also a cost-effective way of saving other systems from becoming legacies in the bad sense of 'islands of isolated automation'.

This is why the symbiosis between CORBA and Java is so beneficial for all concerned. Java's greatest weakness lies in being new, different, and potentially incompatible with existing corporate systems. By weaving CORBA into the tapestry of J2EE, Sun has ensured that any potential incompatibility should not become an affliction.

Individually, CORBA and Java are two of the software achievements of the 1990s. Together, their strength multiplies — because each compensates for the other's weaknesses. It is entirely possible to program in Java without taking any notice of CORBA. It is feasible — though less likely — to design CORBA applications that have no Java components.

When a powerful, scalable distributed system must be built and deployed swiftly, without sacrificing connectivity to existing systems, a combined CORBA/Java solution can offer the best of both worlds. This is why CORBA and Java ignore each other only at their mutual peril.

Web Services are already here ...

Dr Keith Jones
IBM Worldwide Software Solutions

Management introduction

The topic known as Web Services is currently one of the hottest subjects in the IT industry. Web Services are expected to be the next logical step for exploitation of the Internet.

But is any aspect of Web Services yet real? What can be implemented now and what is needed to complete the technological picture? Initial releases of Web Services technology are available from several vendors. These already solve some of the problems encountered with previous architectures that had attempted to distribute application function.

In this analysis, Keith Jones looks at the many promises that have been made in order to assess how they (Web Services) might bring the expected benefits across the board — to business, government, consumers and other major Internet user groups.

Why, and why now?

To most people the Internet is synonymous with the World Wide Web and the hundreds of millions of Web pages now available for download via browsers. In reality the underlying networking fabric that transports those Web pages is the Internet.

Running on that networking fabric are a number of different standards-based applications:

- **the Web**
- **email**
- **file transfer**
- **voice interaction**
- **instant messaging**
- **and many others.**

The number and scope of applications is constantly expanding. Web Services, while not applications per se, are another manifestation of what the Internet is encouraging as additions.

As Internet protocols became established — on a wide variety of embedded, hand-held, laptop, desktop, work station and server hardware — the efficient electronic exchange of Web pages and other data has become the focus for:

- **standards bodies**
- **open source organizations**
- **software vendors.**

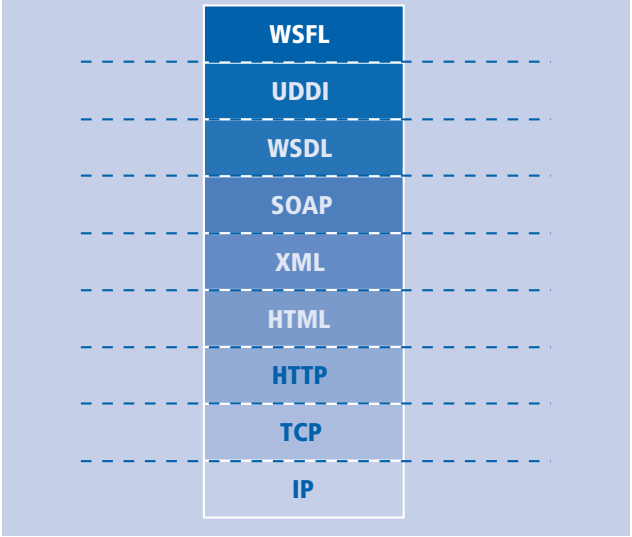
First came HTML. After that there was XML, then XHTML and XML over HTTP. These solutions have been maturing and are still evolving. XML, with its powerful typing and document structure capabilities, is already largely established as the markup language of choice for:

- **data exchange over the Internet**
- **the meta-data that defines that data.**

Some will argue that the Internet phenomenon is being driven largely by open standards and open source consortia. Certainly the result has been broad acceptance by solution builders and providers. In addition, the 'conceptual stack' (Figure 7.1) continues to evolve, as good a measure of success as anything else, and the lower (Internet) layers are now relatively stable.

In contrast, the upper layers have become an intense battleground for competing ideas, millions of marketing dollars and intense industry politics. In all fairness, Web Services may reasonably be described as being only the latest of such battlegrounds.

Figure 7.1: Conceptual stack



Looking for a distributed component architecture

For the longest time it seems the IT industry had been looking for a software component architecture that would deliver sustainable productivity gains and cheaper software. Several attempts — only partially successful — have come (and some have gone), including DCE, CORBA, COM/DCOM and J2EE. Each one, with its supporting architecture and implementation, made some progress. But most suffered limitations that have either not been resolved or have provoked a shift to alternatives.

Web Services is, arguably, the latest attempt at identifying a software component architecture that delivers re-usable software components. The hope is that, if Web Services components are defined in open-standard XML, which is hardware and software platform independent and programming language independent, there is a reasonable chance of the ever promised productivity quantum leap.

So what are Web Services?

Just as the Web is a service on the Internet, so too are Web Services. They (Web Services) are, however, focused on facilitating communications between applications that are connected via the Internet. Data exchange is potentially both more varied and complex than the textual presentation of data that occurs with Web pages.

Web Services not only incorporate open standard data exchange between applications but they also include mechanisms for publication and discovery of standard

interfaces for those applications over the Internet. Examples of anticipated Web Services include:

- **weather forecast acquisitions**
- **purchase order submissions**
- **command and control communications**
- **credit worthiness checks**
- **customer service requests**
- **gaming tournaments**
- **and many more.**

A wide spectrum of topics is, therefore, covered by Web Services technologies, embracing almost everything from a simple networking RPC mechanism to the most complex business process infrastructure. In truth, Web Services are now painted so broadly that they have become an enabling technology umbrella which covers the facilitation of interactions at electronic speeds between programs (applications) of all kinds that are connected to the Internet.

Some will, not unreasonably, argue that there is nothing new here:

- **browsers have been interacting with Web servers using HTML over HTTP for years using an RPC-like mechanism**
- **secure interactions between applications of this sort, using SSL encryption, have been carrying confidential data and supporting monetary transactions for a similar period.**

But there are differences. Web Services encompass much more, including:

- **improved content markup**
- **more reliable data exchanges**
- **encapsulated services**
- **service implementation varieties**
- **intermediary services**
- **discovery services**
- **interaction models.**

Improved content markup

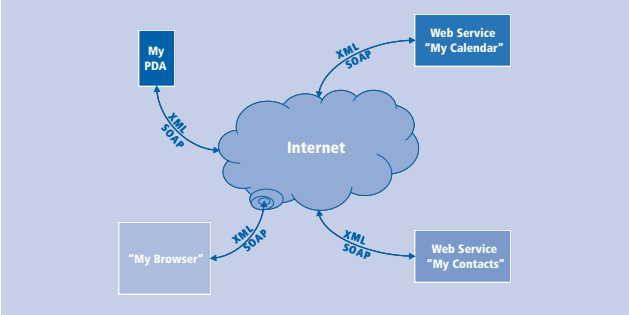
It is reasonably well known that HTML was loosely specified and that browser support for markup is inconsistent. This has lead to mixed Internet end user experiences. The HTML 4 and CSS (Covering Style Sheets) standards are improving that experience and XHTML has taken yet further steps toward consistency. But transport, interaction model and content markup have been at least as important.

XML may yet provide the best solution for content markup. Just as Java removed many of the C/C++ programming pitfalls, so XML may remove many of the markup pitfalls found in HTML.

But XML can deliver much more than this. It can provide a broader foundation for the exchange of data of all types between applications over Internet protocols.

To many, the flow of XML data over HTTP is a simple Web Service. An application of this can be seen in Microsoft's Hailstorm vision. This uses Web Services to deliver a consistent integrated browser experience for end users at a variety of hardware end-points (Figure 7.2), including workstations, laptops, set-tops and hand-held devices.

Figure 7.2: Personal data integration



My Favorites, My Preferences, My Calendar, My Contacts, My Wallet — every aspect of 'my personal environment' — may be retrieved from supporting servers in future, through the use of Web Services technology. This flow of personal information will be realized via XML element streams over Internet transports using the Simple Object Access Protocol (SOAP) proposed by IBM, Microsoft, Hewlett-Packard, SAP and others.

SOAP abstracts the process of data exchange as a sequence of one way messages. Each message carries a header for control information and a body for functional request and response data — all in a SOAP standard envelope.

The underlying transport for SOAP messages does not have to be HTTP. It could be, for example SMTP, Sockets, MQSeries, IIOP, JMS or even SNA. However, as you should expect, most discussion currently concentrates on delivering SOAP over HTTP.

Reliable data exchange protocols

SOAP requires that control information and request-response data be specified in XML for the payload of mes-

sages to be exchanged. This combination of technologies delivers a capable, secure and reliable data exchange mechanism that improves upon earlier HTML/HTTP implementations. SOAP requests can be as simple as parameters for an RPC — for example, as simple as a stock quote or as complex as a multi-megabyte XML document containing trading partner contract details and purchase orders.

Today, most of the current discussion about Web Services is about the simpler variety of SOAP requests. Over time this will progress as complexity increases.

SOAP not only defines the envelope for message data but also the encoding rules for the XML elements. Whilst encoding and decoding of XML data is hidden from developers most of the time, this function is an extremely important piece of Web Services technology. It is important at run time, for platform independence and performance.

Digital signatures and encryption technologies — when applied to SOAP XML document flows — not only secure confidential data but also open up the possibility of pay-per-use for software services as well as valuable content transactions. Licensing standards have been proposed to support and extend this concept.

As browsers begin to make SOAP requests for Web Services, as well as normal HTML/HTTP requests, it is easy to see that the end user experience will be enriched with both depth of content and new function occurring because of Web Services. This will apply as much to consumers as employees and enterprise end users alike.

Standalone applications running on embedded, hand-held, laptop and workstation devices will also access Web Services in due course. It is interesting to note, however, that more and more client systems are now browser-based and are becoming 'thinner'. Web Services confirm this direction, by enriching the sources for function and content available to those browser clients.

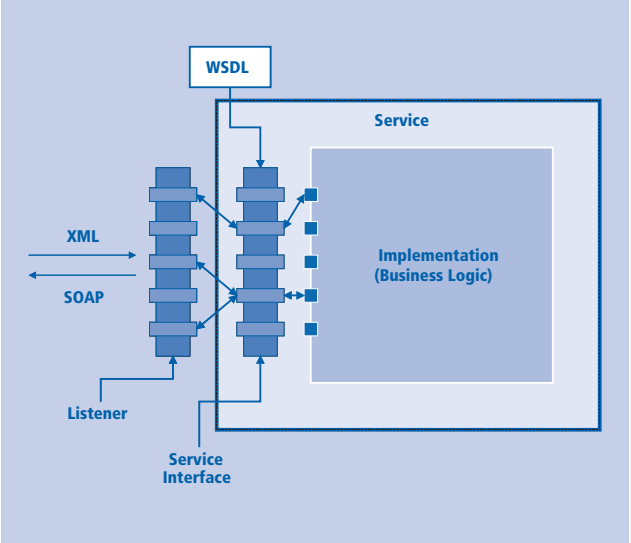
Well encapsulated services

It is not sufficient, however, for Web application designers to know how to construct SOAP XML messages to invoke Web Services. A standard Web Services Description Language (WSDL) is needed. This has been proposed by IBM, Microsoft, Hewlett-Packard, SAP and several other vendors.

WSDL formalizes the encapsulation of Web Service interface and binding details — as (surprise, surprise) XML documents. These are then independent of platform, operating system and programming language (Figure 7.3):

- **a Web Service interface description contains operations with the expected input, output and fault message contents**
- **a Web Service binding contains details of protocols and ports where the service may be accessed on the Internet.**

Figure 7.3: An encapsulated service



This encapsulation enables the separation between Web server and Web Service provider machines. It also provides the flexibility to configure different Internet access paths to the same service, where needed.

If Web Services exist on server machines attached to the Internet behind well-formed standardized interfaces they are logically equivalent to the business logic and data access components that lie behind most existing Web applications; the presentation component is unspecified and left to the Web page designer. This may suggest a rationale for future Web application structures, which can serve both Web pages and Web Services.

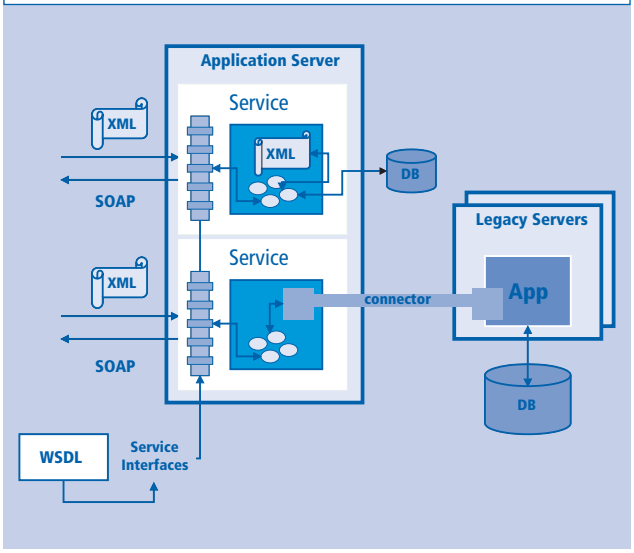
Variety of service implementations

The implementation of Web Services function behind encapsulating interfaces can take many forms. New function can be implemented in Java, to the J2EE Web application architecture or in VB (or C#) to the .NET architecture or in other forms. As in all good object oriented design, Web Services interfaces are designed to be separate from their implementations.

An interesting design issue to be answered for Web Services is the question of how far 'back' XML elements need to flow into the business logic and even into the databases supporting the service. There is no requirement for XML to flow back beyond the SOAP-server interface. But it may be convenient, for other reasons, for new business logic to be constructed around XML document trees. A major consideration in making this choice is the implied need for XML application programmer skills.

Existing enterprise application function can be exposed as Web Services by connecting to it and adapting the interfaces to correspond to a WSDL description (Figure 7.4). This is probably done best by re-using an Application Server that already provides good connections to legacy applications. ERP and CRM functions are typical of those likely to be exposed in this way.

Figure 7.4: Legacy business service



There is a further design issue have for those Web Services that are to be based on existing application function. Legacy business logic was almost certainly not designed for programmed access over the Internet. This suggests that some functions will need to be masked while others will need modification or to be added — to accept the XML parameters. The most obvious issue is, however, how you deliver compensating actions.

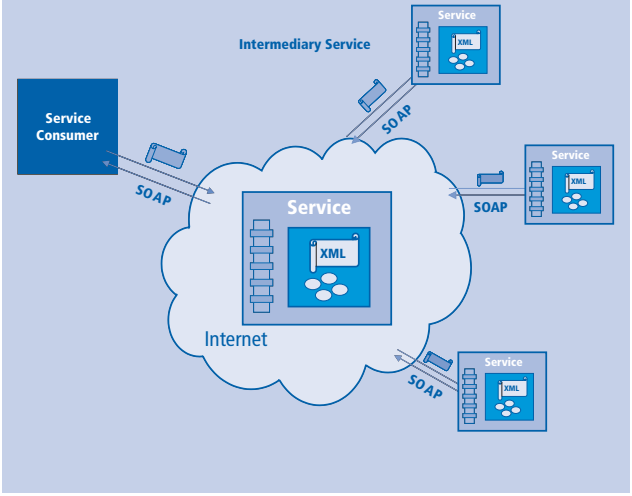
Java access beans (implemented as Servlets or Session EJBs) already provide a convenient mechanism for building wrappers and adapters for existing logic. IBM's WebSphere Studio Application Developer tools and Application Servers are already available for this purpose, as are approximate equivalents from the likes of BEA, Sun and others.

Intermediary services

The full usefulness of Web Services is not yet completely exposed by a collection — however large — of Internet end point services based on either new or existing business logic. Aggregating services into Web sites — acting as portals and e-market places — provides a convenient way to compare similar products and to use related services for a business purpose. Car parts markets and travel agency sites are obvious examples.

Other types of intermediary sites will evolve quickly using Web Services as this technology becomes more widely exploited. Web Service Brokers will route requests to 'service servers', based on required characteristics. These will offer added value — such as store-and-forward, logging, pre-authorization and non-repudiation — and will compensate in part for the current unreliability of Internet communications (Figure 7.5). Working examples can already be found in Grand Central and Slam Dunk Networks. There will be others.

Figure 7.5: Intermediary Web Service



Of course some Web Services will be implemented by accessing other Web Services (through intermediaries, as required) to fulfill business needs. This will be important between Internet partners — but may also be important within enterprises where applications on heterogeneous platforms must be integrated following organizational restructuring or acquisitions.

Discovery of services

For Web Services to achieve their full potential it must be possible for would-be service consumers to discover new ones (Web Services) as they are made available on the Internet. UDDI was formulated and proposed by IBM and Microsoft (with others) to provide a standard mechanism to:

- **publish service details**
- **discover service details**
- **access services using a global service registry.**

This information is held in UDDI registries and falls into three main categories:

- **business definitions**
- **service definitions**
- **pointers to detailed specifications.**

Service descriptions are not physically held in the registries but pointers to these details are recorded. IBM, Microsoft, Hewlett-Packard and have already set up UDDI registries for public access. Others will follow. These form a network of service registries with links for mutual replication of their contents and updates, which is performed on a regular basis.

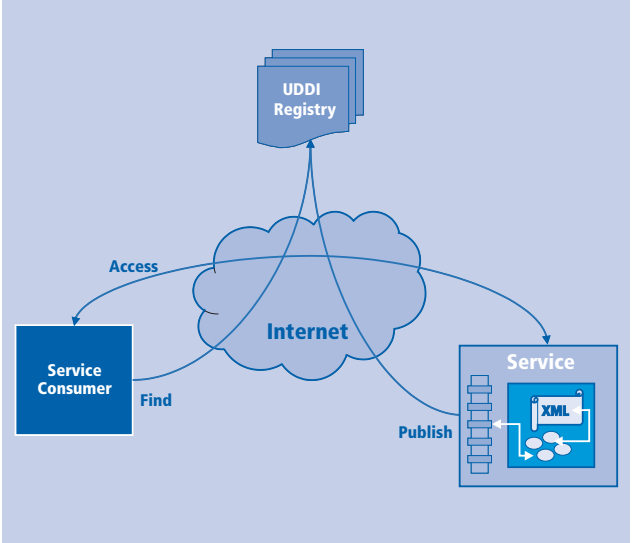
There are several ways in which such UDDI registries will be used. Within trusted enterprise networks private service registries will provide a local inventory of services available for integrating into new business logic. These registries may not be connected to external sources. But, over time, they may grow to become a catalog of re-usable function within an organization.

Programmers developing new Web Services will be able to publish their service descriptions to the UDDI registries. By doing this, others can discover their details when they need that information.

Development tools will hide much of the complexity required to generate well-formed Web Service description (WSDL) documents. These tools will also hide much of what is needed to generate Web Services access code and frameworks for new Web Services in implementation languages of choice. Indeed, IBM (in WebSphere Studio) and others are already delivering such capabilities.

As Web Services technologies and their related applications become more sophisticated it will be possible to access service registries at run time and dynamically choose an available Web Service from a number of comparable Services (Figure 7.6). While this is the vision often cited for Web Services, it is not yet ready for full exploitation. But it is coming.

Figure 7.6: Web Service discovery



As the number and variety of available, reliable Web Services increases it will become possible to create business logic that composes new revenue-generating products through dynamic collaboration with business partners over the Internet. Enterprises will probably choose to provide those Web Services for which a core-competency is held. They will probably rely on partners to provide other related Web Services, as this scenario becomes a reality.

More sophisticated interaction models

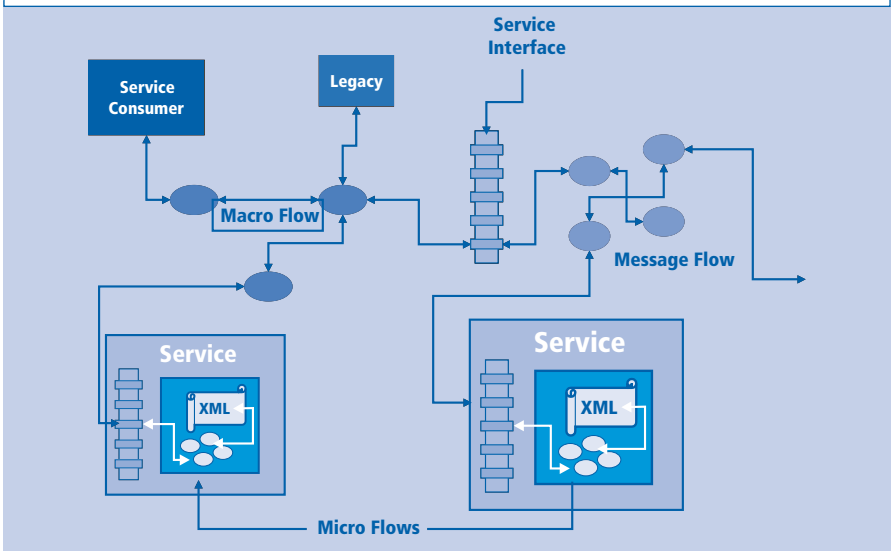
Most of the current focus on Web Services depends upon a very basic RPC interaction model between participants. However, most people recognize that a wide variety of interaction models must be supported if Web Services are to be successful in supporting the broader integration of business processes.

Many business to business flows are fundamentally asynchronous in nature and the number of different interactions needed to complete a transaction of any significance complicates many. Web Services based on SOAP messages flowing over asynchronous transport networks will evolve to support this need.

Several different levels of granularity can be identified in this discussion of business flows. At the top level, business processes can be modeled using state-full work or macro

flows (Figure 7.7). These may incorporate component Web Services, legacy systems and human interactions to reflect business at the highest level. Industry groups and standards bodies — such as ebXML and OASIS — have already recognized Web Services as playing a key role in this space.

Figure 7.7: Web Service flows



At a lower level, message flows can be used to model the detailed interactions between components composed to satisfy specific business needs. Web Services participate at the message flow level as XML documents are exchanged, transformed and processed — either at intermediary nodes or at service provider nodes in a network (IBM's MQSeries Integrator, for example, already delivers most of the function needed for this level of flow).

At the lowest level micro flows are used to model element interactions between components that implement Web Services. These might include flows between:

- **business and data access logic**
- **Web application logic and legacy systems (through connectors)**
- **service logic and other enterprise systems through integration servers and the like.**

WebSphere Application Server already delivers most of the function needed for this lowest level of flow.

Macro flows are state-full and typically reflect interactions between large systems both within enterprises and between business partners. Message flows are usually state-less and bridge the gap between macro systems and elemental application micro flows running in Web application (and other) servers.

A Web Services Flow Language (WSFL) has been proposed to formalize the description of both inter- and intra- enterprise flows and make them accessible in open standard format as XML documents. Development tools are expected in the coming months that will create and manage WSFL descriptions and facilitate the generation of participant run-time code as needed.

In future, therefore, Web Services are expected to participate in macro, message and micro flows. Such a broad base provides a rich diversity for business systems implementation both within the trusted networks of large enterprises and between partners across the Internet.

The real value of Web Services technology will emerge as a mix of:

- **an alternative standards-based integration technology for enterprise applications**
- **the basis for new revenue generating offerings**
- **the means by which competitive advantage on the Internet is delivered.**

Web Services middleware exists

Already several implementations of Web Services middleware are available from leading vendors. There are broadly three main categories of solution appearing, those based on:

- **extensions to the widely accepted Java J2EE architecture**
- **Microsoft's .NET architecture**
- **existing (or new) open source projects.**

Others may yet emerge.

Key Web Services technologies — like XML, SOAP, WSDL and UDDI — are available, most often as add-on function to existing middleware. This is good news because it suggests that Web Services are more evolutionary than revolu-

tionary. Such an approach makes Web Services easier to introduce: they are not 'all new'.

IBM's WebSphere Studio and Application Server middleware (at least in its Version 4 release), for example, can be used to deliver production level Web Services capabilities. Similar initiatives have emerged or are emerging from Oracle, BEA, Sun, Microsoft and other vendors.

New Services are already being built from existing legacy or Web application business logic as a relatively uncomplicated evolutionary step using these products. IBM's WebSphere and DB2 solutions are based on extending J2EE. The same is true for Sun's iPlanet, Oracle's 9IAS, HP's NetAction and BEA's WebLogic products. In addition, the Java Community has defined programming interfaces for convenient access to XML documents and processors, SOAP flows and Web Service registries that will be included as standard in future Java packages.

In parallel, Microsoft continues to extend and develop its Visual Studio, SQL 2000, Biztalk 2000 and other Server products to support .NET. Much of this is semi-available but .NET is expected to:

- **embrace fully the WSDL interfaces**
- **exploit the SOAP XML flows used to construct Web Services**
- **use UDDI**
- **inter-operate with other Web Services implementations when using standard Internet communications.**

In addition, from the Open Source category, the Apache Software Foundation is delivering several key pieces of Web Services technology as extensions to existing related open source projects. Jakarta Java projects and Apache SOAP, Xalan, and Xerces XML projects are all contributing software from a variety of different sources.

As more of these Web Services implementations are delivered there will be more and more choice but also a greater dependency upon the open standards to deliver practical levels of inter-operability. Tools focused on Web applications, including Services, will also be critical for every aspect of design, development, debugging and deployment of the Web Services vision.

The Web Services future

The forecast from most leading analysts indicates that Web Services technologies will be adopted in the next 3-5 years. Such penetration presumes that they will produce business benefits by generating value via cost savings, new revenue opportunities and/or delivery of competitive advantage.

Interoperability between implementation platforms and solutions will be critical to success. But that is not all. Equally important will be the delivery of several missing (or incomplete) technologies that will be needed for full scale adoption of Web Services. These include:

- **security**
- **transactions**
- **licensing**
- **versioning**
- **as well as resolution of the commercial issues associated with contractual arrangements.**

The good news is that there are many standards bodies, vendors and users already focused on resolving them. Surprisingly, perhaps, the future for Web Services is straightforward if the development of supporting technologies proceeds as expected.

Like most other Internet 'killer' applications, Web Services will be absorbed into the fabric of everyday business around the globe. They will then cease to attract much attention. They will have become successful.

Management conclusion

Web Services have emerged as an opportunity for the IT industry to tie several distinct technological threads together:

- ***Internet technologies, particularly those which have become widely established as the underlying network fabric for Web applications***
- ***Java, which has become the most widely distributed skill amongst programmers and J2EE the most often implemented architecture for Web applications***
- ***XML, which is fast becoming the most widely accepted of standards for data exchange amongst platforms of all types***
- ***SOAP, WSDL and UDDI — which are providing platform, operating system and language independent solutions for secure efficient communications between enterprises.***

More needs to be done to complete the spectrum of technologies required for Web Services to fulfill their promise. But enough is already available, including in accessible middleware products, to start exploitation with a reasonable expectation of decent returns over the long haul.

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 1460-7220