Volume 15, Report 3

MIDDLEWARESPECTRA

incorporating Enterprise Middleware

Contents **MIDDLEWARE**SPECTRA — **August 2001**

Adding EJB to legacy — to create applications appropriate for 2001

Peter Willson,
Vice President, Research and Development
Walker Interactive Systems

Management introduction

Peter Willson is the Vice President responsible for Research and Development relating to enterprise service technology for Walker Interactive Systems (Walker) — the software company headquartered in San Francisco which specializes in cross industry applications (such as General Ledger, Accounts Payable, Purchase Order and analytic software) for large organizations. Walker's customers come from all industries, with specific concentrations in:

- *banking and finance (for example, the Halifax Bank, Abbey National and the Royal Bank of Scotland)*
- *airlines (for example, United, Quantas, Cathay Pacific and British Airways)*
- *education (for example, UCLA).*

Other notable users include the United States Postal Service, the Royal Mail, Nabisco, Frito Lay and AT&T.

Traditionally Walker's applications have been mainframe-based, using CICS. In this discussion, Mr. Willson examines the steps that Walker has taken — and is taking — to update and modernize its applications through

use of Enterprise Java Beans (EJBs) running on application servers (like WebSphere and WebLogic). As he shows, this has also enabled Walker to deliver customized solutions faster to those customers who need special features. In effect, EJBs and related technologies have enabled Walker and its customers to continue to exploit their legacy code while simultaneously introducing new capabilities.

A changing customer environment

We have been operating in the mainframe marketplace for over 20 years now. With this as a base, I can reasonably characterize our customers as being at the more complex end of processing and with a clear need for business transaction integrity. This has been achieved through exploitation of IBM's S/390 (or, as we are now supposed to call them, the z-series) systems, which are usually found at the heart of most of our customer's operations. That said, mainframes are not the only systems that these customers own. Nearly all of the customers, that I have recently talked to, possess a highly diverse technical infrastructure, with a vast mixture of different offerings from different hardware and software vendors.

On the other hand, if you look at the technical landscape, one of the aspects that also characterizes most 'business transaction processing' — at least until a few years ago — was that it was predominantly driven from a character-based terminal interface, most usually the IBM 3270 model. Even after PCs were introduced, with sophisticated graphics capabilities, 'green screen' emulation was deployed on those PCs (to access mainframe applications). This was because those applications, including ours, had been designed around 3270 support.

On a business level, another common characteristic used to be functional specialization. Applications were designed for use by dedicated departments within large organizations. For example, anything to do with Accounts Payable would be handled by an Accounts Payable Department with dedicated Accounts Payable staff who had been deeply trained in both the processes as well as the applications which delivered the Accounts Payable functions.

This specialization has started to break down. One driver was the desire to extend selected function out to people in the wider areas of the business. An early manifestation of this was the introduction of client/server technology — usually with a Windows-based thick client equipped with an alternative user interface (to the 'green screen') which was connected to the mainframe running the 3270 application. Both the 3270 and the alternative user interfaces continued to be supported because it was financially inefficient to replace all the 3270 screens or to retrain all the 3270-familiar staff.

The Internet arrives

In the late 1990s the Internet phenomenon hit large organizations. The desire to provide a Web Browser-based interface over an Intranet or Extranet attracted, as businesses looked for a simpler client deployment model which might also provide access to existing applications to an even broader group of users: some of these users might be within the organization while some might be outside. Enabling customer, or supplier access, was seen as a way to make an organization more efficient. The result was a move away from the Windows-based client models to a Windows-based Web Browser model, which is inherently simpler to deliver and maintain (through use of downloadable Java-applets, etc.).

In the mainframe world, where Walker operates, the first deployments were client Browsers which connected to traditional CICS applications using simple screen scraping technologies. In effect, these delivered 'green screen' presentation through the Web Browser: the underlying COBOL or other transaction applications built on CICS did not change. We were basically taking processing that was already in place and re-presenting it through the Web Browser interface over an Intranet — in parallel to the 3270 interface over SNA.

The attraction of this was that it increased the number of people who might access an application as well as, in some instances, making an application somewhat easier to use — because you could introduce pull down menus rather than needing to know all the obscure codes which are so often a requirement of character-based applications. On the other hand, if users had not been trained to use the functions (as had the Accounts Payable Department staff), there was a real risk that such users could not know what they were supposed to be doing.

Win-win at a major airline

Another way to look at the challenge is to consider

the connection between core business operations — which are the investments that we and our customers have made in CICS that are still, today, running most large businesses — and third parties, whether customers or suppliers (Figure 1.1). Increasingly our customers were telling us that they wanted to provide better access to their core systems for their customers and suppliers.

Let me give you one example of this. A world leading airline (which wishes to remain anonymous) is a complex organization with complex support issues. It has a large number of staff who deal with invoicing queries from suppliers — where the airline has ordered something and the supplier provides it and sends an invoice. All too often payment may not be as forthcoming as the supplier wishes: the range of possible reasons is extensive, from mismatches on delivery, invoice to purchase order price mismatches, quality of delivery errors, mishandling of paperwork ... or whatever.

For an airline, handling and resolving all these queries is a substantial expense, and consumes a lot of highly skilled people. Anything which might improve resolution of queries was going to be attractive, but not only for the airline. Suppliers would likely be happier as well.

To address the overall issue, we offered a product (written using our new technology, and I will discuss this aspect later) which enabled the airline's suppliers to access its existing CICS-based purchasing system over the Internet via a Web Browser. Using this Web Browser, a supplier can see:

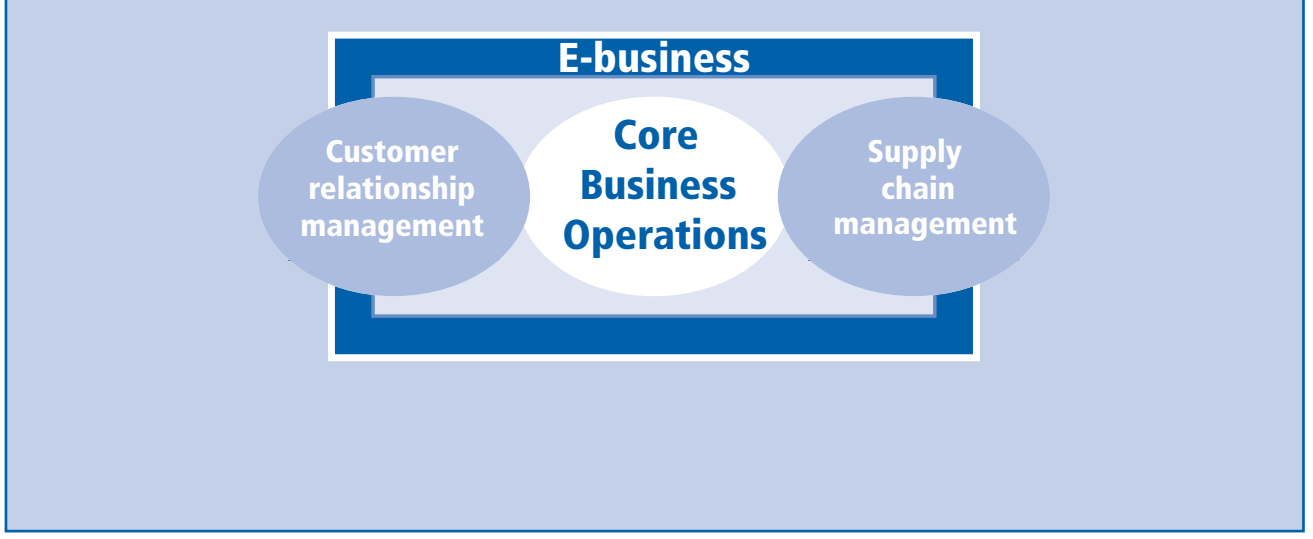
- **the status of all invoices sent**
- **when payment is scheduled**
- **what any problem is (should there be one).**

Just by knowing these details, the supplier can determine what to do. If there is (say) a price mismatch or a quality mismatch, the supplier can initiate the actions to solve the problem. If it was a delivery that the supplier thought had been made, but the airline has no record of, it can investigate. If payment is scheduled, they do not have to call the airline.

The result is a win-win situation:

- **the airline needs less purchase order support staff because suppliers are helping to resolve their own queries**
- **suppliers can see what is happening and, by initiating actions earlier to resolve issues, they receive their payments sooner.**

Figure 1.1: Core business and customers/suppliers



One extra benefit

Plus, there is one other benefit. The application that the airline has made available is its original Walker one, albeit with a different technology front end. The airline has not had to change either application code or training. Yet is has been able to extend its usage to a whole new set of users.

This is, therefore, an example of how to extend the use of a 'legacy' application. But we could not have achieved this had we not incorporated new technologies and taken the time to understand that we could not simply take an old 3270 application and just give Browser access to it. We had to do something more. This prompted us to look for a new way of building applications that would be relevant to electronic business to business relationships.

The importance of layering

Before adopting any particular technology, we investigated. Among many other considerations we examined our own existing application structure. Looking back, this had always been highly layered (Figure 1.2).

We had already:

- **separated the control of dialog from business logic**
- **implemented a structure where each of these could be driven by a message that was essentially protocol independent**
- **arranged for a message, with all the business data in it, to come in and drive one or more processes in the form of a business service.**

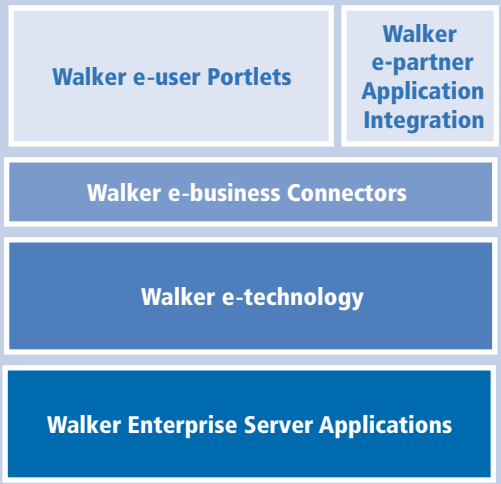
After our exploration, we realized that we needed to:

- **incorporate these business transaction processes into new applications**
- **provide access to business data (where business transactions were not required).**

Using this as a base, we have built a platform consisting of two broad functions:

- **one is access to the relational databases (DB2); access is possible directly or from applications written in another language using JDBC or ODBC technology — and we made the decision to provide this on a read-only basis (for reasons of integrity, we decided that whenever we wanted to update databases we would use the existing CICS COBOL business processes)**

Figure 1.2: Separation of function at Walker



- **the other comprises the packaging up of the CICS business processes into 'services' that can be driven by the presentation of a message (technically, coming in through a CICS COMMAREA).**

In addition, we decided that we would:

- **create the new application function using 'application servers' (typified by IBM's WebSphere or BEA's WebLogic)**
- **do our programming in Java, for portability.**

This all made sense in a number of different ways. As I mentioned earlier, most of our customers operate many different kinds of platform, including S/390, UNIX and Windows NT. Indeed, our experience is that most intermediate servers, like those running application servers, are located on UNIX or NT size machines. In such circumstances, Java and application servers were the obvious location for our new code (especially if Java would really run in most places).

Integrity, scalability, performance and architecture

Walker's customers are large. Business transaction integrity is essential, if only because there are such large volumes of transactions flowing through each day (just think of all the passengers carried and logistics commitments made each day by global carriers like United Airlines or Cathay Pacific).

Transaction integrity matters. Our customers always demand a high degree of integrity. Any new applications that we built would be updating operational databases. These are the ones that our customers use to run their production systems.

Equally important, any new applications (and application function) had to be highly scalable. Here we differentiate between scalability and performance. To us:

- **scalability means the ability to take an application and increase the numbers of users — potentially, from hundreds to thousands or even tens of thousands — without the performance for any one individual user degrading**
- **performance is the measurement of speed or throughput overall (rather than that experienced by each user).**

Sorting the balance between these two is almost always complex. After some degree of research (and using our relationship as a development partner of IBM), we designed an architecture which would make use of three main aspects:

- **Java**
- **Enterprise Java Beans (EJBs)**
- **the 'Model View Controller' design pattern.**

In practice it is the latter which provides the scalability. It is where we:

- **separate out the dialog aspect of an application: we put this into a Java Server Page (JSP)**
- **place the control aspect, the 'traffic cop' piece that decides which functions are going to run where, into a servlet**
- **take the business logic (which may be doing algorithmic processing in Java or updating DB2 tables using CICS services or accessing DB2 using JDBC): we put this into Enterprise Java Beans.**

The advantage was that we could now deploy a multi-layer physical architecture which could exploit the many different forms of platform available in our customers' IT portfolio — whether:

- **edge servers**
- **HTTP servers**
- **JSP engines**
- **multiple EJB servers**
- **etc.**

It is this which gives us a huge amount of capability, including both vertical and horizontal scaling for our applications. Yet, we have still not had to re-develop our existing CICS applications: we have left these applications unchanged (although we were significantly assisted by our earlier design decision that had insisted that applications be built to very granular requirements). The effect is that CICS applications have become 'back office' services for the EJBs or the servlets or the JSPs.

To deliver we use a common connector framework. This is a Java API that allows you to build a CICS COMMAREA and submit the contents to CICS in order to run the CICS transactions from the data contained. We generate this COMMAREA ourselves as a Java class — the Java class is the representation of the mapping — so that the back end remains unchanged, as CICS services. We could do this because our programs were already architected to be driven from COMMAREAS (although there were other ways of achieving the same objective of bridging between the Java and COBOL world, depending on the interface expected).

With this capability in place, we could now start to build new applications in Java while making use of existing COBOL code where it was already running in CICS. This was what our customers wanted. They did not have to throw out their CICS applications and completely start again. Instead our approach meant that they could continue to use these as the proven and dependable core business systems while still adding new capabilities.

Using EJBs

In practical terms, the starting point (for the user) is an HTML page (Figures 1.3-1.5). For example, a user might enter some data associated with a request into that HTML page which, via HTTP, is communicated to a Java servlet.

The servlet receives the HTTP, determines what actions to start and:

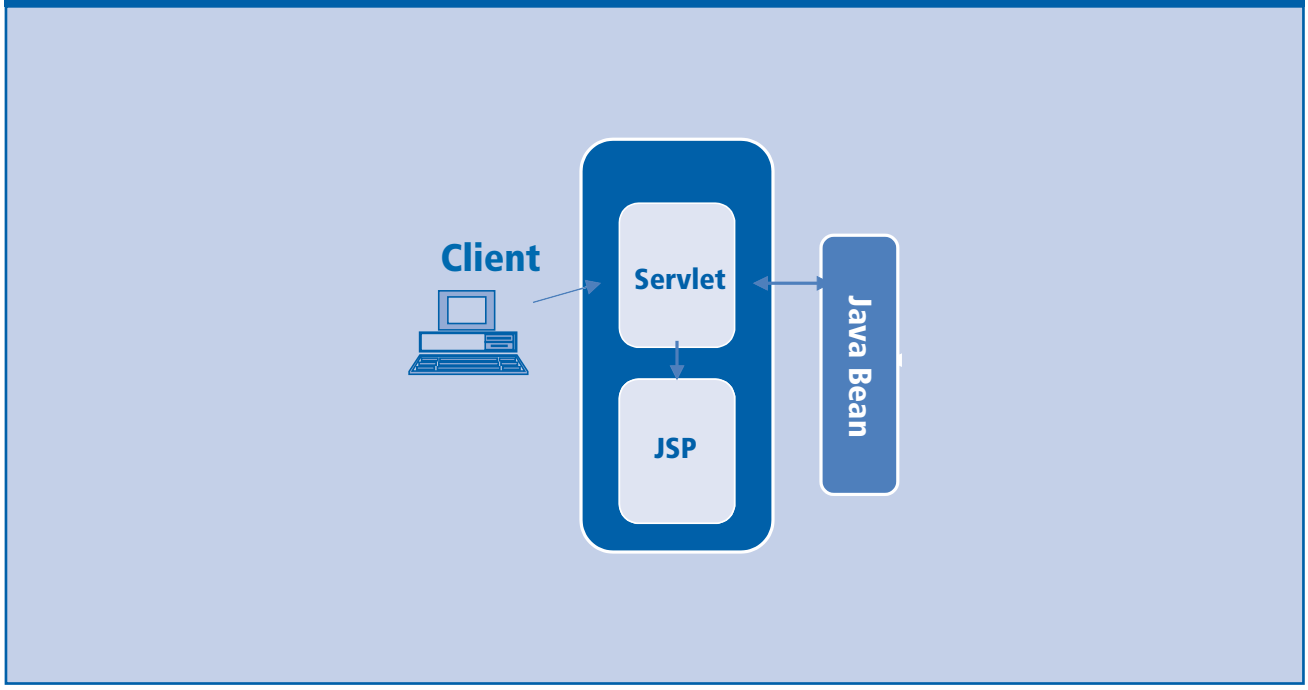
- **takes the information about the request out of the request object**
- **puts it into a Java bean**
- **sends the Java Bean to an Enterprise Java Bean server (currently we use three of the four types of EJB, depending on the specific needs).**

The EJB then makes use of the existing CICS transactions (say a database request) by:

- **instantiating an object**
- **using Java setters to set the properties of the particular object**
- **exploiting the CICS Transaction Gateway to communicate the contents of that object to the CICS server application.**

When the response returns from the CICS/database interaction (or from ODBC or JDBC, if it is a read-only request), the response goes back to the

Figure 1.3: Use of Java structures I



EJB. The EJB updates the relevant Java Bean which, in turn, goes back to the servlet. The servlet forwards the reply to the Java Server Page, which literally translates it into HTML for communication to the client over HTTP.

While this may seem complex, its attraction is that it is highly flexible. Bear in mind that, because of the way Java and EJBs work, the servlet and the JSP might be on one machine while the Enterprise Java Bean server might have been deployed on a separate machine in a different location — or they could all be on the same machine.

What is so attractive is that the technology handles all this. RMI over IIOP takes care of serializing the contents of the Java Bean, the sending of it (using the Name Service Directory to locate the necessary resources) and returning the information to the correct destination.

This is the essence of the framework we have built. Yet, there are many variations on the broad theme, largely depending on the application and whether we are using entity EJBs or stateless sessions, or whatever.

Entity and session beans, and application servers

In effect, what we deliver are CICS programs running in an MVS/CICS/DB2 environment without change. To these applications we connect applications (EJBs, servlets, JSPs, etc.) which use Java.

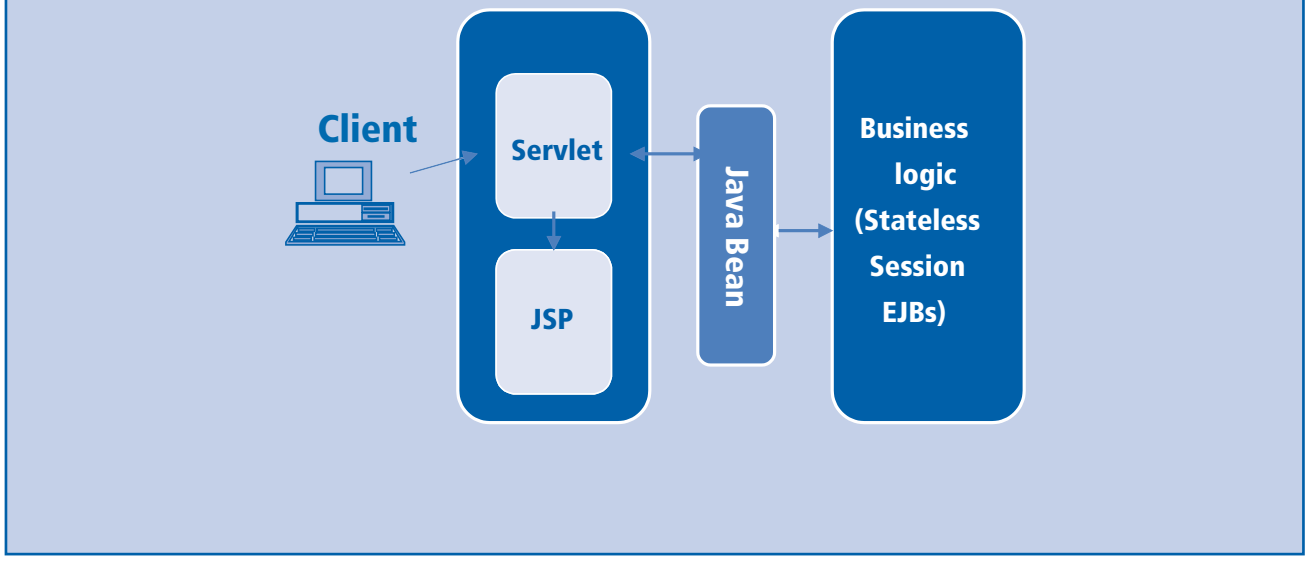
One major concern for us was deployment when development finished. As I said earlier, a principle attraction of Java should be its ability to run on any platform that supports it (which includes most platforms today).

If we take our 'Java applications' — each of which is the sum combination of EJBs and Java Beans and servlets and Java Server Pages — we have already deployed these Java applications on a number of different platforms in a tight timeframe, including on:

- **WebSphere Application Server, Advanced Edition 3.5 on Windows NT4, in June, 2000**
- **WebSphere Application Server, Advanced Edition 3.5 on Solaris, in August 2000**
- **WebSphere Application Server, Advanced Edition 3.5 on AIX in September, 2000.**

In December last year (2000) we ported to the new EJB container that implemented the CICS transaction server 2.1 (Figure 1.6) even though this supports session beans only (not entity beans); this

Figure 1.4: Use of Java structures II



was in Beta at the time. Since then we have ported to WebSphere Application Server 4.0 on both OS/390 and z/OS (and Linux, for WebSphere Application Server, Advanced Edition 3.5).

Of the four applications that Walker announced last year, only one — a shopping cart application used by the Halifax Bank in the UK — makes use of entity beans, in this instance to store the state of the shopping cart. This application cannot be implemented in its entirety within the CICS EJB container — because CICS supports session beans but not entity beans. While some may argue that this lack of entity bean support in CICS is a problem, we do not see it that way. Instead, all you need is an application server to provide the entity bean support.

This is little different to our other three EJB-supporting applications. They use session beans and can be implemented entirely within CICS — except that you still need a JSP and a servlet engine and CICS does not provide these either, or a directory naming service. Given that this is the case, adding an entity bean server is no big deal — despite what the purists might suggest.

We took our applications, therefore, and with virtually no changes, we have successfully implemented these with the CICS container. The consequence, besides re-use of existing code, is the scalability (and performance) that our customers need. This occurs because the EJB structure enables Beans to 'get relatively close' to the CICS programs, which delivers the speed. We have thought of going still further, for example by using J-CICS. The problem for us about this, however, is that it would bind that application irrevocably to the S/390 and CICS combination, a platform constraint our customers do not want.

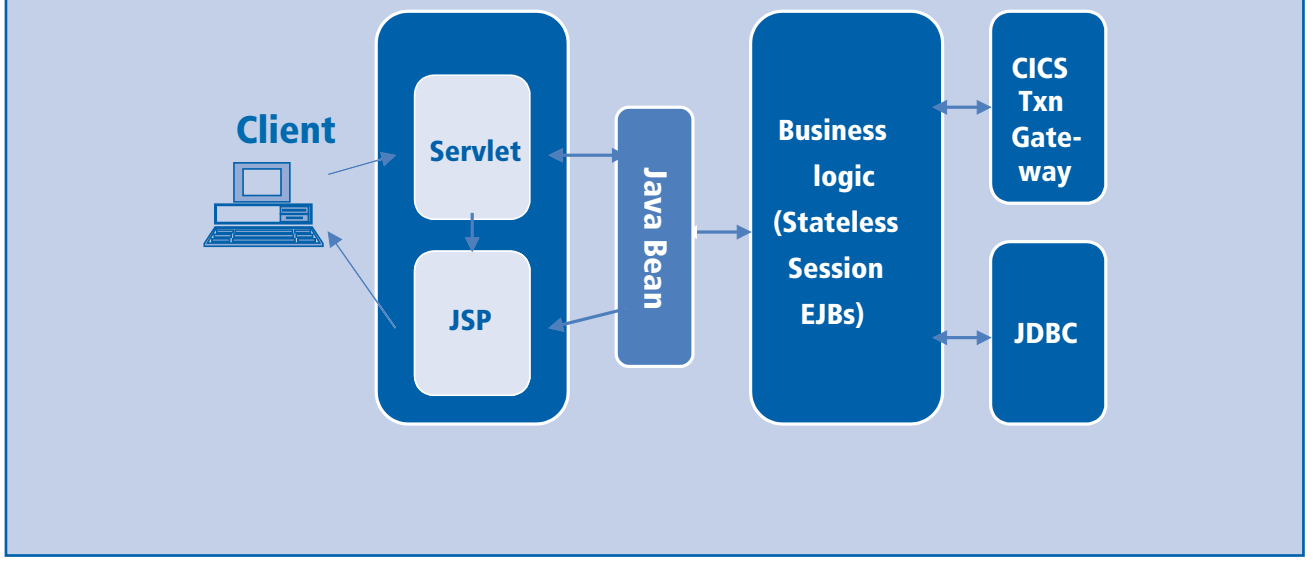
Portability and platform flexibility

What I did not mention earlier is that we have proved that applications built in Java (in the broad sense) for one application server can run on others. We have one customer who wished to deploy onto BEA's WebLogic application server. We did not have to change any of our application code.

What we did have to do is create 'deployment descriptors' which — in the implementation coming from WebSphere — are in the form of a serializable file. WebLogic prefers XML. All that was needed was a conversion of the WebSphere serializable file into XML, which was not difficult. (Indeed, IBM plans to move its deployment descriptors for EJBs to XML in due course.)

On the basis of what we have seen so far, portability and application flexibility exist across mainstream application servers. But you do still have to

Figure 1.5: Use of Java structures III



be careful about the deployment details (rather than the application code itself).

Lessons learned

Our experience with Java and EJBs and CICS has confirmed the validity of one decision that we made at Walker many years ago. This was to layer function and to keep those different functions separate. If I look back I can see how we started with a very simple server model and, as we grew, this became ever more complex.

By taking the layering approach we then prevented ourselves from ending up with so much complexity that only a total rebuild would have worked. That we have not had to do this is absolute validation of the value of layering and separation. It is what has enabled us to move forwards with Java and EJBs while not throwing out what already worked. Both we and our customers have benefited.

The use of EJBs and Java — with their separation into JSPs, servlets, Java Beans and EJBs:

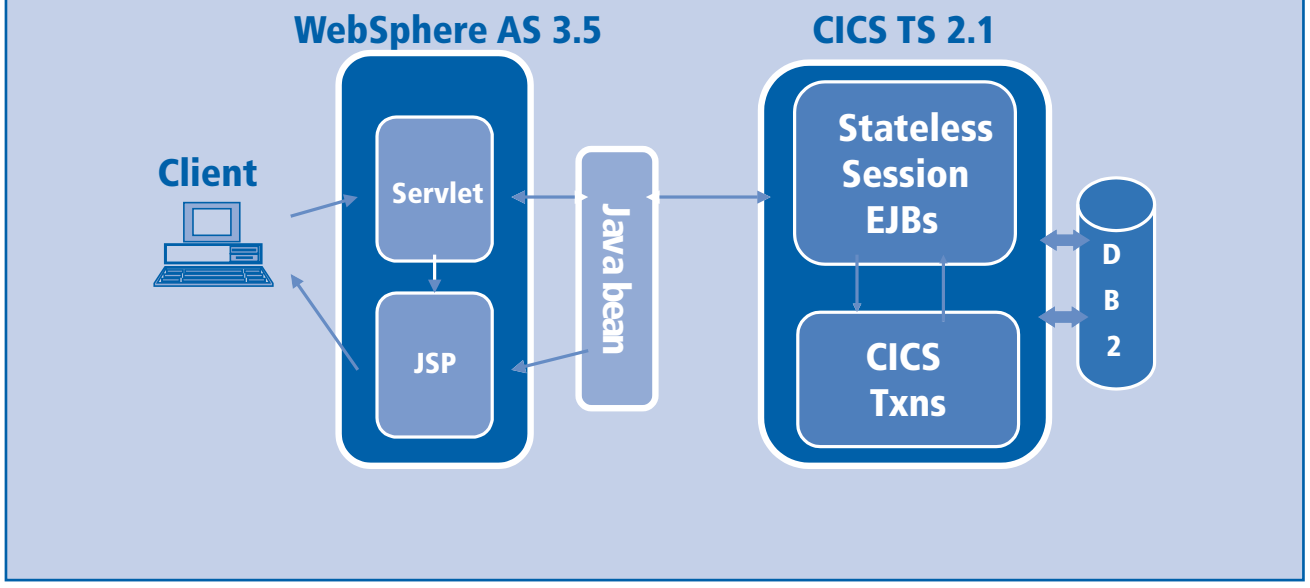
- **creates applications that can be customized for individual customers relatively easily, especially with the possibilities inherent in HTML for improved look and feel**
- **can be implemented in a scalable form; they take advantage of vertical and horizontal opportunities by deploying on multiple server platforms which, in turn, can combine to support major workloads**
- **still allow previous ('green screen') users to continue their usage.**

The fact that we have been able to package up the interfaces to our existing CICS applications means that we have been able to re-use and extend those applications as well as incorporate them into new processes. This has saved us huge amounts.

But what has impressed us — and our customers — even more is that the Java/EJB/CICS combination means we are able to develop and ship solutions which match individual customer requirements on a much more rapid basis than previously. Whereas, historically, customers had to wait until we shipped a new major release — which was always a major exercise for us to deliver and for our customers to implement — now we can focus on delivering what is wanted. Furthermore, it is much easier to re-use this in subsequent products or services.

The result is that we are now much more responsive to market requirements. We can shorten the time between customer requirements reaching us

Figure 1.6: EJB and CICS



and being able to develop and deliver them (a recent and significant one took only four months, from scratch).

Finally, I need to return to the subject of application servers. Although we are a partner of IBM's, we do not tell our customers which application server (or directory server) they must use. We do tell customers that we use WebSphere as our own development environment. But, if they wish to use another run time — like WebLogic or iPlanet or whatever — this is fine with us.

Because we stand by the Java environment as a whole, rather than attempting to be clever round the edges, we avoid dependencies which might constrain customers; we intentionally avoid binding ourselves to any environment. This is both attractive and seductive, which explains why the appeal of Java to the enterprise is so great. By working to openly accepted standards we are giving customers more in terms of choice. They like it.

Similarly, we do not have to go in saying that 'we think you should run this as a three or two tier physical implementation'. Instead we can show the customer how what he or she wants can work. We can point out the pros and cons of each of the options while still leaving the choice to the customer. That is the beauty of Java on all environments, including CICS.

Management conclusion

The appeal of Enterprise Java Beans, and the whole Java environment, is often assumed to apply only to non-mainframe environments. As Mr. Willson describes, that is an incomplete picture. Java and EJBs have a real role to play in the mainframe world, especially for CICS users:

- ***extending the life of existing, proven, applications***
- ***enabling new function to be added without losing what already works***
- ***adding to the range of deployment options available.***

Besides these technology-related benefits, Mr. Willson also illustrates how EJBs and Java are improving the business functions which can be offered to customers, as well as speeding up their development and delivery times. All in all, Java and EJBs do not have to cause the demise of legacy applications; rather they are able to enhance their role.

What needs to change to support enterprise computing in a mobile world?

Duncan Johnston-Watt
Managing Director, Fixed Income Technology
Instinet

Management introduction

Duncan Johnston-Watt is Managing Director — effectively Chief Technology Officer — for Fixed Income Technology at Instinet, a subsidiary of Reuters plc. Before taking up this role he was head of Development and Systems Architecture for Instinet's Fixed Income products and, with his team, built the latest Instinet Fixed Income Trading platform around J2EE.

In this case study he talks about:

- *mprism.net (a collaboration between Sun, Compaq, Reuters, Isocra and SpiritSoft)*
- *mprism's first technology demonstrator (iPrism)*
- *what has been learned about making applications and middleware available in a mobile environment.*

Instinet and its Fixed Income Platform

Instinet's raison d'être is the provision of an anonymous, transparent and neutral service for fixed income instruments — for example government bonds, US treasuries and asset classes based around the Euro. Where Instinet differs from a broker or securities house is that it does not trade on its own account nor bet against the people using its system. We do not take positions.

Instead what we provide is transparency of pricing to the market. Using Instinet you have full depth: you see the orders that are in the stack both at the same price as the very first or top order and what is coming behind.

Another much prized feature is that of anonymity. Instinet provides a platform in the middle so that when a trader at Bank A trades with Instinet which is simultaneously trading with Banks B, C and D, none of these Banks know who they are trading with, just that an offer has been accepted and has become a trade. Even when settlement occurs, anonymity is preserved.

This is important because most financial institutions do not like everyone to know what they are doing. This, however, is only workable if the intermediary is trusted.

In addition to the intermediary organization being trusted, it is crucial that there is complete confidence in the platform. In 1998 I headed the team which developed the new Instinet Fixed Income trading platform. We decided to adopt what subsequently became known as J2EE (we were a very early adopter as this was not formally announced until JavaOne in 1999, almost a full year after we elected to run with Java enterprise APIs, EJB and JMS) and we have proven its value by delivering a large scale, business to business system built using it.

That system is now Instinet's Fixed Income platform. It provides an inter-dealer broker service that hooks up all of the major investment banks around the world. It has gone live in Europe and in the U.S. and now Instinet is looking to partner with others in the Asia Pacific region, particularly in Japan. Already billions of dollars worth of fixed instruments are being traded on it every day.

mprism.net

With the new Fixed Income trading platform deployed, my focus changed — and for the past few months I have been driving 'mprism.net' (*mprism*). This is a Reuters-sponsored technology initiative currently involving:

- **Instinet**
- **Reuters iFinance**
- **Compaq**
- **Isocra**
- **SpiritSoft**
- **Sun Microsystems.**

The collaboration draws upon the collective expertise of the various participants covering:

- **trading applications**
- **wireless technology**
- **mobile devices**
- **enterprise middleware**
- **communications and services management.**

mprism seeks to use Java technologies end to end across a broad spectrum of devices and networks (particularly including wireless) to enable enterprises to deliver intelligent services on a range of devices, including:

- **desktops**
- **handhelds**
- **Internet appliances**
- **smart phones**
- **ultimately anything that conforms to the new J2ME/CDC specification.**

As part of its objective, it is seeking to understand what are the challenges behind the quality of service issues which need to be applied to the mobile space when this is linked to mission critical systems (like trading systems). In this context, *mprism* plans to explore what platform-independent infrastructure and tools are needed to enable enterprises to deliver services and applications to a mobile world. This includes looking at:

- **what we mean by an intelligent server infrastructure**
- **what an application needs to know about quality of service**
- **how applications can adapt and deliver what is needed in a volatile (and often hostile) environment like that found in mobile communications.**

I should add that we do not stop here. We have recognized that the emergence of voice-capable, mobile, communicator-class devices brings a new potential to deliver change. If voice-capabilities evolve as rapidly as we think they will, we envision a cultural revolution that will fundamentally change the way we interact with the world around us. *mprism* wishes to take the lead in the understanding of the impact of wireless — and wireless when combined with voice-capabilities — in order to provide the necessary structures and infrastructure.

iPRISM derivation

The iPRISM demonstrator grew out of an interest shared with Steve Ross-Talbot, the Chairman and CTO of SpiritSoft (see also, **FINANCIAL MIDDLEWARESPECTRA**, February 2001, page 12) as to what can usefully be delivered in a mobile wireless world. We both came from a point of view which had been shaped by building powerful enterprise systems. In my case this was most recently the Fixed Income platform which is, in reality, a highly sophisticated B2B exchange. In his case it was work in the financial sector and the issues raised when SpiritSoft was building its Java Messaging Service engine and Event, Condition, Action engine (SpiritWAVE and SpiritINTELLECT respectively). We both were convinced that the traditional enterprise model did not necessarily provide the sorts of services that a mobile device would require.

In particular we were both struck by the absurdity that people in Japan people were prepared to pay 100Yen (c. US$1) a month just to wake up each day with a new background on their mobile phone. While that is one kind of service, albeit a fairly degenerate one, the kind of services we were starting to think about were those applicable to enterprises which today are delivered largely over proprietary and relatively static networks which tomorrow might go mobile.

We felt that:

- **change was in the air**
- **the mobile 'solutions' available today were merely uninspired derivatives of what was already familiar**
- **most of these initial efforts fell short of what was truly needed.**

One of the key issues we identified was the volatility that inevitably comes with a mobile and wireless world. Another was how — where mobility was reasonably well serviced, as in Manhattan or Tokyo — people started to:

- **exploit mobility**
- **expect wholly new services delivered in new ways (the daily change of telephone background in Japan is just one example).**

In effect we tried to imagine what would be the consequences of using something like the Instinet Fixed Income platform in a mobile environment. While wireless enables people to be mobile, did that mean that all existing transactions and information services should be made available out on those mobile devices?

From our perspective, we wanted to understand what (if any) the value was for providing a mobile service:

- **not as a replacement for the full desktop environment that traders typically have at their finger tips**
- **but to offer what we now term 'tear off computing', the ability to 'tear off' an amount of technology, put it onto a mobile device and take it with you as you move out of the traditional work place.**

If you think this is far fetched, consider a trader (or salesperson) going to a meeting, on or off site — or even going out to lunch with that client. As soon as he or she has left the trading desk, he or she is information poor (which may mean powerless). Our challenge was to work out which functions were needed, how these might be delivered and whether — if reception/transmission quality rose or fell — the availability of any services should be capable of change.

iPRISM

iPRISM is the first *mprism* technology demonstrator. It uses a combination of technologies:

- **handhelds (Compaq iPAQ 3600s running Linux with a Java runtime environment for executing our demonstration applications)**

- **servers (Solaris, NT/2000, or any other system running Java)**
- **wireless**
- **messaging (SpiritLITE, for lightweight Java messaging).**

iPRISM demonstrates that complex transactional services — such as Instinet's Fixed Income service and to a lesser extent Reuters financial and news information services — can be delivered by wireless communications to devices as small as Compaq's iPAQ handhelds. It deconstructs classic financial services and shows how such services need to be rethought if they are to work in a complex environment where 'the enterprise hits the road'. It explores quality of service issues, from when reception/transmission is good through to barely acceptable and what has to change in the way that interactions and services — including at the application and user levels — need to be accommodated.

To investigate and demonstrate, iPRISM uses two Reuters Group services:

- **our Fixed Income service/trading platform**
- **Reuters iFinance, which delivers news and quotes across the Internet and now, for the first time, via a wireless network.**

The Reuters iFinance team had built a thin client tool kit which enabled it to deliver applets, all written to use Personal Java. What they had not done was use a Java Messaging Service (JMS) compliant transport for delivery of information — which is what SpiritSoft's SpiritWAVE and SpiritLITE deliver.

With the development of iPRISM, Reuters iFinance has introduced a lightweight JMS transport to deliver information to an application it designed specifically for the iPAQ form factor. On the iPAQ there is a client which enables a user to use drop downs to provide a stock watch or query a specific stock price or receive breaking news. The lightweight JMS provides the message middleware transport infrastructure.

Reuters iFinance's implementation for iPRISM, however, went further. Rather than just deliver an individual quotes it takes the request for this information as a cue pre-emptively to serve up all the known background information relevant to that quote in case you should want it. What happens is that the back-end, on receiving the individual quote request, searches all the news stories relevant to that quote or company. So, when you receive the quote, you can also drop directly into the news section where you will have all the headlines and most recent stories relating to that company.

In essence, the back-end provisions (or serves up) the information it thinks you may need. In the future we can envisage that, heuristically, it can learn what you, 'Mr Jones', are likely to want. Say you ask for CISCO, it will provide the quotation and all the recent stories about CISCO. However, it may anticipate that you will also drill down further: for example, based on your previous behaviour, it might expect you to ask to see the prices of the last 100 trades in CISCO of over 1000 shares — and pre-emptively obtain this for you.

We recently showed iPRISM at *Java on Wall Street* and also at the grand opening of Sun's Stockholm Center of Excellence. We had quotes and news being sent from the iFinance server engine in Palo Alto out over the Internet and then over a wireless LAN to an iPAQ.

At the same time — on the same iPAQ — we showed the Instinet Fixed Income transaction service delivering both information and transaction processing to a Java application connected to our Instinet Fixed Income trading platform servers — again using SpiritLITE to provide the light weight messaging middleware. What we added was quality of service considerations and overlaid this on the existing Java messaging service.

On the back end we used Isocra's LiveReport engine to react to the same quality of service events and adapt its delivery strategy accordingly. In effect we used LiveReport as the interface to our complex internal object model to enable us to take a cross section through the available information and serve this up as an active report. In doing this we avoided presenting the user with the need to interact directly with the underlying complexity of our internal object model (which clearly made no sense). If, subsequently, you chose to drill down further, another LiveReport was requested, generated and published.

To complete the technology thought here, these LiveReports were defined in XML. They have

both a static component and a dynamic component. The latter is what really tells the application server where to look for changes and where to expect changes to occur. Only when changes occurred were these communicated to the iPAQs. This minimizes network traffic. By generating new reports in the background on the back end server, the quality of service rules applying to the application meant that reports were being tailored to the available bandwidth (for want of a better term).

On the transaction side we made sure the same information about quality of service was available to the client application running on the iPAQ. When someone tries to trade, the application looks at the signal strength as an indicator of the likely effective quality of service available. If a user tried to initiate a transaction but the quality of service was insufficient, the application would warn the user that trading might have additional risks — principally that poor quality of timely information might disadvantage the trade. It can even refuse to accept a transaction if the service level has fallen too low to be acceptable for the business.

Quality of service issues

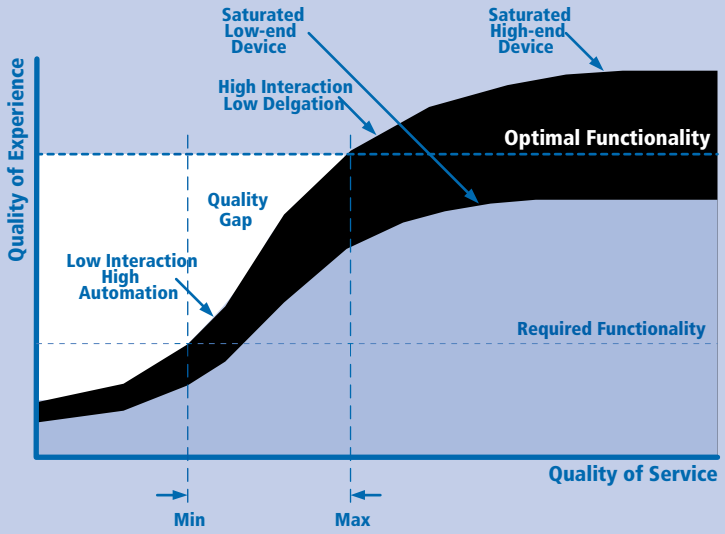
As Steve and I suspected in our original discussions, multiple issues arise because of the wide variation in terms of the quality of service that can be delivered through a mobile network. These can be measured in terms of latency and bandwidth available at any given time. There are also less obvious associated issues, including how do you present quality of services issues to a user in a way that is meaningful and usable to that user.

The one that stands out — as you walk around — is how service quality rises and falls (Figure 2.1, and directly analogous to the reception bars on your cell phone). Just crossing a street may mean a big jump up, or down, in reception/transmission quality. Going from one office to another may have exactly the same effect. The only predictable aspect is that you cannot predict what reception will be like over the lifetime of a session.

If you are seriously contemplating delivering a mobile service to an enterprise or business — something which has complexity either in terms of content and/or in terms of the amount of interaction (for example trading) — you have to address the quality of service issue up front. Certainly that is the proposition at which *mprism* is looking.

Let me offer an example to illustrate the concerns that we discovered. Assume you are a Fixed Income trader going to a meeting with a client in their office. You know that they wish to trade something, but not what. You arrive at their office, the meeting starts and you find out that they want to sell (say) $10M of the 30 Year Long Bond. You

Figure 2.1: Quality of service/experience trade-off chart



take out your iPAQ handheld and look up the current bid and ask prices.

As it happens, this client is in Manhattan in a corner office with a great view through the window. Reception is good. The prices for all the US Treasury on the runs — the most active US 'govvies' — are flowing through, over the wireless network. You can tell the client not only what is happening in the market but, if they agree, undertake the trade from the iPAQ. The client is convinced that your bid is competitive and moments later the trade is completed.

Now consider this scenario with one small change. The client takes you to a conference room which is inside the building: there are no windows (or views). Reception is masked but it is still above 60% — which enables updates every second or so. This is still sufficient to be able to receive current prices for the Long Bond (but not all US Treasuries) and work with the client to make the trade. The quality of service has dropped off; in response, the system has automatically chosen to deliver less information. One strategy would be to pick a different frequency for updating all information to the device. However, in this scenario, the system is configured to reduce the volume of information sent by focusing on just the instrument you wish to trade in order to make best use of the quality of connection available.

In variant three, the client's offices are on the fringe of the reception area, say giving 40% (or less) quality. You are still receiving updates but these are at a rate of less than one every two seconds. The danger is you may be behind the market.

In effect, the information being received has indicative value only. The quality of the data has dropped off to a point where you really should not be trading using information that may be out of date in a market where every second counts. While you have a broad feel for what is happening in the market, you may not know sufficient to undertake the deal that you agree with your client. Indeed, it may be worse: in a volatile market you may not be able to deliver the price that you agreed with the customer — thus exposing your firm to consequential losses. From the firm's perspective, do you want your trader to consummate deals that create this level of financial exposure?

In the last scenario, iPRISM provides a 'guard' on the handheld itself. If the reception/transmission falls below a pre-determined level, then the client software will refuse to accept the trade and will explain why (to the trader) on his or her iPAQ. The trader can then try to find a location with improved reception/transmission which allows the trade to occur — or can look for an alternative (such as using a phone to call the trading room and place the order by voice). Furthermore, server side logic inspects all in bound trade requests and applies a 'time-to-live' test; that is, it acts as a gate keeper so that, unless the user has explicitly overridden the standard options, trade requests that have taken more than a second or so to reach the trading system are rejected.

My point is that the commercial realities in the mobile world mandate that you should provide users with the appropriate quality of experience applicable to the environment they are in at any given moment. This needs to work:

- **in terms of the behavior of the service (as experienced by the user on the device itself)**
- **according to the quality of service that is available.**

Middleware — sitting in the middle between the handheld and the server — needs to be capable of taking decisions about the deterioration (or improvement) in the quality of service. At some points it may even need to stop users committing to an unfulfillable action (you can imagine this with a sales person and out-of-stock inventory).

What you are looking for is intelligent service provision (whether provided at the back or front end). To achieve this you need to have:

- **a strategy for delivering information based on the quality of the service (so that if you know that the quality of the service is poor you adopt a different approach to that applicable when you have good quality communication)**
- **a middleware tool for delivering the intelligence which can determine what is appropriate (probably via some form of dynamic rule) and act accordingly.**

What to do

In observing this, from the iPRISM research, we

have drawn some broad conclusions. The first is that you need to be able to describe the quality of service that is applicable to your type of business.

Second you must assign 'break-points' where a change in quality of service changes what is available and how it is delivered. Figure 2.1 illustrates how this is modeled for a given application or service. Armed with this analysis, you can manage the quality gap between the service you would like to deliver in an ideal world and the service that can be delivered in the current context.

Fundamentally, you need an understanding of what qualities of service matter to you if you are to think about the kind of intelligence that needs to be included in the infrastructure itself. If *mprism* is doing anything right, it is that we believe we are creating the appropriate infrastructure with the right tools to enable an enterprise — be it a financial services organization such as Reuters, an investment bank or indeed any business which believes there are productivity gains to be had from utilizing these concepts — to take its existing services and effectively move these into the mobile world.

This, however, will mean re-architecting your applications and services. In particular it will mean layering them so that they are designed appropriately for that mobile world.

When I say 'layer' I mean deciding where — for your given application or service — the cut off points exist. In effect this means that, for each layer, there is an appropriate amount of functionality which is needed to deliver:

- **at the highest level**
- **the lowest (acceptable) level**
- **in between (which may mean several intermediate layers).**

Again let me illustrate. As the quality of service improves, then typically the bandwidth (or the environment you have) is richer. In the UMTS or G3 world, for example, you will theoretically have universal mobile telephony on a 'permanently on' basis.

What I hear people saying is that, 'Oh that means I can expect in excess of 2 megabits/second to my cell phone or PDA'. Maybe so. But when you go into the internal conference room or out onto the street, you may find the best you will see is 384 Kilobits/second. Once you climb into your car, this may deteriorate further.

Our expectation is that you will need to explore and define a relationship between:

- **the quality of the service available**
- **the richness of experience needed for the user of that particular application.**

In any event, we are keen to work with near-term 2.5G technologies where clearly there will be significant constraints for some time to come and which therefore stand to benefit from our approach.

Applications, layering and deconstruction

Once you have an understanding of where the 'break points' are, you face a second set of issues. To provide the requisite services you will — in our experience — need to deconstruct existing applications and/or services — and rebuild round the layers.

If this seems like overkill, or excessive, there are two immediate benefits. The first is that layering obliges you to think about which devices (including desktops, handhelds, smart phones, Internet appliances, etc.) are capable of doing what. You will likely find that some do not have enough onboard capacity to download many or even whole applications. This gives you a major clue about what you need to deliver in terms of:

- **the most pristine of environments**
- **the worst of environments.**

Knowing these extremes, and what is acceptable in the middle, provides the basis for heuristic processing at the back end which can determine what device will receive what applications (or components of applications) in any given mobile environment. With this knowledge you can dynamically change provisioning according to circumstance. The nice aspect about this is that you are not shifting the entire service — or the entire application; you are only shifting the amount that you need for the particular quality of service available.

A good example of this applies to analytics. There may be no point in shipping out the code across the mobile network unless your user is in an area

where the quality of service level means that it makes sense to do it. Only then do you provision the application/service.

The second advantage of layering is that it obliges you to break down monolithic applications into practicable ones. Not only does this rid you of monoliths but smaller, complementary, applications (or modules) are easier to upgrade or replace without having an impact on other applications.

To make the determination of what should be used when and where, we intend to use an Event Condition Action (ECA) framework. This can represent what is possible where and when. It is also separate from the applications or modules: its 'rules' can be changed as required. In this context:

- **an event might be the desire on the part of a user to initiate something**
- **the condition will be to check the available quality of service and establish whether it is acceptable to deliver this (or not)**
- **the action will fire the thread of code which matches the circumstances.**

By breaking applications down into smaller units, you can expose and invoke the logic. You have made explicit what is implicit in most applications. And, if over time, you discover that people can tolerate a different quality of experience than you first thought, you can change the ECA model and/or the applications — as best fits the need. This is a major benefit: it makes delivery of flexibility both easier and faster.

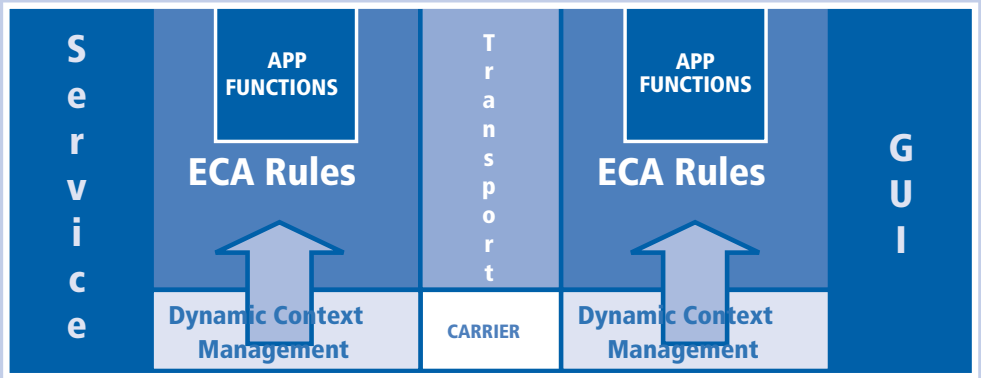
Similarly, on the server side, the same notion of Event, Condition and Action can drive or moderate the behavior of your services. This is exactly how you would change the way you provision content — and minimize (even avoid) constant periodic refreshes.

Where next?

The initial iPRISM technology demonstrator was exactly that — a classic 'show and tell'. What we focused on was how we could — on the server side — intelligently predict what a user might need and effectively have that information ready when it was asked for (rather than waiting for a user to request it). We also included some important quality of service considerations.

Now we are intent on taking the demonstrator to the next level — with an increased focus on delivering intelligent quality of service provision. Where, in the original demonstrator, quality of service was — to all intents and purposes — hard

Figure 2.2: Architecture



coded, we are looking to introduce the SpiritINTELLECT Event, Condition Action capability (Figure 2.2) on both:

- **the front end devices (on an iPAQ or similar class of device)**
- **the back end servers.**

Assuming we are successful, and I see no reason why we should not be, utilizing an ECA engine will enable us to formalize behavior in a structured manner. For example, the Instinet Fixed Income trading application — with which I am most familiar — has already been broken down into components. We can plug these into the ECA engine so that it can govern the behavior of the service as encapsulated or represented by the events, conditions and actions appropriate to the available quality of service.

Once this is working, ECA gives us the ability to modify rules over time and separate from the individual application components or modules. This, in turn, will allow us to fine tune the behavior as necessary.

In addition, on the server side, I think the importance of ECA will be most significant when you are trying to deliver a subtle interplay between services. For example, if you are managing your own calendar you are usually working with a service and that will alert you to what to do and when. We think it is possible to offer a more useful, and powerful paradigm, while still giving people the ability to manage their own calendars.

In our analysis, a stream of calendar reminders might be communicated to a back end server which 'thinks' about what those calendar events mean to you. Rather than just having an electronic tap on your shoulder, we are exploring how the calendar event can be used to produce enhanced information.

For example, let us say you have a meeting with client A. The back end would prepare a client update which will be ready (say) the hour before you go into that meeting. It would bring together the latest in relevant events — perhaps breaking news, a change of leadership, the current inventory position or whatever was relevant to that meeting with that particular person in that particular business.

In effect we would fine tune information to reflect the context so that you have what is needed when you need it. As information changes, the ECA engine would ensure that the various back end services would produce what is needed — and communicate it out to you.

You could say that we are in the business of creating smart services that are precisely targeted to meet your particular requirements at any given point in time; that we are using smart delivery strategies and smart devices to help us achieve this; and that one of the key differentiators is our use of Java technologies end to end.

Lessons learned

My first (lesson learned) is that collaborations — of the form that *mprism* represents — are going to be the way forward. Each of the companies in *mprism* has different strengths and understanding. It is through the combination of these strengths that real progress will be made — as the first iPRISM demonstrator has shown. Everyone wants to take ideas and move these quickly out of the lab and into the real world. But to do this you do have to select your partners with great care and then work at understanding each organization's aspirations and motivations.

My second lesson learned is that, like all good paradigm shifts, the basic insights are not necessarily unique to us. There are others who have expressed equivalent ideas and objectives — some of them are now our partners. The differentiator is in the speed of execution and in gaining of market acceptance.

Finally, we have proved that once again there is real value in being early adopters of Java technologies. Back in 1998 it was J2EE but now the focus is firmly on J2ME and also in the mobile space where these two very different worlds collide. Java has become a key to the future.

Acceptance of this fact was not prevalent as recently as last summer. Today it is obvious. Not to use Java technologies in their various manifestations (especially J2EE, J2ME, and, as a bridge between these two worlds, JMS variants) is fast becoming a constraint — especially if you want to go beyond the walls of the typical business.

Why is this? It is because there has been a tremendous amount of work done to create an intelligent

infrastructure for Java which can deliver to people what they need. Sun has proved that. We have proved that. SpiritSoft has proved that — and so are, and will, many others.

Management conclusion

Mobile computing is coming. The question is no longer if, but when. As Mr. Johnston-Watt describes — via the Reuters iFinance and Instinet demonstrator examples in mprism *— it is achievable today, albeit with some effort.*

In the middleware context, however, what Instinet has achieved in an opening of the next generation of Pandora's box. Using tools like SpiritSoft's Event, Condition, Action engine (SpiritINTELLECT) running over a JMS messaging implementation, Mr. Johnston-Watt shows how computing can become portable and usable, even in the demanding environment of the finance sector. This is important. If the finance sector can do it reliably, other sectors will follow. And the implication is clear. Middleware follows into the mobile space, albeit with different operating characteristics.

Database — middleware's Open Source Achilles heel?

Amy Wohl, Principal, Wohl Associates
and
Charles Brett, President, C3B Consulting Ltd.

Management introduction

In May 2001, Gartner Dataquest valued the database market at $8.8B for 2000 (Figure 3.1). For most people what impressed — or alienated — was that Oracle, IBM and Microsoft shared 78.8% or nearly $7B.

There is, however, another way to look at these figures. They are an opportunity. In particular they are an opportunity for those who see Open Source code as the way forward. Even more attractive is the simple fact that there are tens of thousands of current customers who are paying that $8.8B who, in a follow up mode to Linux, could decide that the financial implications of Open Source Databases (OSDBMS) possess distinct attractions.

In this analysis, Amy Wohl and Charles Brett consider what such a scenario might mean for both customers and vendors. In addition they consider the future downstream implications for middleware in general.

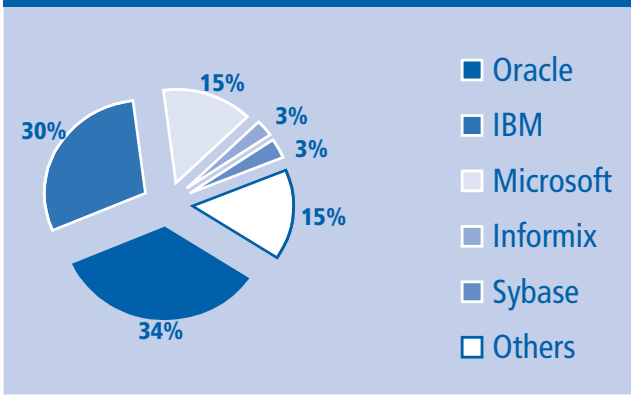
Setting the Open Source scene

Historically, new functionality starts out being built into those applications that require such functions. These functions may, over time, migrate into separate applications, which make the functions available to many other different types of application — thus obviating the need to develop and incorporate this function in each new application. This has become known as middleware.

But the story does not stop here, as both IBM and Microsoft have proved (and Sun may yet do with much of iPlanet). Many middleware functions eventually are themselves incorporated into operating systems (as, for example, Microsoft chose to do with message queuing in Windows NT). This, at least, has been the pattern of the past.

If, however, many users and organizations standardize on Open Source — for the true issue is about Open Source, not Linux as an operating system — this process might change, even though there is no obvious advantage to the creator of new functionality in putting such new function into Open Source (except fame and glory). Yet, as Linus Torvalds himself points out, the desire to create artful work may, in itself, be a sufficient reward.

Figure 3.1: RDBMS market (Source: Dataquest)



Under such circumstances, commercial developers would undoubtedly prefer to keep functionality in separate products, whether applications or middleware, because these developers would then be able to continue to collect licence fees for their intellectual property, much as they currently do. Yet such continuity may not be practical.

The software world may be on the verge of major change. The existing middleware business model may be a house of cards.

Continuing with the traditional software model assumes that:

- **either no clever Open Source developers choose to replicate middleware functions in future versions of the Linux operating system (in much the same way as Microsoft has routinely incorporated new function into Windows)**
- **or the possibility of reducing even a part of the $12B spent each year on middleware will not attract the attentions of financially-minded organizations — who may even 'sponsor' Open Source development (by way of developer time foregone or similar).**

Indeed, the existing business model for middleware may yet be proven to be a house of cards about to fall. In its place would come Open Source middleware, made ready for re-assembly on an Open Source platform or incorporated into an Open Source operating system (like Linux). Function that is expensive today would be offered inexpensively (or free), separately and with the potential to be exploited by an endless array of applications.

To understand why this might, or might not, happen, it is necessary to look at what Open Source (rather than just Linux) has achieved, and how.

Advantages and disadvantages

The Open Source movement already offers both advantages and disadvantages to developers and software users. Its advantages include:

- **a world-wide community of developers, where peer recognition matters**
- **its capability to self-organize; it adapts to the goals of the group, rather than to any one constituent community**
- **software which evolves, based on input from many sources**
- **the concept that the best work is rewarded with world-wide recognition**
- **bug fixing; bugs are found — and fixed — more quickly than via traditional approaches**

- **the ability to find (and access) 'deep' expertise outside your organization.**

Amongst its more obvious disadvantages are:

- **no single point of control**
- **potential fragmentation; no single standard is mandatory**
- **lack of direction (which may lead to lack of progress)**
- **resources which cannot be scheduled or held to account**
- **a model with no direct economic return for effort**
- **a risk that developers may withhold their best work for sale to the commercial (profit-making) sector.**

These disadvantages are emphasized by those who have most interest in intellectual property rights. Anyone who writes (whether words or code) would like to be paid for the intellectual property they produce, as Microsoft made clear when it opened a Pandora's Box by standing up in public (at New York University — in the person of Vice President Craig Mundie) and declaring that it is not possible to offer Open Source software without it having to be offered free under the terms of the GPL (General Public License).

Mundie (like Jim Alchin, in an interview in mid-February) was expressing the concern that the Open Source movement could suppress software innovation by destroying rights to intellectual property. What apparently concerned Mundie and Allchin were the terms of the GPL as the main licence used to assure free access to Open Source software, and the associated terms and conditions applicable to its use and distribution.

The GPL makes it clear that such software is freely available to anyone. This includes its source code. Furthermore, changes to GPL licensed software must also be made freely available, although such changes are not in any way warranted. But the GPL does ***not*** say that applications, which run on top of Open Source software but are separate from them, must also be made Open Source or that they cannot be sold for a fee (or that their source code cannot be distributed on different terms, for example, to an ISV's partners, say only for review but not revision).

Microsoft's stance was not welcomed. The Open Source community has begun to reply. Generally this has been in a negative way. For example, a number of Open Source gurus note that the GPL is only one of several Open Source licensing possibilities. Most others are more flexible, at least in terms of protecting a software developer's Intellectual Property.

If this is proved right, it means that mechanisms do exist for successful protection of intellectual property rights in an Open Source world. If so, the issue now becomes, are these sufficient to enable the multi-billion dollar financial savings (that 'Open Source solutions' might generate) without materially having existing software revenue streams? Perhaps, however, a better question to ask is who is being threatened, how, and by whom?

It is Microsoft, Sun and IBM who are stimulating the changes

Software companies have a long history of aggressively defending their intellectual property rights — from piracy by users, from unlawful incorporation into others' products, from changes to the source code which break the owners' terms and conditions of usage. Microsoft may be the perceived sinner 'de jour' but it is certainly not alone in this regard.

For example, Sun has 'enjoyed' an ongoing discussion with the industry about how Java should simultaneously:

- **be an 'open standard'**
- **yet have Sun controlling the content of its source code.**

Sun attempts to deliver this through the Sun Community Source License. In effect, Sun offers access to the Java source code while seeking to retain control. Sun claims this is necessary in order to:

- **prevent forking (fragmentation) of the code base**
- **ensure changes occur at the fast pace necessary for the industry to make progress.**

[You can view an extensive discussion of Sun's idea of a license which is at once somewhat open —

akin to being somewhat pregnant — and somewhat closed at: www.sun.com/981208/scsl/principles.html.]

To Sun's chagrin, demanding such levels of control has produced the opposite effect:

- **HP is rolling its own version of Java**
- **Microsoft initially sought to create its own Microsoft-specific flavor of Java and has now moved (because of Sun's litigation) on to produce a Java replacement — C# — complete with its JUMP strategy (Java Upgrade Migration Program).**

This has put Sun in a difficult position to argue against Microsoft's anti-Open Source point of view. On the other hand, Sun has not been 'all bad'. It has placed StarOffice into the Open Source domain and expects, over time, to hand control to the Open Source community of interest that forms around it. This is managed for Sun by Collabnet (at www.collabnet.org). Similarly, after it purchased the NetBeans Java Development Environment from some Czech developers, Sun made this openly available.

At the same time, Microsoft, Sun and others seem intent on discounting (or ignoring) the effect that IBM has been having on the Linux market. Even IBM itself may be underestimating its own impact.

IBM now offers Linux support (but, deliberately, does not have its own Linux distribution) for each of its four hardware platforms (from the Intel-based x-series, as well as the i-series, p-series and z-series — AS/400, RS/6000 and mainframes to the rest of us). For each of these versions there is a version of most of its middleware (infrastructure) software, from WebSphere through DB2, MQSeries and a host of other software products. This middleware is currently sold at a tidy profit (with its source code being protected).

Apache matters more than Linux

While IBM's promotion of Linux is clear, its support of Apache is, arguably, more significant as a pointer to the future. Apache is an application. It is an Open Source Web Server which has garnered considerable support, including that of IBM.

But Apache and IIS (Microsoft's Web Server) are two sides of the same coin with the same implications. In neither case do you pay directly for a Web Server application:

- **for Apache, you download it, per an Open Source licensing model**
- **for IIS, you obtain it as part of the Windows NT/2000 operating system (at 'no extra cost').**

The point is that IBM (and Sun and Microsoft, in their different ways) is already reflecting changes which have been 'imposed' by Open Source considerations — specifically in the way that some forms of software are purchased. In so doing they are:

- **altering the ways in which business customers value operating systems and middleware (this will also probably apply, before long, to how consumers value software)**
- **removing barriers for additional Open Source activity.**

The cumulative effect is that there will be more and more Open Source activity in areas where software vendors previously had thought their revenue streams were safe. Furthermore, the risk exists that others will be threatened — in ways that were either unforeseen or ignored. As will be explored later in this analysis, the database market makes an excellent starting point.

The motivations for Open Source

When asked, most CIOs are reluctant to admit that Linux and Open Source are a factor. Yet IBM tells of occasions when senior executives in user organizations have stated that 'there is no Linux in use in our organization' only for those same CIOs to discover that multiple — measured in tens or hundreds — of initiatives are already undergoing 'investigation' in their own organizations (albeit unknown to them).

The reason for such activity is simple. While many may argue about the intellectual merits or relevance of Linux, it is economics that ultimately matter.

Linux is essentially free, compared to traditional operating systems (Windows NT/2000, Solaris, HP-UX, MVS, etc.) — except for the time that has

to be invested to make it work. Unfortunately for most middleware vendors, it is their largest customers who have the most to gain from Linux and Open Source. Linux becomes more attractive the more copies are introduced: in effect the 'people/management overhead' of Linux is shared the more it is deployed.

Against adoption of Open Source products is the argument that no self-respecting executive would place his or her organization's operations on an operating system where there is no control of the development process — and not even a commercial relationship with the source. Indeed the same argument is applied to the notion of an Open Source RDBMS: could you possibly rely on it?

But such counter Open Source arguments do not stand up. Too often their proponents selectively forget the many implementations of mission critical applications that already exist using Open Source solutions. Just count the number of Linux and Apache Web Server combinations that are in production today.

Furthermore, many miss a key point here. This is that Open Source gives you control of the source code. You know what is in it. You can add to it as you need. You, as an IT shop, can have more control with Open Source than you obtain by using traditional proprietary software.

This has a powerful appeal to large customers with big IT shops which have the competence to write and maintain their own system software. Relying on IBM or Sun or Microsoft to make changes or fix problems in a timely fashion is not one's first choice. Being able to do this for yourself — as is possible to some degree with MVS, VM and VSE — has always been one of the attractions of the mainframe. Being able to do this for more than operating systems and Web servers would provide greater security as well as a financial saving.

Indeed, as commented earlier, two of the underlying attractions of Open Source are 'the ability to obtain access to 'deep' expertise outside your organization' and 'software which evolves based on input from many sources'. Suitably used by the larger organization, and compared to proprietary software products, Open Source software:

- **reduces costs**
- **delivers greater dependability**
- **provides more self-control.**

Now apply this to databases

As Dataquest demonstrated, the database market is worth some $8.8B. That means organizations — from large to small — are paying a large amount per year. Databases are a healthy, mature market.

But there is another way of looking at those billions. This is the CFO's way. Database expenditure represents a significant opportunity to save money. In large organizations, in a period of 'economic downturn', expenditure on databases can amount to tens of millions of dollars. If there is a less expensive alternative, then it will be investigated — if not promoted. Prima facie, the issue is not whether such Open Source databases (OSDBMSs) will attract but whether they will be built.

The answer is that they already exist. To make matters worse for the future revenues of the big three DBMS vendors (Oracle, IBM and Microsoft), three OSDBMSs are already available:

- **PostgreSQL**
- **mySQL**
- **Interbase.**

Today none of these is yet up to the performance, capability or reliability of an Oracle 9i or a DB2 7.2 or a SQL Server 7.0. On the other hand, none are stuffed with all the extra features which Oracle, IBM and Microsoft have added to try to justify the purchase of upgrades (irrespective of whether customers will actually use these features). Indeed, some will argue that a 'slimline OSDBMS' is preferable to a 'functionally bloated' commercial database, not least because the risk of errors is reduced.

Furthermore, ODBMSs are largely free. They already have a dedicated Open Source community and they are making functional progress. PostgreSQL 7.1, for example, has removed the irritating 8K row limit while substantially improving performance and usability. And it is available on Linux, on HP-UX, on Solaris and on BSD UNIX. OSDBMSs are not necessarily restricted to Linux (just as Apache is not).

Voting without using your checkbook

As if this were not sufficient, users (a few at present) have started to take advantage. NASA replaced an Oracle database with mySQL for its NASA Internet Acquisition Service. When faced with a software bill it could no longer afford, it

converted to mySQL. This provided the function without the cost.

At the other end of the organization scale, the Wireless Developer Network and GeoCommunities began with a Microsoft (IIS) and Oracle (DBMS) combination. This was not working out as anticipated and the decision was made to go to Open Source (Apache and PostgreSQL on Red Hat Linux). This is now working across 12+ servers and driving mission critical applications as diverse as book sales, message boards, mailing lists and software sharing. As with NASA the attraction was cost reduction. In this case the ability to avoid expenses in small start-up companies was a magnet, although running and associated expenses have also proved to be lower.

While neither of these examples are proof that OSDBMSs will succeed, they are early indicators. They may yet be precursors of major change.

Of almost equal importance are the numbers of people associated and the tools that become available. In the case of PostgreSQL, there is already a flourishing set of technical guides and a list of conversion advice for converting from:

- **DBF**
- **FileMaker Pro**
- **Access**
- **mySQL (other Open Source implementations are not immune)**
- **Oracle.**

Finally, in June, 2001, Red Hat blessed the concept of an OSDBMS. It is going to create a Red Hat distribution of PostgreSQL. The cost is staggering: a mere $199/month or $2295 for a one time charge.

The impact on middleware

If Apache was the first appreciated Open Source product, Linux was the second (although, arguably, TCP/IP was the original for what has eventually become the Open Source model). Of these Linux mattered more — because it hurt the vendors more (they did not have serious Web Server revenues to lose when Apache appeared).

The next logical target, in financial terms, is the database. Why? The savings are too great to ignore.

If OSDBMSs are accepted — say in 2-3 years — they will also represent a thickening of the Open Source wedge. A new cycle will then start.

If OSDBMSs succeed, other major software expenses will be examined, to be assessed for their suitability to 'go Open Source'. Expect to see other previously profitable middleware attract Open Source developers. Obvious examples of products standing on the threshold include:

- **application servers (WebSphere, WebLogic, iPlanet)**
- **messaging (MQSeries, JMS implementations, etc.).**

The point is that Open Source really does have the potential to be the Pandora's Box which Mundie and Allchin forecast (albeit for their GPL reasons).

Possible vendor impacts

If the OSDBMS concept does acquire momentum, the first impact will be felt by those currently enjoying the $8.8B of sales in 2000. Of these, Oracle looks likely to be hit hardest. Not only does it lack an operating system into which to fold database function (as Microsoft could do by moving SQL Server into Windows 2000+), but it also lacks the hardware on which to locate microcode beneath Linux for which it can charge (as IBM will likely do on its i-, p-, x- and z-series hardware).

On the other hand, Oracle possesses user applications — Oracle Financials. As applications are where software revenues will concentrate (as a percentage of the overall market), Oracle is not without opportunity.

IBM will not escape unhindered. While it will have the ability to charge for specialty microcode (to enable dynamic partitioning when using Linux, for example) and to include DB2 in its operating systems at no charge, the 30%+ of its profits derived from software, mostly from middleware like DB2, will shrink. Furthermore, the attrition will start in its larger customers, who will be the ones proving OSDBMS.

On the other hand, IBM will be able to reduce its costs. It can effectively contract out development

to an Open Source community, which is sponsored by those interested in financial savings, IBM's customers.

Microsoft will also be hit — although, like IBM, its savings in development costs may offset the loss of revenues if it chose to go down the operating system route (by making SQL server part of Windows 2000). As for other middleware function, Microsoft already incorporates much of this free in Windows 2000 — and believes this is where middleware should reside.

The story at other vendors differs, depending on their specialization. Compaq, for example, has little middleware of its own and has already happily embraced Open Source. It will be little affected, unless it tries to enter the software business.

In contrast, Sun is exposed all ways round. If it continues to promote Solaris and urge its customers not to buy Linux for their servers (as seems likely), every decision to use an OSDBMS will likely be a decision not to use Solaris or Sun hardware. This will place Sun on the rack if it insists on fighting Open Source in general and OSDBMSs in particular. (You can also rely on Sun's management to repeat the mantra 'that it is not possible to perform serious work in an Open Source environment'). Sun has much to lose, even if not database revenues per se.

For developers of applications which are for 'use by users', in contrast, the position will be much more attractive (at least for a while). Traditional barriers to entry, not least the cost and variety of middleware, will be savagely reduced.

On the other hand, highly successful application software vendors will now run the risk that their own applications will draw Open Source attention. Given the amount paid to the likes of SAP, Siebel and PeopleSoft, their functions may yet become natural candidates for Open Source versions. Financial success and big customers are what are likely to drive Open Source activity in the future.

Does any of this matter?

Having argued all of the above does not mean that it will happen. Given the fast moving nature of the software business, it is quite possible that Open Source will not be able to catch up.

Equally, should Linux or Apache fail in some public way — like being associated with the downfall of a large organization — then Open Source in general, and OSDBMSs specifically, will likely be tarred by association. This has to be a prime threat and it is difficult to manage: we just do not know what might happen.

Other risks to OSDBMS also exist. Fear, uncertainty and doubt — cast in the form of 'would you trust your business to a bunch of uncontrollable geeks?' — will certainly be one argument used. Another will be that 'database is too big and important for an Open Source Solution? A third is that there will be insufficient interest in refining Open Source databases. Lack of development support would be a killer.

On the customer side, there are several arguments that may yet mitigate against acceptance of OSDBMSs. The first is a lack of enthusiasm, and thereby support, for 'yet another DBMS'. Possibly more significant may be the feeling that, in large organizations, the database is really only a small part of the IT cost and therefore not worth risking.

Although it would not help OSDBMS, a refinement of this would be the rise of an expectation that database function should be an operating system function. If this took hold, the results would largely be the same as OSDBMS — a reduction in DBMS revenues, but without the advantages of Open Source.

In this instance, Microsoft and IBM would likely be the big market share winners (if you can have market share of a market that has disappeared). Yet this would probably be too much of an inverse irony for the 'custodians of the soul of Open Source' to allow to happen. That might be exactly the spur to keep an Open Source development community interested.

Management conclusion

As suggested earlier, there are plenty who will argue that no one in their right senses will move their data to something as unknown as an Open Source database. Why take the risk?

The same was said of Linux. Yet Telia (in Denmark) and others have been putting Linux (and Apache) on large IBM boxes in order to consolidate the multiple Linux instances they already possessed onto

one system image. The management, financial and efficiency advantages are too great — especially for larger organizations. Open Source starts small and spreads wide. It also starts as much at the top (in large organizations) as at the bottom.

OSDBMS has the potential to be middleware's Achilles heel. It will only be this if OSDBMS succeeds, which it might not. But a saving of $5B+ is a powerful incentive.

The reality is that Open Source in general — and OSDBMS software, in particular — will have its greatest appeal for large customers. By exploiting the financial saving that OSDBMSs can offer, these large organizations will achieve savings — major savings. In parallel, because large organizations are acting to freeze their database checkbooks, there will be additional recruits to the OSDBMS community. With more recruits, rewarded as much by the challenge as any pecuniary interest, more work will be done more quickly to give OSDBMS products the greater capabilities they currently lack.

In addition, OSDBMS will not be as bloated as many of today's RDBMS products. At the same time customers will be able to tailor or add specific function that works for them (before choosing whether to make it available to the broader OSDBMS community). Simultaneously, congruity of database engines is likely to increase — something most IT executives appreciate.

If the driver for all this is primarily economic, the impact will be to change the way we buy a great deal of software. The net result should be that, over time, most infrastructural software will be less expensive. Middleware will not go away, but selected forms will:

- ***cost less (or nothing)***
- ***be of superior quality***
- ***be built to common approaches.***

This is good for customers, good for competition and good for application developers. It might even be good for middleware developers, finally convincing them to focus their efforts and considerable resources on delivering new initiatives, refinements and architectures. The result will be a different software world where prolonged financial success will act as a magnet to draw the attention of an Open Source community.

Indeed, this may yet prove to be the real control on Microsoft, IBM, Oracle and others — rather than the Justice Department. Mundie and Allchin may yet prove to have been right, if for the wrong reasons.

How are Internet trading platform vendors deploying middleware?

Wayne Richards
Technical Specialist and Consultant

Management introduction

A revolution has occurred in the way that middleware supports Internet-based commerce. While the 'buzz' surrounding Internet trading platforms has risen (and fallen), the role that middleware (in the context of application servers) has acquired a level of importance not seen since middleware's early days as an ad hoc cobbling of disparate infrastructure services. Put another way, middleware has been legitimized by these Internet trading platform vendors, and essentially has found a new home. Trading engine software has come to rely heavily on middleware services that not only enhance the power of the trading applications but also provide integration with legacy and/or back office systems.

In this analysis, Wayne Richards (who has been involved on both sides of this revolution, developing middleware solutions for the financial services industry before working with a 'dot-com' trading company that sought to leverage the Internet, B2B exchanges, financial risk mitigation and a trading engine) provides a practical examination. It is particularly pertinent to those in the process of integrating middleware with Internet trading platforms. For those who doubt that trading engines have changed, read on about how and why the likes of Commerce One, Ariba, Moai, VerticalNet, At The Moment, NextSet, ecMarket and others have changed or adapted.

Doing business in the e-dimension

As more and more businesses, of all sizes, enter the electronic marketplace, enterprises are scrambling to connect themselves to their value chains as rapidly and effortlessly as they can. These expansion trends have resulted from rapid changes that seem intent on:

- **replacing the traditional hub and spoke business model, that drove the original growth of trading communities**
- **substituting a peer to peer model, where organizations can respond with equal vigor to the variety of trading opportunities among their value chain partners.**

The technology enabler for this transition has been the advent of the Internet coupled with the development of Internet trading platforms. While the 'single' Internet provided for connectivity options that have multiplied more rapidly than ever, a plethora of Internet trading engines and platforms has emerged — each touting the benefits of its products and/or services.

Companies such as Commerce One, Ariba, Moai, and VerticalNet were some of the first to embrace this shift into the new frontier known as Internet exchanges. These were soon followed by a second- and third-tier group of solutions providers, each concentrating on specific business verticals or trading models and including:

- **At The Moment**
- **NextSet**
- **ecMarket.**

Indeed, so complicated and competitive has the B2B Internet trading space become that these same 'Internet trading platform vendors' (or ITPVs) are finding themselves obliged to juggle what appears to be an endless array of technology solutions, relationships and standards. It has become all too clear — rather quickly — that the 'one size fits all' approach to technology that they aspired to deliver has failed yet again.

One of the immediate problems realized by this plethora of solutions was that not only did ITPVs lack a cohesive set of middleware services, but there were no standards among their ad hoc solutions for application integration. With that realization, an entire life cycle of proprietary product design, development, support, and integration gradually gave way to the adoption of common, off-the-shelf middleware products that support each trading platform, plus its back end applications.

What happened?

Those B2B companies that were initially 'pure Internet plays' easily availed themselves of the premiere versions of Internet trading technology. As startups they carried minimal or no technology baggage (proprietary systems, legacy systems and back office systems). But the older, more established (and still in existence) organizations that wanted to exploit the Internet's e-commerce highway soon discovered that the majority of the solutions offered by these new ITPVs lacked the basic capabilities which would enable their existing applications to integrate with the trading platforms being sold by those same ITPVs.

What became clear was that the leading ITPVs needed to add or offer sets of integration technologies, or middleware solutions — in the form of application broker and/or application server tools — in order that their customers could link their existing applications to their shiny new trading platforms. Without this, expanding market share was going to be nearly impossible.

Unfortunately the ITPVs found out (the hard way) that the adoption (and delivery) of solutions using middleware — or sets of solutions — was difficult. The result has been varied, with almost as many solutions as there are trading platforms and almost daily announcements about new B2B technology integration.

So far, the end result is far from clear. Customers are finding it difficult to reconcile the hype about trading platform solutions with the reality of implementation. Put another way, the sheer volume of hype versus reality continues to compound the confusion.

The implications

What has this meant to the ITPVs? Actually, their adoption (and introduction of middleware) indicates that they are beginning to 'get it' when it comes to integrating their products with the software that already exists in their customers' enterprises. It may also mean that the majority of the

ITPVs have understood that it is better to buy than to build middleware products and services — both technologically and economically.

More importantly, what does this mean to end users? It is that end user requirements are finally being addressed, including:

- **the level of off-the-shelf system integration that needs to be achieved**
- **the types of inter-system communication protocols that need to be utilized**
- **the type of information that needs to be transported from the trading engine to the rest of the enterprise**
- **the data that needs to be transformed before/after transport (and how)**
- **the security requirements that must be satisfied**
- **the mechanisms for managing these trading platforms.**

Integration

Integration affects organizations in two ways:

- **first, they need to be able to connect internal applications — including legacy systems — to their chosen trading engines**
- **second, organizations never know how far ahead (or behind) their new trading partners' systems are.**

Transparent trading engines handle protocol translation 'on the fly' in order that their customers can use selected or preferred back office systems. What customers want is to support their trading partners with different systems — without adding layers of integration software or jeopardizing performance.

Transparent communication and transparent data management

Transparent communication enables organizations to execute transactions with a large number of different electronic trading partners without the equivalent number of customized interfaces. Corporate IT managers are constantly looking for solutions that support at least the major protocols in use, most usually:

- **HTTP**
- **SMTP**
- **FTP**
- **CONNECT:Direct (NDM)**

It is no coincidence that these are also usually the only protocols that security managers will let though firewalls.

With nearly a dozen major standards for document structures and packaging — X12, EDIFACT, XML and others — format transparency has become critical. The need to exchange data or documents with a large community (and in ways which are understandable at each end) is now unavoidable.

Transparent security

Security technologies are currently one of the most diverse and rapidly evolving components of e-business processes. As a result, organizations are consciously avoiding trading engines that have committed to a single security standard.

Trading engines that support multiple security options — or allow for the inclusion of independent security standards to be fitted, retrofitted or upgraded, as necessary — are proving to be not only more cost-effective in the long run, but more secure as well.

Transparent system architecture

To maximize flexibility and respond to new trading partners, organizations need trading engines that are independent of specific:

- **operating systems**
- **databases**
- **translators.**

In most industries legacy systems with older or proprietary components need to co-exist with the advanced XML solutions established for e-business. In consequence, user organizations are facing the need:

- **either to support both**
- **or to accept technological limits to their operational or business efficiency.**

In practice, the first wins. The second is rarely acceptable.

Meanwhile, over in application broker land ...

While the ITPVs were busy at work developing their products, another group of software companies noticed the shortcomings that the ITPVs had been creating in the areas of:

- **integration**
- **communications**
- **data management**
- **security**
- **transparency.**

These 'other' software companies originally had devoted their efforts to building specific middleware solutions that were primarily directed to enterprise customers faced with making their internal applications interoperate with each other. These middleware vendors hawked their products as application (or message) brokers — and stuck to the middleware basics:

- **message communication**
- **data transformation**
- **rules-based routing**
- **elementary security**
- **and the like.**

Their focus was to sell their solutions to the large enterprise customers with little or no regard to adherence to standards and/or extensive reusability. Their reasoning was simple — this was where the money was.

In this market, generalized, standards-based products were not as important as being able to tie together an organization's existing applications and make them work with each other. Besides, there were no mutually accepted standards that these application broker companies could (or would) support.

Thus, a number of proprietary middleware solutions came into being, from the likes of:

- **TIBCO**
- **New Era of Networks (now part of Sybase)**
- **WebLogic**
- **Vitria**
- **STC**
- **Crossworlds**

to name but a few of the 40-plus application broker products that currently are available for purchase.

The application server emerges

But then something happened. These same application broker companies realized that the applications themselves — the programs that enterprises were using — were becoming bigger and more complex, both to write and maintain. At the same time, pressures were increasing for applications to share more of their data (and, sometimes, functionality). More applications were either located on a network or used networks extensively.

It seemed logical, therefore, to have some kind of program residing on the network that would help share application capabilities in an organized and efficient way — including making it easier to write, manage and maintain the applications. The end result of this thinking is what is now called an 'application server'.

These application servers first appeared in client/server computing and on LANs, and were often associated with 'tiered' applications. People described the functionality of applications as:

- **two-tiered (database and client program) — which was the original client/server**
- **three-tiered (database, client program, and application server)**
- **n-tiered (all of the above plus whatever).**

The net effect has been (and still is) to complicate application development. Furthermore, such complexity inhibited consistent adoption (the implementations were many and various, even within one organization).

The Internet, the Web, B2B and application servers

The Internet, the World Wide Web and the desire for a B2B trading model changed all this. The Web-based B2B trading model is automatically three tiered (database, B2B trading program and Web server). Managing data along with application functionality suddenly becames not only an exercise in better program design but a necessity. It was this that vaulted the application server from obscurity to the top of the pedestal. Scores of vendors

jumped in to develop application server products (see **MIDDLEWARESPECTRA**, May 2001, page 44).

Unsurprisingly, not all vendors see the role of the application server in an identical way. They were not competing just to make everything identical. The result is a world where different application servers have different roles. The claimed justification is that not every customer requires the same functionality.

Scalability is a good example. Some organizations want an application server that simply helps them organize their applications for the Web, to give them better control over the business logic they contain or to make it easier to monitor and secure the data. They do not need thousands of servers.

Yet other organizations, especially large ones, do need to manage thousands of servers. For them, scalability in an application server is crucial.

So some application servers feature scalability, while others proclaim other strengths. Some even try to do everything (usually without success).

One more aspect needs mention. It is (of course) yet another complication. Application server products belong to a variety of programming regimes. Most, though not all, are written in Java. Some are Microsoft friendly; others are not. This latter distinction shows up in support for J2EE, CORBA or Microsoft COM+ (and, of course, some try to support all).

And so ... back to the trading platform vendors

Having said all this, what have the Internet trading platform vendors been doing about their requirements? The answers may come as a surprise and depend on which vendors are examined.

The costs of supporting multiple standards for integration, communications, information and security are already prohibitive. While ITPVs could, by adopting 'transparent' trading engines, offer customers reduced IT administration costs and achieve full integration of front and back office technologies, most have been slow in delivering real solutions.

Generally speaking, there are three groups of ITPVs. These groups are roughly gathered according to the age of their respective product(s).

Group 1 of the ITPVs includes those that were the first to bring out Internet trading solutions. Initially, their main focus was on the trading engine itself, with little or no regard for much else in terms of application integration and/or legacy support of the rest of an organization's enterprise. Since those initial product offerings, most of the companies in Group 1 have recognized their myopia and have taken steps to address integration issues. Companies like Commerce One and Ariba fall into the Group 1 category.

The second set of ITPVs (Group 2) includes those that, while quickly learning some of the integration shortcomings of the Group 1, built integration tools and solutions from scratch — each using proprietary technologies, although with some attention paid to trying to adhere to industry standards. The trouble was, few of the standards had been formalized.

Companies such as Moai and VerticalNet Solutions (the recently merged VerticalNet and Tradeum) provide examples of a serious attempt — but in a non-serious way — to bridge the integration gap. Most of these companies, too, have realized that their integration solutions left much to be desired.

The third set of ITPVs (Group 3) are those that came late to the party. Their tardiness may have been well worth the wait. These are the companies that decided to build their platforms around standards-compliant architectures, most commonly Sun's Java and J2EE. Companies such as @theMoment, ecMarket and NextSet are in this category.

These Group 3 vendors have built their trading engines on a Java-based transparent architecture. They provide interoperability with existing industry standards while offering open-ended adaptability. Their aspiration is to protect customers' IT investments as new technologies evolve.

Comparisons

A recent analysis of 17 ITPVs bears out this analysis. Figure 4.1 rates the eight vendors already discussed plus a couple more, according to their technical capabilities. Attributes were assessed according to a ten-point, non-weighted rating with accumulated results. The attributes were objectively scored against requirements for trading system solutions.

It should not be surprising, but close examination of the analysis indicates that the level of 'openness' and middleware integration offered by the ITPVs is related to two factors:

- **the age of the product**
- **the depth of that ITPV's pockets.**

But, regardless of whatever group an ITPV is considered a part, it should be quite clear that most, if not all, of these vendors are rushing to complement their trading engines with a viable form of middleware based on that offered by application server vendors. Indeed, multi-faceted alliances are being struck between these 'front-office' ITPVs and the 'network plumbing' application server companies.

While there is much movement in this area — and a serious attempt to improve the capabilities of enterprise applications to be able to work with trading system platforms — interesting questions still remain:

- **which trading platforms integrate with which application servers (and how easily)?**
- **which 'back office' ERP systems integrate with both trading platforms and application servers?**
- **what complementary standards (if any), do the application servers and the ITPVs embrace?**

Therein lies today's challenge. ITPVs cannot possibly integrate their products with all of the leading application server products (there are over 40 of these; again, see also **MIDDLEWARESPECTRA**, February 2001, page 44). Likewise, application server vendors would not (and could not) do well in supporting the myriad of trading engine products available today.

Middle of a muddle

The probable result — unfortunately — is likely to be a mess. By way of illustration, consider Figure 4.2, which lists the same aforementioned trading platform vendors and the application servers that they are supporting, either through partnerships or common sense.

While Figure 4.2 lists all of the current corporate relationships among ITPVs and application server vendors, determining what level of integration each partnership supports is anything but easy to ascertain. Unfortunately, ascertaining just what percentage or level of integration these alliances entail requires real experience. But to obtain this you have to buy both (trading platform and application server). This is expensive.

Figure 4.1: ITPVs analyzed

Trading Platforms Comparison Matrix

Attribute/Requirement (score 1-10)	Commerce One	Ariba	Moai	Tradeum	NextSet	At The Moment	EcMarket	Oracle	BusinessBots	TradeTech
Hardware Platform Independence	6	8	9	8	9	7	5	8	9	4
Software (e.g. C, C++, Java, EJB)	4	8	8	7	9	8	6	8	9	8
Open, published, and accessible APIs	3	4	7	4	8	8	7	3	8	3
Component - based Design	7	6	7	4	8	9	7	8	8	3
Open Information Schema (XML, ASN)	7	8	9	8	9	9	8	8	9	3
Data Base Independence	4	7	7	8	9	9	8	1	7	3
Two Phase commit XA interface	3	3	3	3	6	6	3	3	3	8
Interface to back office systems	7	5	2	8	6	8	4	5	6	6
Service based architecture	5	7	8	4	7	9	7	5	8	2
Middleware does most of the grunt work	3	4	6	5	7	8	8	6	6	4
Availability/Failover/Error Recovery	6	8	7	8	7	6	8	7	6	8
Scalability (both up and down)	7	7	8	8	8	8	7	8	6	8
Interface to Sys Mgt products	4	6	5	5	4	6	8	5	4	4
Security (SSL, PKI)	7	9	5	9	9	9	9	7	7	7
Messaging	6	8	2	5	8	9	3	8	4	7
Technology Score:	79	98	93	94	114	119	98	90	100	78

It should come as no surprise, therefore, to discover that the majority of application server alliances, while making for compelling reading on press releases, do not fully embrace nor fully integrate their products with the trading platforms. Put another way, it continues to appear that no one ITPV cares to 'lock itself to' a specific application server vendor, and vice versa. Because of such middleware integration uncertainties, corporate IT managers who may have been hoping to facilitate their business vision (of connecting to the Internet in order to conduct e-business) once again find themselves in the middle of a muddle.

What is also interesting to note, regarding the trading platform/application server relationships, is that the apparent market leaders in both categories seem to have chosen each other, while the 'also rans' in each group appear to have found one other. Once again, it is not technology worth that determines a product's success, but rather the marketing and the corporate market share that holds sway. Additionally, very few Internet ITPVs offer application server integration with more than two or three application server vendors.

Yet, if one bright spot does exist in this miasma, is that those trading engines that were built using the Java and EJB technology can avail themselves of any application server that complies with the Java J2EE standard. This appears to be where trading engine technology should be headed, if it is not already. It is also the one criterion that can be usefully applied by users. Only buy a trading platform if it supports J2EE.

Management conclusion

What have the Internet trading platform vendors done with respect to integrating their products with an organization's IT enterprise? They have:

- ***first, built their own middleware that was tightly coupled to the problems they were trying to solve, and quickly learned that this was a path down a blind alley***
- ***next, realized that there was more to corporate enterprises (in the way of legacy applications and back office processing) than they were originally either willing to admit or accept***
- ***eventually, turned to application server vendors for a solution, deciding that it was better to buy than to build.***

Fortunately, the key to the hoped-for solution has been the Internet. Trading engine and application server vendors are accepting that it makes more sense to join, rather than oppose, what the Internet community adopts.

Figure 4.2: ITPV Corporate relationships

Trading Platform	Supported Application Servers/Vendors
Commerce One	New Era of Networks (NEON) iPlanet Application Server (iPlanet) HAHTsite Scenario Server (HAHT Software) **Aptivity** Application Server (Progress Software) Microfocus Server Express (Merant)
Ariba	BEA WebLogic (BEA Systems) IBM WebSphere (IBM) TIBCO ActiveExchange (TIBCO Software) **B**usiness Broker (Mercator))
Moai	BEA/ WebLogic (BEA Systems)
VerticalNet Solutions	Versant enJin (Versant)
NextSet	BEA WebLogic (BEA Systems) IBM WebSphere (IBM) Smartsockets (Talarian) Other EJB-compliant application servers
At The Moment	TIBCO ActiveExchange (TIBCO Software)
EcMarket	Capstan Network (Capstan)
Oracle	Oracle Application Server (Oracle)
BizBots	Capstan Network (Capstan)
Trading Technologies	Proprietary solution

How this will actually happen is still unclear. But the precedents set by the likes of TCP/IP, HTML and XML are there for all to see.

The hope is for the continued emergence and enhancement of a set of standardized trading platform services commensurate with an already 'conventionalized' set of middleware services. This is entirely necessary because:

- ***the world has become wired***
- ***applications have become more and more complex***
- ***enterprises need to deal with each other (and not just their internal systems)***
- ***IT managers know that they cannot afford endless duplication of function, or incompatibility***
- ***the Internet has shown that there is a way forward which works.***

The CORBA Component Model (CCM)

Tom Welsh
Consultant

Management introduction

The Common Object Request Broker Architecture (CORBA) has always been open to the criticism that it was too complicated, too low-level and too difficult for the average developer. Moreover, CORBA2 did not, and does not, provide a component architecture. Arguably, it is little more than a software bus — an intelligent pipe between applications.

To compete with COM+, .Net and Java, the Object Management Group (OMG) members felt that CORBA needed an extra dimension. The result was the CORBA Component Model (or CCM).

In this analysis, Tom Welsh examines CCM and places it in both its historical context as well as positioning it in terms of .Net and the various Java initiatives, especially J2EE. He also assesses where CCM is going and from where the principle developments are likely to come, now that many of its original commercial vendor proponents (including Oracle, Sun and IBM) seem to have lost interest.

Setting the scene

The idea that eventually turned into the CORBA Component Model (CCM) was first aired in a document which appeared in May 1997 — called 'CORBA Component Initiatives' and authored by:

- **IBM**
- **Netscape**
- **Oracle**
- **Sun.**

It began by explaining the requirements: 'With IIOP interoperability for CORBA, and a diverse set of distributed object services, the OMG's Object Management Architecture has come of age as a tool for expert programmers to generate complex multi-tier systems. It is now time for OMA and CORBA to take the next step, by providing a component framework enabling CORBA objects, representing both client-side and non-visual server-side functionality, to be assembled into systems, using visual development tools and scripting languages, by unsophisticated programmers.' Noting that 'there will never be very many expert programmers', the document described several desirable features of a future CORBA component architecture, all of which have found their way into the CCM specification of today.

A Request for Proposal (RFP) was issued in June 1997, and no fewer than seven submissions were lodged by the November deadline. After nearly two and a half years, a specification — or rather a related set of specifications — had finally been adopted. At that point OMG's work on CCM would have been done (for the time being), had it not been for the consortium's new finalization process.

After some bad experiences with previous specifications (that had proved difficult — or even impossible — to implement properly), OMG decided to add an extra stage to its technology adoption process. After a specification was adopted, it added a requirement which obliged a scrutiny be undertaken by a finalization task force (FTF). The idea was that this would perform the first maintenance revision, in the light of feedback from pioneer implementers.

CCM entered the finalization stage in November 1999. It is still not complete. So many issues have been identified that agreement remains oustanding. Having announced CORBA 3 — of which CCM is a major element — in September 1998, the OMG is uncomfortably aware that the clock is ticking. The likeliest outcome seems to be that the essential core of CCM will be finalized by the deadline (end-2001), with some of the trickier parts such as Component IDL (CIDL) being deferred or possibly redesigned.

The rationale for CORBA components

For its first eight years, CORBA had little to do with components. In fact, as the first language binding was to C, it did not even require the use of objects. Essentially, it (pre-CCM CORBA) was built around an improved remote procedure call (RPC) with support for polymorphism and some other object oriented features.

To this day, a typical CORBA system comprises a set of client stubs — remote method interfaces, which behave like local ones — plus a corresponding set of server method interfaces. The rest is 'plumbing' — indispensable but transparent to the application programmer.

In this scenario, components are optional. The client can be a component, or a piece of hand-written C++ or COBOL code. Exactly the same applies to server objects, which usually belong to a single application program (although they can be invoked separately). Before CCM, CORBA did not concern itself with how client and server objects were written.

Thousands of CORBA projects have been undertaken in the last few years, and certain successful design techniques have emerged. CCM packages some of these patterns, promising to lower the cost and reduce the risk of using CORBA right across the board.

CCM, therefore, encapsulates ten years of practical experience in building distributed object systems, incorporating:

- **Portable Object Adapter (POA) servant management**
- **transactions**
- **persistence**
- **security**
- **event handling**
- **configuration**
- **interface connection.**

On this basis, OMG expects that developers using automated CCM tools will be able to put together

secure, reliable, efficient and scalable CORBA systems in much less time than hitherto. Such tools should automatically generate up to 90% of project code. In other words, applications will require only one tenth as much hand-written code as equivalent applications produced with conventional techniques. Moreover, the generated code will be exceptionally reliable and efficient.

As such, CCM goes way beyond CORBA. As Jon Siegel, OMG's director of technology transfer, puts it in the book 'Quick CORBA 3' (John Wiley, 2001): 'CCM is designed for large, distributed enterprise and Internet applications that need to run with transactional assurance, security and high throughput, and are best coded by programmers whose skill and knowledge is of business rules and processes, rather than the esoterics of server-side scalability patterns and the nuances of two-phase commit and rollback.'

The CCM and related specifications

Because it is firmly based on the existing CORBA and CORBAservice specifications, CCM cannot be viewed in isolation. It is an interlocking set of specifications, comprising the following main parts:

- **CORBA Components, including an Abstract Component Model (expressed as extensions to the Interface Definition Language [IDL] and the object model) and a Component Implementation Framework (CIF), centered on the new Component Implementation Definition Language (CIDL)**
- **the Component Container Programming Model, expressed as two alternative views — the component implementor and client view — as well as the the container provider view**
- **integration, with persistence, transactions and events (through the respective CORBAservices)**
- **component packaging and deployment**
- **interworking with EJB 1.1**
- **the Component Metadata Model, expressed as extensions to the Interface Repository and the Meta Object Facility (MOF).**

In this context 'component' is a new metatype in the IDL. Whereas previously a developer began by defining interfaces, there is now the alternative of defining components — and associating one or more interfaces with each component.

Components come, therefore, in two types:

- **basic**
- **extended.**

CCM basic components

Basic components provide a simple way to 'componentize' an ordinary CORBA object, without significantly changing its programming model. A basic component has only one interface and one thread of execution. It may use transactions, security and simple persistence for a single segment. But it does not have access to the event model.

Basic CORBA Components are functionally equivalent to EJB 1.1, a design decision from which important consequences flow. It means that CCM:

- **provides a blueprint for a vendor developing an EJB server based on CORBA**
- **an EJB deployed into a CCM container automatically becomes a CORBA Component**
- **Java clients using the EJB programming model can view CORBA Components as if they were EJBs**
- **CORBA clients can view EJBs as if they were CORBA Components.**

Like EJB, CCM is strictly a server-side architecture: clients cannot include CORBA components. Furthermore, to access the extended CCM functionality, a client must use a CORBA 3 ORB. (None of these exist yet, but then neither does the extended CCM functionality: more on this later).

A client based on a CORBA 2 ORB is referred to as being component-unaware. It is restricted to invoking basic component functionality.

CCM extended components

Extended components provide a richer set of functionality than the existing CORBA model. At this level, a component:

- **supports multiple interfaces and allows navigation between them (previously, a CORBA object could have only one interface)**
- **can make use of multi-threading and the CCM event model.**

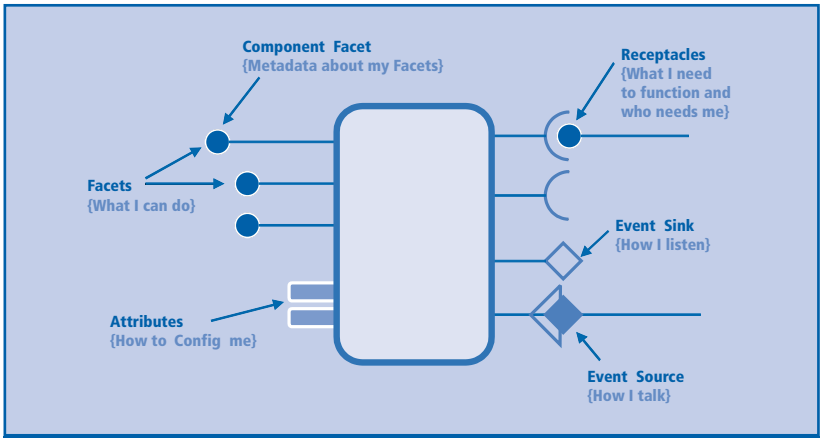


Figure 5.1: Structure of a CORBA Component (Source: OMG)

The Multiple Interfaces specification permits a single object to present multiple views of itself through an interface selection mechanism. This is much like COM's IUnknown — although CORBA advocates point out that it returns a list of interfaces, whereas IUnknown's QueryInterface mechanism has to be interrogated separately for each interface.

Multiple interfaces are known as facets (Figure 5.1). A client can navigate from any of a component's facets to its main facet, which provides a list of all supported facets. Persistence, too, can be controlled at the facet level as well as for an entire component.

Receptacles are the client stubs that a component uses to invoke other component interfaces such as are described in its configuration file. Event sources and sinks are named connection points through which events can be emitted or consumed using the Notification Service (CCM's equivalent of Java Message Service). Attributes and configuration are named values exposed through 'accessors' and mutators', which are mainly used for configuration. Entity components, like entity EJBs, have primary keys that clients can use to retrieve particular sets of data. (This is the first time that the CORBA has supported persistence in a way that is visible to clients).

Other considerations

Like EJBs, CORBA Components have home interfaces that provide standard factory and finder operations. Instead of deploying individual component instances, CCM installs a factory object for each component. When a request is received, the factory may either create a new component instance to deal with it, or activate an existing instance that is in an idle state.

CCM uses XML descriptors to label components with packaging and deployment information. It also provides assembly descriptors, which show how two or more CORBA Components should be interconnected. For greater flexibility, components can be linked into an assembly. Alternatively, if more convenient, a single large component can be divided into segments. Components and descriptors are packaged into Component Archive files (CARs), just as EJBs are shipped in JARs.

Although CORBA components can be written in any language for which there is an IDL mapping, a new scripting language (IDLscript) has been designed for the purpose. Because scripts will run in the context of CCM containers, the language does not need complexities such as memory management, pointers, or even compilation and linking. Scripting accelerates the development cycle, and is especially useful for prototyping.

Containers and policies

Like EJB, CCM relies on the idea of a container within which components can be installed and run. A CCM container is a specialised POA (Portable Object Adapter) with some built-in CORBAservices (persistence, transactions, security and notification) which can be distributed across multiple computers for load balancing and fault tolerance.

The POA is extremely complex, having literally hundreds of 'activation/passivation' settings that give developers enormous flexibility. But, by 'hiding' this complexity inside a relatively simply container, CCM greatly reduces the technical expertise required to get an application server up and running.

Containers come in four main types, three of which are exactly like the corresponding EJB types

(Figure 5.2). The fourth type — Process — has the same server characteristics as an Entity component, but does not expose its primary key to the client.

CCM also supports four progressive servant lifetime policies (Method, Transaction, Component and Container). By offering a wide range of memory residency periods for activated servants, these enable programmers to make performance tradeoffs such as response time against scalability.

The Method policy, for instance, requires a servant to be activated just long enough to perform a single operation. This is a suitable policy for short, relatively infrequent operations.

The Container policy, on the other hand, lets the container decide when the servant is passivated. This opens the door to various heuristic optimizations.

CCM Roles

CCM divides the responsibilities involved in the creation of a server application into the following six roles (these are similar to analogous EJB roles):

- **server provider: this delivers a suitable development and run time environment that is transactional, reliable, scalable and secure; however, this environment does not need to conform to the CCM or EJB interfaces (that is the job of the container and delivering this is regarded as being a system programming task)**
- **container provider: this delivers a container that supports all the CCM or EJB interfaces on its 'inside', while interfacing with the server on its 'outside'; configuration, installation and run time support are part of the container package (again this is regarded as a system programming task)**
- **component provider: this role requires an expert in the relevant application domain (banking, merchandizing, celestial dynamics, etc.) who is also a competent programmer; however, specialized tasks like transaction control, security and scalability are taken care of by the container and the server — so this is an application programming task**
- **application assembler: this role entails building an application from components (which may have been bought in, previously written or created ad hoc); if all the required components already exist, it may require no actual programming skill — or the assembler may be the same person as the component provider**
- **application deployer: this person takes special distribution files, deploys and configures them so that any and all references are resolved**
- **system administrator: this person ensures that the application starts up and runs smoothly, if necessary adjusting run time descriptors to control the behavior of components.**

Implementations and progress

As of September 1999, the following organizations had made commitments to support CCM:

- **BEA Systems**
- **the Co-operative Research Centre for Distributed Systems Technology**
- **Expersoft (now part of Vertel)**
- **Fujitsu**
- **Genesis Development**
- **HP**
- **IBM**
- **Inprise**
- **Iona**
- **Oracle**
- **Rogue Wave**
- **Sun**
- **UBS.**

Among these are no fewer than five of the leading J2EE/EJB application server vendors (including BEA, IBM and Sun). Yet, for various reasons, none of the above are known to have made significant progress toward creating CCM products. The delays in finalization, coupled with the distinct lack of obvious demand, probably account for this state of affairs.

In addition, those vendors with J2EE aspirations and products already have their work cut out just to keep up with the rapid pace of the Java specifications; in addition, they must compete with each other. Any effort committed to CCM inevitably dilutes their J2EE focus, and vice versa.

That is not to say that no progress is being made. A number of CCM projects appear to be under way, but in several cases little or nothing has been published:

- **Exolab.org (of OpenORB fame) is said to be working on a CORBA 3 ORB**
- **the authors of JavaORB are also believed to have a CCM project in mind.**

There are four other initiatives involved in the development of the first complete CCM implementation.

Component Category	Container Implementation Type	Container Type	External Type	EJB Equivalent
Service	Stateless	Session	Keyless	Session (Stateless)
Session	Conversational	Session	Keyless	Session (Stateful)
Process	Durable	Entity	Keyless	—
Entity	Durable	Entity	Keyful	Entity

Figure 5.2: CCM Component Categories (Source: OMG)

There is also fragmentary evidence of several other organizations' commitment to CCM — for instance Siemens, One, Sprint and Eurescom/GMD/Humboldt University. This is a case where the Open Source community is making a valuable contribution by writing code without the incentive of profit. [Diego Sevilla Ruiz maintains an excellent Website dedicated to tracking all the latest CCM information. Its URL is http://ditec.um.es/~dsevilla/ccm/.]

EJB 1.1 does not necessarily equal basic CCM

It might conceivably be argued that any EJB 1.1 server is automatically compliant with the basic level of CCM functionality. But this would be based on a misunderstanding.

It is true that basic CCM components are more or less compatible and interoperable with EJB 1.1 (although not necessarily the other way around, unless great care is taken). It is not a two-way proposition. As already stated, the Basic level of CCM is functionally equivalent to EJB 1.1. In principle any compliant EJB can be deployed into a CCM container, and will then behave like a CORBA Component written in Java. Extended CCM is a superset of EJB, offering additional features such as:

- **distributed event handling**
- **multiple interfaces and navigation**
- **segmented persistence**
- **multithreading.**

CCM also goes beyond EJB in being language-neutral. Although any given implementation needs support only one language — C++ and Java are already the favorites — the CCM specification is compatible with all languages for which OMG IDL mappings exist, including Ada, C, C++, COBOL, IDLscript, Java, LISP, Python and Smalltalk; unofficial mappings exist for several other languages.

It should also be noted that J2EE, which embraces EJB, draws heavily upon CORBA:

- **the JNDI directory service is integrated with the Naming CORBAservice**
- **the Java Transaction Service (JTS) and Java Transaction API (JTA) are based on the CORBA Object Transaction Service**
- **J2EE 1.3 requires that EJBs communicate with one another using IIOP**
- **J2EE security has much in common with OMG's Common Secure Interoperability Version 2 (CSIV2)**
- **J2EE includes J2SE, which has its own CORBA ORB.**

Does CCM have a future?

CCM is an ambitious undertaking. Even with all the previous work on CORBA, COM+ and EJB to draw on, it is not surprising that some of the world's leading experts are taking years to get it right. They know that:

- **COM+ is a local object model which was subsequently extended across the network but is effectively limited to a single family of operating systems (Windows)**

- **JavaBeans and EJB are sophisticated component models, but were designed around a single language.**

iCMG

Internet Component Management Group (iCMG) is a company set up by CORBA expert Thomas J Mowbray and others, to explore the potential of component-based and distributed object software. iCMG itself is spread around geographically, with twin headquarters in Virginia, USA and Bangalore, India (details of iCMG and its products can be found at http://www.componentworld.nu/).

Currently iCMG is working on two CCM projects:

- **the K2 Component Server**
- **the K2-CCM Container.**

An EJB-CCM bridge is also provided, so that EJB clients can invoke CORBA Components through EJB views.

OpenCCM

LIFL (the computing department at the Université des Sciences et Technologies de Lille, in France) has announced the availability of source code for OpenCCM. This project aims 'to design, implement, compile, package, deploy, and execute distributed applications compliant with [CCM]'.

At present OpenCCM is a work in progress. Java is the only language supported and some areas — such as CIDL and CIF — have not been done (this was inevitable, as the corresponding specifications are incomplete and look like being deferred still further). However the existing code has been successfully tested on Solaris, Linux and Windows 2000 with:

- **three different ORBs**
- **two IDLscript interpreters.**

Support for C++, packaging and deployment are planned, along with other features.

Figure 5.3: CCM activities

COM+ (and its ancestors OLE2, COM and DCOM) were built right into the fabric of 32-bit Windows; they also became standard tools for the millions of Windows developers outside Microsoft. Naturally, they have seen a lot of use. The same applies, in a lesser degree, to Java.

CORBA, on the other hand, has never enjoyed such phenomenal popularity. It has mostly been deployed in a pragmatic spirit, to solve problems for which no other solution were adequate.

In theory, CCM combines the language neutrality of COM+ with the platform neutrality of EJB, while preserving upward compatibility with the existing CORBA infrastructure. It is a fully abstract server-side component model, of which EJB is a subset.

Commercial impetus

Nor does there appear to be there any commercial impetus to deliver CCM. EJB/J2EE are already out there holding the line against Microsoft's server-side line up. They have the lion's share of attention.

EJB has been strikingly successful by most criteria. After a brisk specification process, it was quickly implemented by dozens of vendors — the first product, WebLogic's Tengah (now part of BEA), appearing just as the standard was formally announced. Taking that performance as a ten, CCM has earned something like a three so far. With the best will in the world, and no obvious conflict or mistakes, it has somehow managed to hit every rock and run aground on every sandbar.

For a start, CCM is pushing the limits of the OMG's finalization process. The current deadline falls toward the end of 2001, and there is just not enough time left to resolve all the open issues. Nothing OMG has done before has been so complex. The ramifications of CCM seem to extend everywhere, from the ORB Core to the Security CORBAservice. More critically, it has become clear that certain parts of the specification — such as the CIDL and the deployment model — simply cannot be fixed in the time available. Fortunately, these parts are not mandatory and so might be deferred to a later date.

Having said all this, specification and implementation are mutually dependent. The best specification in the world has no practical value until it is

productized. Early implementations inevitably reveal gaps, weaknesses or actual mistakes in the specification. The result is that CCM is in a curious near-deadlock: implementations are few and far between because the specification is not finished, while the FTF is handicapped by a lack of practical experience.

Timing has not favored CCM, which was first mooted in 1997, at much the same time as EJB. While the various CCM submissions were being discussed, EJB was published — and Microsoft's first COM+ beta surfaced. By the time CCM was adopted at the end of 1999, EJB enthusiasm was in full swing. Then J2EE came along, filling in many of the holes in the EJB value proposition.

Today there are between 30 and 40 production-quality J2EE servers which leaves few potential takers for CCM in the short term. Instead, most CCM development work is being undertaken by Open Source teams (Figure 5.3) — in many cases sponsored and funded by telecommunications companies or government agencies. Whether these can produce anything fast enough to contain J2EE and/or .Net will determine whether CCM has a future.

Management conclusion

It is tempting to write CCM off as an experiment that failed — yet another grandiose ivory tower dream like OSI or an AD/Cycle. That might be a short-sighted view, especially if CCM finds a foundation in the Open Source world.

It is true, however, that CCM will probably not yield any very exciting dividends for some years to come. But, if you are honest, you will have to admit that neither J2EE nor Microsoft's alternative .Net solution are ideal (one is language-specific and the other is platform-specific — and rigidly locked in to a single vendor). Undoubtedly, in an ideal world, it is preferable to have the option of a distributed component model that is language-neutral, platform-neutral and vendor-neutral.

That is the promise of CCM. But its promise is now dependent on an Open Source community delivering what is needed. If this occurs, CCM may yet take its rightful place. But, with the major vendors concentrating on J2EE and .Net, it is going to take a tremendous effort and success from the Open Source community to overcome the momentum and investment of the 'alternatives'.

MicoCCM

Frank Pilhofer, one of the developers of the MICO Open Source ORB, stated in November 2000 that he was starting work on a CCM implementation based on MICO. Although Alcatel is funding the work, all resulting code will become part of the Open Source MICO distribution.

This software is known as MicoCCM. It is still at the prototype stage and currently runs only on Linux, with the GNU gcc compiler. Service and Session components are supported, and can have attributes, facets and receptacles. The event model is not yet available but the first partial version of MicoCCM is scheduled for July 2001.

TAO

TAO ('The ACE ORB'), which stems from research at Washington University in St. Louis, is by far the best-funded and most heavily staffed Open Source middleware project. With hundreds of contributors, it is supported by the US government's Defense Advanced Research Projects Agency (DARPA) as well as Boeing, Cisco, Ericsson, Microsoft and many other corporations. It is marketed commercially by OCI — in much the same way as Linux is sold and supported by Red Hat and others.

Together with Real-Time CORBA, Fault-Tolerant CORBA, and almost every other CORBA specification, the TAO team plans to create a CCM implementation. No timescales have yet been published.

Figure 5.3: CCM activities (continued)

‚Java — middleware or 'just another programming language'?

Keith Jones
IBM Software Solutions Worldwide

Management introduction

Critics have often observed that without XML — and more recently Web services technology — Java would now be just another programming language. There are many who would like this to be true.

In this analysis, Keith Jones — recently back from JavaOne — examines the volume of activity around Java. He explores the language, the virtual machine, the platform libraries and application components which, he argues, proves that Java:

- *is not 'just another programming language'*
- *reflects a continuing evolution that continues to deliver significant value at multiple points.*

State of the art

The Java language was first introduced in the summer of 1996. Every year since, there has been a pilgrimage by growing numbers of the faithful to JavaOne in San Francisco. 2001 was no exception. The 20,000 programmers, engineers, managers, consultants and other believers met as expected for a week to discuss the new developments, to exchange notes and experiences and to define the current state of the art in the Java world.

If it is hard to believe that there can be this much continued focus and attention paid to a programming language — however elegant or sophisticated it may be — the truth is that Java is really much more than just a programming language. This applies even after you take into consideration its:

- **strength of type guarantees**
- **elegant approach to memory management**
- **machine independence, as afforded by its bytecode technology.**

A common runtime execution environment, a fast expanding set of functional libraries and an emerging set of re-usable components, all these complement the language definition. The result is 'Java, the phenomenon' rather than 'Java the language'.

This is borne out by the growing community of proponents amongst product vendors, users from enterprises (large and small) as well as standards organizations. It is this concerted combination which maintains Java's forward momentum.

Java — the language

The programming community often likes to define the value of Java in terms of its remedies for the notorious deficiencies and complexities in C++. Java has now displaced C++ — and Visual Basic — as the language most often taught in Universities. It is the language most often used for coding commercial applications. Last year, at JavaOne, one of the reasons for this was clear — its focus on Java as the language for a new class of real time applications.

If the number of changes to the Java language proposed by the developer community is a reasonable measure of stability and maturity, then the language alone is already delivering the value expected. With the exception of some concerns over the atomicity of changes to memory in highly threaded environments (and proposals to add such extensions as an Assertion capability), 'Java, the language' is already serving programmers well.

Java — the Virtual Machine

The 'write once, run anywhere' promise that comes with Java would not be possible without common runtime environments available across a wide variety of systems platforms. Critics point, however, to platform peculiarities as evidence that Java is failing to live up to this promise.

At JavaOne this year it was evident that Java Virtual Machines (JVMs) are now available on large numbers of cell phones, handheld assistants, set-top boxes and all manner of other network capable devices — in addition to its traditional home on desktop and server machines. That promise, of 'write-once, run anywhere' is being realized to some very significant degree, even though some low end virtual machines are necessarily extremely compact and do not always support all aspects of the language and available platform libraries.

Java has also been criticized for being an interpreted language and not compiled for execution speed. However, recent developments in virtual machine technology and just-in-time compilers are addressing this concern — by focusing on the most often used bytecodes at run time and by introducing optimizations to them on the fly. This approach works well because it allows different platforms to optimize in the ways most appropriate to a particular platform's strengths.

Java — the platform libraries

At the 1999 JavaOne conference three different packages of Java library functions were identified for the first time — in place of the previous Java Development Kits. These packages signaled a new level of maturity in Java technology — now called Java 2 (in idle moments I have often wondered why the conference was not renamed JavaTwo).

Since then the J2ME, J2SE and J2EE packages have been the focus of a large and still growing number of Java Community members, including both users and vendors from every industry. Application scenarios, functional enrichment and compliance testing have all been included.

J2ME (Java 2 Micro Edition)

The Java 2 Micro Edition understands that some

Java platforms are constrained by the hardware resources available. Real time devices on the factory floor, cell phones, Personal Digital Assistants, and other devices in the consumer space currently have:

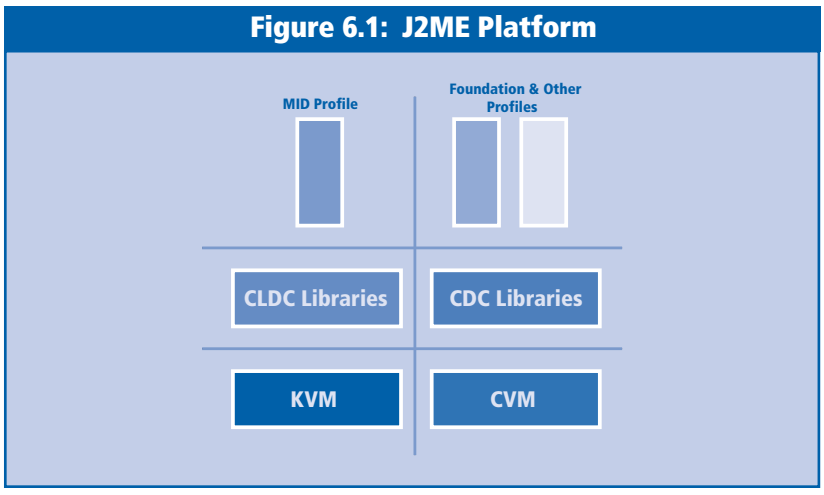
- **relatively little memory (compared to PCs and servers)**
- **few processor cycles**
- **narrow networking bandwidth**
- **modest video capability (again by comparison to desktops and server systems).**

A number of 'configurations' have been defined to recognize the different categories of device; different 'profiles' have been defined to support different modes of usage within those categories. For example, the most constrained devices are addressed by the Connected Limited Device Configuration (CLDC) and Foundation Profile that runs on a very small KVM (K for kilobyte — as in 'small'). The Mobile Information Device Profile (MIDP) is also defined for CLDC devices such as cell phones (Figure 6.1).

Larger devices are addressed by the Connected Device Configuration with Foundation and other Profiles running on a full JVM — now called a CVM. Smart communicators, paging devices, PDAs and set-top boxes are typical of the consumer category: these are assumed to have 32-bit processors and at least 2Mb of memory available for Java.

The J2ME platform specification may still be at the base level. But there are already over 100 million Java-enabled devices delivered — including 3 million Java enabled cell phones. One of the prerequisites for market success would seem to have been satisfied — a very large number of platforms acting as the base to attract vendors and developers to define extensions, new configurations and profiles now exists.

Figure 6.1: J2ME Platform



J2SE (Java 2 Standard Edition)

The Java 2 Standard Edition is the platform closest to the original Java 1 Development Kits used to create the first client applets and standalone applications. Now at its third revision level, J2SE 1.3 (Figure 6.2) is the core set of library functions used in conjunction with full-function Java Virtual Machines across a wide variety of platforms.

This core set includes support for:

- **GUIs (Graphical User Interfaces)**
- **networking and RMI (Remote Method Invocation)**
- **improved security (including signed applets)**
- **interoperation with CORBA systems (through JavaIDL and IIOP)**
- **database connectivity (JDBC)**
- **directory access (JNDI)**
- **performance enhancements and debugging interfaces.**

Much of the value provided is core Java language library function, but a significant proportion provides interface access to underlying operating system services or middleware product function. For example, the GUI support provides look and feel for applications running on Windows, Linux and other platforms.

Similarly, the networking library provides access to TCP/IP and other Internet protocols. Database connectivity support enables access to relational databases such as DB2, Oracle and others. The Java Naming and Directory Interface (JNDI) enables access to CORBA naming and LDAP directory servers.

While these do not offer the equivalent of full enterprise level capabilities, they do facilitate a large and useful set of applications.

The largest collection of extensions to Java is targeted for this platform. The next revision level (J2SE 1.4,

code-named Merlin) is expected to include:

- **XML parsing**
- **data binding extensions**
- **application event logging**
- **assertion language facilities.**

In addition, non-blocking I/O support will be added, as will extensions to Java database connectivity (to accommodate the latest ANSI SQL99 changes).

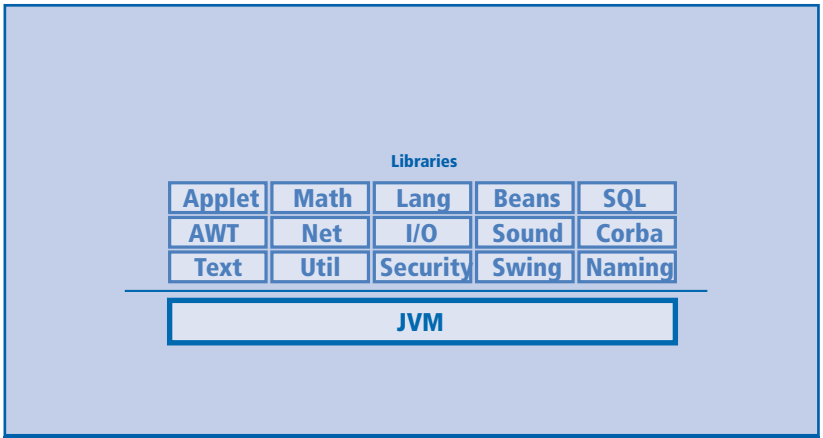


Figure 6.2: J2SE 1.3

Perhaps the most controversial topic amongst developers using the J2SE platform is that of client-side Java. The next version will include the capability to invoke full Java applications (as opposed to just applets) from a Web browser in a secure manner. Other initiatives are also being investigated to provide a more accessible Java client experience, but this lies in the future.

J2EE — Java 2 Enterprise Edition

The Java 2 Enterprise Edition has rapidly become the de-facto standard for the majority of the Java Web applications, and Web application servers, available today. 18 vendors have licensed this platform specification, and multiple products are already available. These include the leaders:

- **IBM, with WebSphere**
- **BEA, with WebLogic**
- **AOL/Sun's iPlanet**
- **Oracle's Application Server.**

To date, developers around the world have downloaded over one million copies of the current J2EE software development kit. As products emerge, many are being subjected to the J2EE Compatibility Test Suite with over 5000 tests included. A large and growing set of application development and systems management tools also support this 'mission critical Java'.

J2EE (Figure 6.3) is currently at the 1.2 specification level. Later this year the new 1.3 Specification will become final. Going beyond the current specification, this new level focuses on requirements such as:

- **management (JMX)**
- **performance (ECperf)**
- **interoperability**
- **messaging and query facilities (EJB).**

A new level of the Enterprise Java Bean (EJB) specification will be at the heart of J2EE 1.3 solutions. There will be improved support in EJB containers for entity beans and container-managed-persistence. Component programmers will also be able to use the new EJB Query Language for locating components in their homes using a SQL-like syntax. These improvements will extend the usefulness of the EJB component model.

The client view of EJB components has also been strengthened with both local (in the same JVM) and remote (accessible, using RMI) contracts. These allow for improved performance in some scenarios without losing Java typing guarantees. The contracts for Session Beans have also been extended to facilitate both state-less and state-full business logic components.

Future enterprise Java servers will also implement the new J2EE Connector Architecture (Figure 6.3). This provides a framework for adapters to be built and plugged in for a wide variety of legacy systems — such as R3, Siebel servers, CICS and other custom systems.

The reason for the development is recognition that one of the most difficult problems for Java programmers is connecting applications (and systems). Vendors will provide JCA (J2EE Connector Architecture) compliant adapters over time as Web applications reach back into central enterprise database and business logic servers.

J2EE and JMS

For the first time it will be mandatory for J2EE compliant systems to support fully the Java Message Service (JMS) specification. As Web application scenarios are extended to reach legacy systems, using asynchronous messaging (and enabling work flows between departments or enterprises), will be an important development for J2EE users. Many available J2EE application servers have already been enhanced to include a JMS solution.

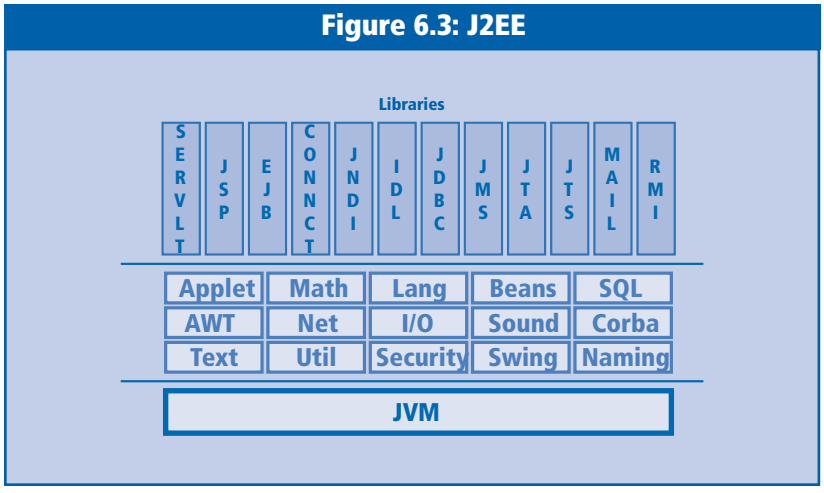
A new type of enterprise bean — called a message-driven bean — will be introduced in EJB 2.0 compliant systems. Message-driven beans will be created automatically by EJB containers to handle messages delivered to specified JMS destinations. As consumers of messages, these beans will invoke business logic to process message contents and take care of transactional and security requirements.

Although somewhat similar to session beans, message-driven beans will not exist in homes and will not have client interfaces for external access. These beans are intended to provide integration points for business logic components in asynchronous enterprise scenarios.

XML and Java 2

XML, the eXtensible Markup Language, is a universal syntax for describing and structuring data independently from application logic. It is desirable, and natural, that XML and Java should complement each other within the implementations of real world application scenarios. Already a number of Java XML interfaces and services are available to facilitate this — as optional packages for Java 2 platforms.

Figure 6.3: J2EE



The Java API for Processing XML documents (JAXP 1.1) — based on W3C standard SAX 2.0 and DOM level 2 interfaces — will be agreed and made available. This will enable applications to parse and transform XML documents using a pure Java API that is independent of a particular underlying XML middleware processor implementation. The proposed JDOM standard will further enhance this.

The Java API for XML Messaging (JAXM) enables applications to send and receive document-oriented XML messages using a pure Java API. JAXM implements messaging based on proposed standard Simple Object Access Protocol (SOAP) 1.1 with 'attachments' so that developers can focus on building, sending, receiving and decomposing messages for their applications — instead of programming low level XML communications logic.

Web services and Java 2

It is natural that, since most Web applications are already implemented in Java, the next generation — so called Web services — should also be based upon Java interfaces and underlying middleware. These applications will be accessible via the Internet using a variety of communications protocols and for a variety of dynamic e-business scenarios.

The Java API for XML Remote Procedure Calls (JAX-RPC) is a proposal that will enable Java programmers to marshal and un-marshal parameters needed for transmitting and receiving XML-based remote procedure calls. This function is needed to program Web service requests — expressed as XML documents — flowing between computers over the Internet.

The Java API for Registries (JAXR) is a new proposal that, if adopted, will enable Java programmers to gain access to the proposed standard distributed registries for Web service definitions (UDDI, ebXML and others) that are, in turn, based upon XML schemas. This API is analogous to the Java Naming and Directory Interface (JNDI) already included in J2EE platforms. But JAXR is specifically designed to handle Web service directory information.

The Java API for XML Binding (JAXB) is the last of a comprehensive

set of new APIs that are being considered for inclusion into J2EE for web services and as a package for download into existing Java application development environments. This API makes it easier for Java programmers to map XML data into Java objects without having to program at the Tag or DOM level.

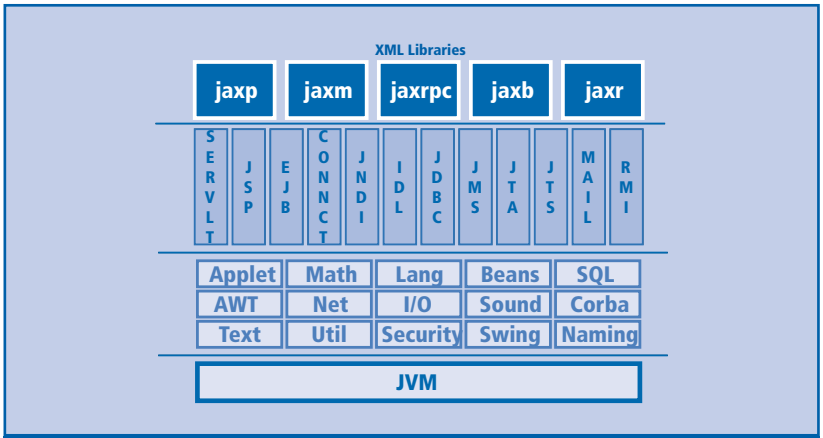


Figure 6.4: Java and Web services

Java APIs for XML are clearly an important requirement for enterprise Web services (Figure 6.4). A comprehensive program of work currently in execution within the Java Community focuses on the application programming model for Web services as the underlying technologies continue to evolve. Providing mechanisms to share security and transactional context between applications, for example, and EJBs that represent business processes, are typical of key areas not yet addressed in the J2EE specifications.

Java and re-usable components

Perhaps the most valuable promise that comes with J2EE is that re-usable components will facilitate a new level of developer productivity in the industry and thereby lower the cost of software production. There is already some evidence for believing in this promise. The most convincing argument is the emerging marketplace for Java components — applets, servlets, JSPs (Java Server Pages) and EJBs — which are suitable for inclusion in a wide variety of applications.

Some vendor products deliver such components at a basic level — for networking, graphical user display, database access and internationalization. Others offer higher level components — for sales, accounting, supplier integration and customer relationship management. Yet more offer complete collections of business processes. One example is the WebSphere Business Components package.

Further strength in this direction will come from the EJB 2.0 enhancements, specifically Java Community projects that seek to marry UML (Universal Mark-Up Language) with Java component technologies and vendor products that support re-use repositories and methodologies.

Management conclusion

Java has come a long way since it was first introduced six years ago. The language is still being revised but is already remarkably stable. That is not to say all is perfect in Java-land. The memory model and performance remain as major topics awaiting improvement.

The Java Virtual Machine is the focus of attention for a growing variety of platform vendors. Its likely success can already be measured by the installed base of tens or hundreds of millions of systems delivered, whether deployed or awaiting exploitation.

The language and virtual machine technology is the substrate for growing libraries of Java middleware function. Whether the platform is handheld, real time, embedded in a household appliance or in an enterprise information server, these libraries — and the re-usable components built upon them — are exploiting the underlying Java technology. Furthermore, much of the current focus is on enriching this value.

The timely conjunction — of wireless and Internet technologies with Java and XML technologies and service-oriented models for application construction — has dramatically changed the thinking behind middleware solutions for today and for the foreseeable future. Further development of this thinking appears to be bounded only by the laws of physics, our own imagination and a natural selection based on perceived value and, of course, actual usage.

MIDDLEWARESPECTRA **is published and distributed worldwide by:**

USA and Canada:
Spectrum Reports, Inc.

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

UK and Rest of the World:
Spectrum Reports Limited

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

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

Email and Internet

Email:
spectrum@middlewarespectra.com

World Wide Web:
www.middlewarespectra.com

ISSN 1356-9570