FINANCIAL MIDDLEWARESPECTRA

incorporating Enterprise Middleware

Contents

August 2002

2 **Systems, architecture and middleware challenges at Fidelity**
Don Haile, President, Fidelity Investment Systems Company

10 **Free Middleware: the surprising facts — Part II**
Tom Welsh, Consultant

18 **Of rules and middleware**
Steve Ross-Talbot, Chief Scientist, Enigmatec Corporation and Said Tabet, Co-Founder of RuleML

24 **Software — at your service**
Dr Keith Jones, IBM Worldwide Software Solutions

32 **Storage middleware**
Geoff. Norman, Consultant

42 **High availability considerations for messaging hubs**
Nick Denning, Chairman and Chief Technology Officer Strategic Thought

Volume 16 Report 3

Systems, architecture and middleware challenges at Fidelity

Don Haile
President
Fidelity Investment Systems Company

Management introduction

Don Haile is President of Fidelity Investments Systems Company, a division of Fidelity Investments, the largest US mutual fund company and a leading provider of financial services. In this position, Mr. Haile is the Chief Information Officer, responsible for Fidelity's computer operations, communications networks and enterprise-wide applications support and development. He is also a member of Fidelity's Global Technology Board which provides co-ordination and guidance to Fidelity's technology community.

Mr. Haile joined Fidelity in 1998 as executive vice president of Enterprise Solutions, in which capacity he was responsible for the worldwide development and maintenance of infrastructure and application products that supported the processing needs of Fidelity. He managed application development laboratories in Boston, Dallas, Salt Lake City and Dublin, Ireland. Prior to Fidelity, Mr. Haile was a senior executive at IBM where he was involved in both hardware and software development, including that relating to telecommunications, middleware, systems management and operating systems.

In this interview he reviews what he has learned, and unlearned, at Fidelity about systems, architectures and middleware in general, and delivery in particular.

All rights reserved; reproduction prohibited without prior written permission of the Publisher.
© 2002 Spectrum Reports Limited

Setting the scene

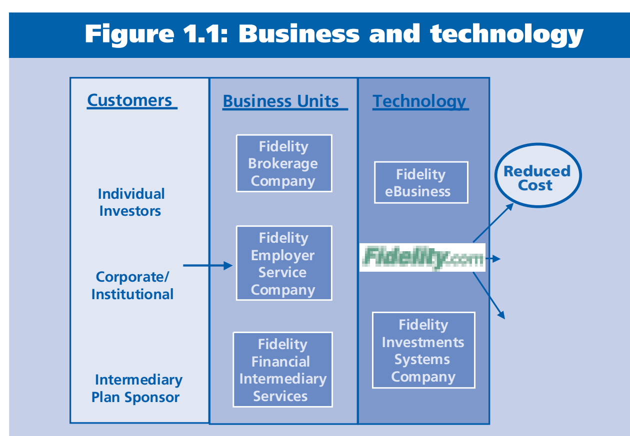
The Fidelity Information Technology organization is similar to the federal government; we have a central IT organization that provides the IT infrastructure and support systems for the entire Company. Individual business units — such as our Brokerage Company — are responsible for IT development; they are equivalent to state governments (Figure 1.1).

There are four major units, and over 20 smaller entities. Each is responsible for its own product research as well as the creation of marketing channels and financial offerings.

The Systems Company has the responsibility to provide the infrastructure — from desktop support to global networking to all critical processing capabilities. While we do not write business unit applications, we do run them in our data centers. In effect, the Systems Company manages all critical computing and network operations. The business units take responsibility for the creation of those applications that are specific to their business strategies.

There is one other distinction of importance — the Internet. We deliberately separated our Web development (from traditional development) and created a single Web approach across the Company (Figure 1.1). This pulls together the various distribution channels.

Figure 1.1: Business and technology



Why was this done? Early on, we recognized that the Web could become a unifying factor across different businesses and products. Let me illustrate why.

Fidelity sells mutual funds and offers brokerage trading over the Web. In addition, we offer employer services — for Defined Contribution plans, such as 401K or 403B products — to corporate customers. Our objective is to provide all of these products and services to those customers in a seamless way via one interface to all aspects of our business.

The result is a huge portal which accesses all of Fidelity's existing applications and infrastructure. Indeed, we have four or five different channels that customers can use to access Fidelity (Figure 1.2):

- **the Internet**
- **voice response units**
- **traditional phone representatives (we have several thousand phone representatives)**
- **the traditional way — postal mail: we receive a huge number of requests (for information or for actions to be taken) through regular postal services**
- **institutional access.**

In the last case, it is perhaps a little misleading to use the word 'portal' for we do more than just offer access. We provide a full service to institutions. For example you might go into your local Bank to transact a stock trade or buy a mutual fund. That Bank may sell Fidelity's products in order to provide you with what you want.

Operations matter

The operational environment at Fidelity has been operationally sound for many years and operated at a very high level for nearly a decade. The network is CISCO-based, with 400+ routers. Even before I arrived it was clearly described with a clear set of architectures in place. The day to day operational processes that had been put in place by previous management teams have served us well — for reliability is critical to our business. This gave, and gives us, the confidence that we can keep our businesses running in a consistent way.

The importance of system reliability has been critical, especially during the economic boom of the 1990s. In particular, between 1998 and the middle of 2000, trading volumes grew consistently. From a CICS transaction perspective we have processed close to 1 million (CICS) transactions a day — and, as a CICS transaction is anything that touches the databases, so a trade itself covers multiple CICS operations. When you add in all of the support processing — change of addresses, account set up, account inquiry, money transfers, etc. — the constant volume increases seemed absolutely unbelievable.

Similarly, on our Web portal sites, we had, and continue to have, 1.5M 'business' visits a week. These are not single touches on a Web page but customers coming in to look at

15 or 20 different Fidelity Web pages — crossing multiple business units.

The reason we have been able to handle the increases in volume is simple: we invested in the processes to bring our operations together. The Operations teams not only understand what has to be done each day but continually use internal metrics to predict, and then adjust, what needs to be in place to satisfy the ever increasing demand.

Size and architecture

The role of the CIO, given the above background, is to design, build and operate our infrastructure in a 24 by 7, highly reliable way. At the same time, the role is also to provide the capability across the enterprise to consolidate and bring together as much as possible on a consistent architectural basis that applies at any level. This means it has to be relevant as much at the desktop level as in the middle and mainframe tiers.

The overall Fidelity budget for information technology approaches US$2B a year. We support over 40,000 workstations and users linked to several large and capable data centers. From an equity trading perspective we process (on average) 150,000-200,000 market trades each day. Fidelity alone accounts for 10-15% of NASDAQ's trading volume.

But even more impressive to me is that those 150K-200K daily trades turn into 10-20 times as many 'system transactions' that are going through CICS, DB2 and other middleware. Note also that these trades by no means encompass all of our activities.

As these statistics should indicate, we are a three tier shop. We run a large mainframe population, with CICS and DB2, for the core transaction processing capabilities. We also possess a substantial middle tier, open systems population (both RISC- and Intel-based) for our mid-tier servers.

In order to deliver architectural consistency, we have — over the past two and a half years — invested heavily in implementing XML across all our tiers. One objective of this implementation was to establish a consistent set of definitions which would, in turn, facilitate how applications would work with each other. A second objective was to focus on how to make our Enterprise both more efficient and able to exploit the merits of federation.

The first challenges: an architecture and XML

When I joined Fidelity, from IBM, some four years ago, it was not unlike being a poacher turned gamekeeper. I went from running a major slice of IBM, selling products to customers, to being an IBM customer. While this was quite a shock, it has been a fun and interesting experience.

When I started I joined the development organization. We had, and have, a large contingent of programmers working in the Central IT organization. For example, the Systems Company alone has 750 programmers who are charged with providing:

- **core infrastructure products across the organization — for example market data (stock quote) provisioning, work flow engines and CRM implementations**
- **software services — for example, providing programming and integration skills 'for hire' to our business units.**

From a technical perspective my initial challenge lay in addressing the plethora of different solution implementations deployed on different platforms. There was no consistent architecture. The first action the team took was to identify such an architecture — something that could unify the organization.

As we all know, architectures look simple but are hard in practice. We based ours on the three tiers, albeit with rationalizations.

The top tier, based around the mainframes, was already in good shape. Most large systems shops have the experience and tools to define what and how they are doing. Fidelity was no different.

In the middle tier, there was less — or no — coherence. Divergence seemed almost to be the rule. Little was consistent. The application capability, or even system to system connection capability, of the middle tier was limited.

Our first step focused on adopting XML. This proved to be a lightening rod, albeit one which challenged our developers. But, by using XML, we saw that we could build a common inter-connect capability across all the various business units. The result would be data definitions that were consistent for the definition of DTDs. Although the implementation of XML was born out of the need to rationalize our middle tier, we applied it across all tiers — and data and applications.

The first result was that we were able to make savings measured in hundreds of servers. A second result involved application server middleware. We have a substantial

investment in these, either already rolled out or being rolled out. Looking back, it is now quite clear that the investment we undertook to introduce XML-based data definitions across the business made exploitation of application server middleware much easier than would have previously been possible.

Other challenges

Another major challenge involved the interaction between my unit and other parts of the organization. There was a (past) history of misunderstanding and lack of mutual sympathy.

When an executive team works to make the organization function in a coherent and consistent manner, such friction matters. Looking back, we initially had a serious management issue to face as we tried to sort out, and then change, the way in which the central Infrastructure group worked with the business units.

The underlying issue was, as is common in successful and entrepreneurial organizations, that individual business units were accustomed to 'doing their own thing'. This included selecting, developing and implementing their own information technology solutions.

Unfortunately, satisfying the business unit picture is not necessarily a good recipe for the overall corporate good. We were, to all intents and purposes, wasting money that could have been deployed to better use elsewhere. This needed to be tackled.

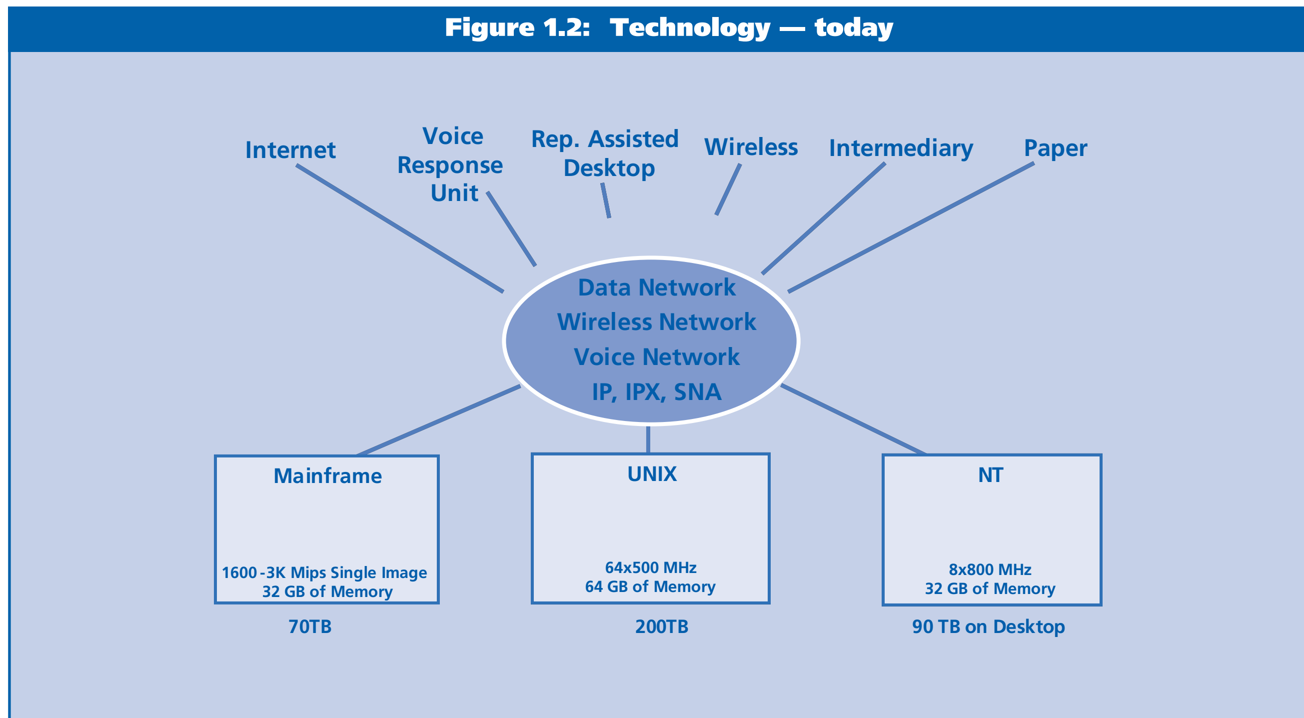
What really assisted in resolving these conflicts was the work we had done to create the architecture around XML. Initially we had hoped that the 'simple' act of establishing an architecture would be sufficient to meld people together.

As a former software architect in my time at IBM, albeit not a particularly good one, I had had the good fortune to work with some great architects. I presumed that, with the similar quality of people in Fidelity, we could create a 'self-compelling' architecture. I thought that such an architecture's sheer clarity and elegance would attract the necessary support.

I was overly optimistic. What I learned was that we needed to enable people to 'volunteer to come to the water'. Then we needed them to convince themselves that, in their opinion, the architecture made sense.

To achieve this, we had to engage the key people in the process, rather than issuing any diktat from the central organization. When we started we kind of assumed we could tell the business units what the architecture and data definitions would be. We ended up enlisting the business units into developing the design.

Figure 1.2: Technology — today



Although this took much longer — two years — than I had anticipated, the end result was superior. It was only when we had involved the business units, when they had become part of the architecture process, that:

- **acceptance occurred**
- **dissemination happened**
- **success materialized.**

Today's challenges

Now that we have resolved the architecture and adopted XML for data definitions, I see two major current challenges (and these apply to most large financial institutions):

- **data proliferation**
- **cost of ownership.**

From a purely architectural perspective, the first of today's challenges is about our data architecture. The reality is that we have never been shown a database that we did not like for some reason or other. We seem to have a version of almost every database ever created in production. These all work, at a certain level: that is not the problem.

The problem is that we have 300TB+ of data in relational databases. This is not only increasing but it is proliferating. We will likely reach a petabyte (PB) of database storage within the next couple of years (Figure 1.3).

The concern is two-fold. There is the fiduciary dimension:

- **how much data do we actually retain and for how long?**
- **who might need access (customers, regulatory agencies, etc.)?**

This is a constant concern. To address some of these issues we are installing Storage Area Networks and are looking at open systems hierarchical storage management.

But an even greater issue is the data itself. It is not information (as yet). We are, therefore, working with several vendors to try to determine ways in which we can understand:

- **what data is in our databases**
- **how this might be turned into information.**

We also want to:

- **discover how much data and information we have — plus how many times we replicate it, copy it, access it, ...**
- **identify where this data exists (and where it does not)**
- **determine how we might move from a traditional data repository to a viable store of information.**

One clear objective in this work is to help us better understand our customers. This comes back to the core issues:

- **where is that data?**
- **how do you turn that data into information?**

We really do want to understand our customers. If we can do this we will be able to deal with them on a basis that each one feels is personal.

If turning data into information is my number one problem today, my second one revolves around total cost of ownership.

It a continual source of amazement (and horror) that, after 15+ years of being sold 'solutions' which supposedly address the cost of desktops and servers, we still do not have accurate insights into what the true costs are. Without this information it is near impossible to manage costs, never mind reduce them.

Increasingly, therefore, Fidelity is focusing on the cost per desktop and per server. This is an activity that is being actively encouraged by our executive management team. Consequently we are for ever seeking ways by which we can introduce more automation.

Nevertheless, there does not yet appear to be a vendor that has an answer that really resolves this issue.The essential challenge is that resolving our problem is not in the interests of the vendors. Clearly, they need to drive their own profits. What has happened, and continues to happen, is that they spend more time figuring out how to increase their revenue streams through clever product and service mechanisms than attacking the heart of their customers' problems.

Let me give you a couple of examples. In at least one vendor product, the complexity is beyond belief. It is so great that it is essentially unusable. Only if you add in humongous (and hugely costly) services can you install the product and make it (somewhat) operational. That is at the high end of the systems management offerings.

At the other end of the scale, the problems are different but just as punishing. Take a major desktop product supplier: consider the complexity that multiple packages and

release iterations generate. With over 40,000 desktops in Fidelity, changes provided by any vendor become critical.

While key providers do offer a semi-automated service — like obtaining critical updates over the Internet down onto the desktops (or servers) to keep major operating systems up to date, not least for security — even this is insufficient.

All too often each time one of these critical updates is downloaded, that system has to reboot. Imagine the cost of taking down and then rebooting 40,000 desktops — even with some level of automation. Now add the cost of rebooting several hundred servers, and making their applications unavailable for a time.

That this cannot be wholly automated is an expensive nonsense for all sizes of business, not just Fidelity. What we know is that we spend millions of dollars each year just to find out what we have on the desktop and to keep the levels of software current. Furthermore, this expenditure does not include that spent on the applications and middleware on those desktops, and how these might be managed.

The net position is that, currently, there is really no straightforward way to obtain the necessary control which does not interfere with business operations. I am in the middle of trying to standardize my desktops and provide the appropriate security. We deliver, but the cost is much greater than most vendors dare admit.

Future opportunities

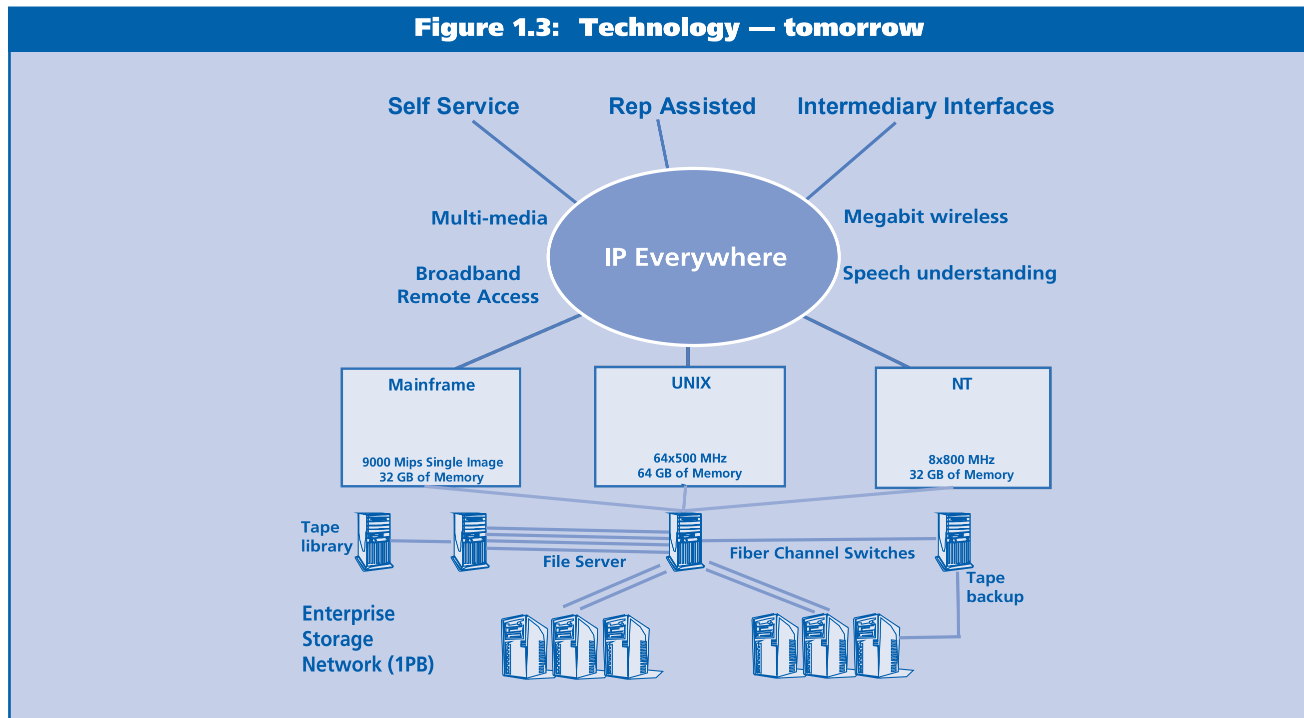
From my perspective, today's challenges — the transition from data to information and obtaining control of the cost of ownership of IT — are unlikely to be solved soon. They are challenges today. They will continue to be in the future.

In addition, I expect the Next Generation Internet to offer some great opportunities for us to support both internal and external customers. Fidelity is already taking advantage of the Web in a major way. As I described earlier, one of our key entry points is the Web — for customers to access our products and services. It is universal, at least by the standards of the IT sector.

In two or three years, however, the Internet will have matured. It will have become a multi-faceted delivery tool. While it is a tool today, it will be even more important in the future. We will have to learn how to operate in this new Web environment, one that will offer much more than just browsing. To me it will involve much greater inter-business (or inter-person) application deployment.

A key aspect of this Next Generation Internet may well be the emergence of true personal computing devices that will be ubiquitous and portable. We already support devices such as the RIM Blackberry, the Palm and the PocketPC. I expect huge progress in this area, as Tablets become richer and lighter and are combined with UMTS (or G3) broadband connections. Add in R/F transmitters which,

Figure 1.3: Technology — tomorrow



when you walk up to a machine, will know who you are and combine these with ever improving biometric capabilities and your Internet expectations are going to be deeper and greater and more secure.

Furthermore, capitalization on television as a media will occur as the Web matures. Within Fidelity, we offer 'Fidelity TV' for broadcast internally over our Intranet (IPTV). Use of this is increasing monthly, as our management sees the value of direct communication with employees.

Today, Fidelity TV is used mainly to bring the Company together. For example, we recently held a set of seminars over a four week period that attracted 2,000 people watching every time we broadcast. These seminars allowed us to share new technology perspectives with the business units. Obvious extensions of this include:

- **delivering 'learning on demand'**
- **improving desktop conferencing.**

The next logical step (as bandwidth and cost improve) is to make such capabilities available to our customers. We envision a 'video portal' with interactive services that will make the Web browser access of today look feeble. This is going to be an incredibly interesting area, as we seek to make it much easier for our customers to work with us.

Linux is another challenge — and an opportunity to consolidate further our middle tier. As I said before, we have accomplished much simplification of the middle tier, albeit on fewer middle tier boxes. Linux on large servers as well as on the mainframe has the potential to offer us:

- **the option to consolidate multiple middle tier boxes onto fewer, larger boxes, not least by exploiting the ability to run multiple Linux partitions simultaneously**
- **mainframe quality reliability**
- **system management tools (particularly those not found on traditional middle tier systems)**
- **inherent maturity**
- **the ability to work with traditional mainframe application and middleware environments.**

The reason that Linux is so attractive to us is that we have sufficient systems expertise to handle the very real complexities of Linux. Linux really only makes sense when you have 100s or 1000s of Linux images, but run only a tenth as many physical machines.

Another aspect of the move to Linux is the wealth of Linux talent in the skill pool. We are already seeing college kids who are Linux trained. They are not AIX or Solaris or even Windows knowledgeable. But they know Linux.

This is not to suggest that Linux is free. Support personnel and maintenance of any Open Source product adds up. But Linux still offers us the opportunity to change our software cost model. Linux consolidation of the middle tier is looking ever more attractive.

Middleware

Middleware has truly matured. Consider middleware from a transactional processing perspective. At last, most middleware is starting to approach the level of maturity that CICS (for example) has had for a decade or more. Even though CICS is 25+ years old, it remains an awesome product. Our ability to support our customers in the way they want depends on it.

By comparison, application servers — such as BEA's WebLogic, IBM's WebSphere or Microsoft's .NET — have yet to deliver the solidity and reliability of older middleware like CICS or even MQSeries. But there is clear evidence that they are getting there.

What we see is a middleware market that is being obliged to become much more mature. Vendors now recognize that significant tools must be put in place to make middleware easier to exploit. The ability to decide what you want to locate where and how to model each function is improving. Fitting middleware into a three tier architecture is also definitely improving.

Finally, and this is of major importance to us, most mainstream middleware is becoming much more reliable. The reason is, in my view, that middleware vendors have (at last) recognized that connecting applications is where, in our business, we must have results. As the vendors design their new versions to be more reliable, the level of control and attention required to run the middle tier has diminished. This is good.

An example of this is Microsoft's .NET which is now seen as critical in the enterprise environment. Microsoft knows that .NET has to be robust if it is to make the transition which will allow us to use its platforms for heavy duty applications.

Not only is middleware's story improving but there is consolidation. The middleware vendor shakeout continues but, most importantly, consolidation of the middle tier is increasing as vendors deliver what their enterprise customers have requested for many years.

Here is another perspective. There has always been a granularity issue about systems — should I buy:

- **another mainframe and move function on to it, (even though mainframes are expensive to buy)**
- **or multiple UNIX boxes at US$10K (or US$15K or US$20K) which each business unit can locate where it is needed.**

What has happened in the past was that many decisions, because of the perceived granularity of UNIX boxes, were made at the business unit level. But this only increased operational problems — there were too many uni-purpose boxes in too many places.

Individually these are difficult, and costly, to manage. They are even harder and more expensive (to manage) as the numbers exceed thousands. They, the vendors, have (albeit unwittingly) created a problem.

Why does middleware matter here? If the middleware vendors do not solve the problem, they almost oblige us (and others) to pursue the mainframe and Linux routes. (Ironically, we see those same vendors having the same problems in their own processing shops. They themselves need to consolidate.) We are, therefore, putting pressure on our vendors to deliver middleware to enable multiple mid and bottom tier systems to be run as combined units.

Let me make the point another way. Think about a system supporting some form of financial trading. In our business that means availability is paramount. What starts as a $20K box uni-processor becomes a two-way (or four way) device and the price doubles or quadruples. Add a disaster recovery site and the cost can double again ...

All this adds to the expense of an operations budget. The choice seems to be between continued and increasing granular device expense, include complex systems management or consolidation on very large servers. We are seriously looking to obtain from this entire area (open systems, middleware and distributed computing) what we already have on a mainframe — 24 by 7, redundancy, backup, etc.

So, in a sense, what I am saying is that, in the middleware arena, the pressure is on vendors to get their act together. If they do not they will be supplanted by Linux-type consolidation. This applies irrespective of whether it is operating systems vendors or pure middleware vendors or a combination of both. We require our various systems and applications to be able to run together as a coherent whole rather than intruding as large numbers of point to point connections. That is where I see enterprise middleware playing its part in Fidelity's future.

Lessons learned

The number one (lesson learned) is, speaking as a CIO, that you must have a strong technical vision with an architecture upon which you can fall back. That said, the development of that architecture is not a central responsibility; you must engage the entire organization if you wish an architecture to be accepted. That is what we did (eventually) with XML.

My second lesson is — expect vendors to oversell everything. This seems to be inescapable. While certain vendors are better than others — they support you better and deliver in the end — I can look back at most weeks on at least one initial sales call where a vendor approaches and promises the world. My favorite (or least favorite) is: 'we'll cut your total cost of ownership by a third within a month'. Beware.

The third lesson I have learned is that unless you have a perfectly centralized, 100% controlled organization (and I have yet to encounter one), the most important dimension to my role as CIO is constantly to interact with, and build bridges to, my internal customers, the business units. Fidelity is not unique in this. I see this need in every financial institution with which I talk. Establishing personal and technical relationships with the business units is a major driver. If I can work with them — and they with me — to build that trust, the enterprise truly benefits.

Management conclusion

Modern middleware — for example, application servers and related technologies — is maturing. It is this coming of age that is the good news from Fidelity for it is needed in environments where multiple applications need to be linked together.

That said, once you have a coherent architecture and a sound operational base and standards, the role of middleware as an integration engine diminishes. This means that middleware faces some clear challenges. As Mr. Haile illustrates, size and complexity are becoming an ever increasing burden — from data (or information) through cost of ownership and the ability to co-ordinate computing resources.

Ever increasing standardization can reduce an enterprise's complexity. But it may, in turn, produce a long term decrease in the importance and role of middleware — as a broker of application integration.

Free middleware: the surprising facts — Part II

Tom Welsh
Consultant

Management Introduction

Writing middleware is difficult, time-consuming and requires considerable expertise as well as resources for development and testing. It seems illogical to give the results away.

Yet there is much middleware available which is free. Furthermore, the volume and scope is continually increasing. In the May, 2002 ***MIDDLEWARESPECTRA****, Tom Welsh looked into free CORBA software packages. Now he widens the focus to other free middleware including:*

- ***operating systems (Linux and associated elements)***
- ***Web infrastructure (Apache, Tomcat, Xerces, Mozilla, XUL)***
- ***application servers (JBoss, Enhydra, JOnAS, ExoLab, Zope)***
- ***relational databases (MySQL, PostgreSQL),***
- ***development tools (Perl, PHP, Python)***
- ***IDEs (NetBeans, Eclipse, languages)***
- ***messaging (JMS).***

All rights reserved; reproduction prohibited without prior written permission of the Publisher.
© 2002 Spectrum Reports Limited

Operating Systems

No overview of Open Source programming would be complete without a mention of Linux, the Open Source operating system. Strictly, perhaps, it should be called GNU/Linux, as Linux distributions include a lot of code from the GNU project initiated by Richard Stallman in 1983. According to Stallman, one Linux vendor found that the Linux kernel itself made up about 3% of its distribution, while GNU components accounted for no less than 28%.

A study carried out by David Wheeler in 2001 showed that the Red Hat Linux 7.1 distribution contained over 30 million lines of Open Source code (most of it C and C++). This was valued at over $1B — had it been developed by normal proprietary means. As well as the operating system itself, the distribution contained the Mozilla Web browser, X-Window system, MySQL database management system and many other independent components.

Although most graphical user interface work undertaken on Linux depends on X-Windows (as it does for UNIX), Motif was largely out of bounds to the Open Source community until recently. In May 2000, however, The Open Group released Motif under a public licence — albeit one which stopped somewhat short of making the software 'Open Source'. Nevertheless, developers are now free to use Motif on Linux — and they can read the source code, with the caveat that any resulting applications may only be deployed on an Open Source operating system.

In addition, there are several ongoing projects that aim to give Linux a GUI as good, or better than, Windows. The front-runner is KDE — the K Desktop Environment (where the 'K' stands for nothing) — initiated by Matthias Ettrich in 1996. Written in C++ throughout, KDE is layered on X-Windows and CORBA.

In the meantime, however, KDE had acquired a rival in the shape of GNOME, the GNU Network Object Model Environment. Started in 1997 by Miguel de Icaza, a system administrator at a university in Mexico City, its original aim was to provide a true Open Source GUI desktop for Linux and UNIX in general. By the time KDE had cleared its name, the GNOME project was too far along for it to be disbanded.

How does GNOME differ from KDE? Apart from its distinctive look-and feel, it uses the GTK++ widget set — which it considers superior to Qt, on which KDE is based. Then there is the matter of programming languages. One of the attractions of GNOME is that it is a multi-language environment, enabling contributors to write their modules in anything from C to Lisp or Scheme.

Open Source projects need a good version control system, as they are the results of collaboration between people who may live in different parts of the world. The standard in this area is Concurrent Version System (CVS) from Cyclic Software, which is provided free with source code.

Web infrastructure

Many Web sites are powered by a combination of software known as LAMP, which stands for Linux, Apache, MySQL and Perl (or PHP or Python). These are the essential components that go to make up a de-facto standard Open Source Web server.

Probably the most popular HTTP server in the world, Apache drives no fewer than two thirds of the Web sites on the Internet. The developers' mission is still delightfully straightforward — "*to provide a secure, efficient and extensible server which provides HTTP services in sync with the current HTTP standards*." Apache has always had a reputation for stability, and boasts a steadily lengthening list of optional code modules supporting everything from authentication and automatic URL correction to Perl and PHP.

One of the earliest and most important Apache extensions is the Tomcat Servlet and Java Server Pages (JSP) engine. The project has also undertaken responsibility for integrating Simple Object Access Protocol (SOAP) on Linux. Apache XML-SOAP (previously known as SOAP4J before IBM donated it to the Apache Group) is currently at Version 2.1, with 3.0 moving forwards. The main pre-requisites are:

- **Tomcat**
- **Apache's Xerces XML Parser.**

While Web sites powered by Apache can be accessed using any browser, Mozilla is by far the most widely used Open Source browser. The project began in 1998 when Netscape decided to donate the source code for its Navigator browser to the Open Source community.

When all is said and done, though, Mozilla is only a browser. So why should it be of special interest to software developers? Essentially because, being supplied in the form of modular source code, it is re-usable. The Gecko rendering engine, for example, is one of the most advanced in the world; it is far better than most development teams could hope to write for themselves.

The Extensible User Interface Language (XUL) allows programmers to create cross-platform user interfaces that work on Windows, UNIX, Linux or Apple's Macintosh. Using this, IBM has even ported Mozilla to OS/2.

Then there is XPCOM, a cross-platform version of Microsoft's Component Object Model (COM). COM is an extremely effective and powerful component framework whose greatest limitation is that it is available for Windows only. XPCOM largely removes that limitation.

Application servers

Should an HTTP server require some backup — particularly in the dynamic page serving department — there are a growing number of Open Source application servers for Linux. Among the best known are:

- **JBoss**
- **Enhydra**
- **ActiveState's Zope.**

But there are many others. Indeed, one of the most remarkable aspects of the Open Source world is its fecundity. Investigating one project usually reveals links to several more, and so on in an almost fractal explosion of variety.

JBoss is the most popular free, Open Source Java 2 Enterprise Edition (J2EE) application server (Figure 2.1). It is supplied by the JBoss Group, based in Atlanta, which provides added-value consulting and feature development services. In April JBoss won the JavaWorld editors' choice award as 'Best Java Application Server' — ahead of BEA Systems' WebLogic Server and IBM's WebSphere, the commercial market leaders. "*JBoss stopped being an application server a long time ago*" remarked one of the judges: "*it is now officially a phenomenon*."

While this may be an exaggeration, JBoss is currently said to be downloaded over 150,000 times a month and clearly has a significant following. Enthusiastic users praise its stability, with servers running for months without any sign of failure.

Moreover JBoss is also close to the leading edge of standards — 'the full J2EE stack and beyond' as the organization's founder Marc Fleury puts it. Not only does it support the new Java Management eXtension (JMX) specification, but JBoss 3.0 is actually built around it. Exceptionally modular, JBoss can be used in conjunction with CORBA, Tomcat, Gemstone's Facets distributed cache and many other Open Source and commercial packages.

Another popular Open Source application server, which claims to have been the first, is Enhydra. After several years' development by Lutris Technologies, the basic application server was made Open Source in January 1999 — shortly before JBoss became available. Confusingly, the free version is called simply Enhydra, while Lutris sells a commercial product in two editions:

- **Lutris Enhydra**
- **Lutris Enhydra Application Server.**

Things looked black for the Enhydra.org Open Source product when Lutris dropped its support in April. However, the ObjectWeb Consortium stepped in smartly to take over sponsorship. Enhydra.org is now being hosted by France Telecom R&D, a founder member of ObjectWeb.

Enhydra itself is a servlet framework, but it can acquire Enterprise JavaBeans (EJB) and CORBA capability through an integration with Evidian's JOnAS and Jonathan respectively. Perhaps its most exciting feature is XML Compiler (XMLC), a technique for serving dynamic HTML pages using XML and Java — which some consider superior to Java Server Pages (JSP). Although XMLC is not a standard, it is said to be extremely productive and can be used with other application servers such as BEA's WebLogic.

The ObjectWeb Consortium, which has taken Enhydra under its wing, is an Open Source community set up in 1999 by France Telecom R&D, Bull/Evidian and INRIA, the French National Institute For Research In Computer Science and Control. It hosts a number of projects including:

- **JOnAS (Java Open Application Server), an Open Source EJB server**
- **Jonathan, an adaptable distributed object platform which supports several 'personalities', including CORBA and Java Remote Method Invocation (RMI)**
- **JORAM (Java Open Reliable Asynchronous Messaging), an Open Source implementation of the Java Message Service (JMS) specification**
- **JORM (Java Object Repository Mapping), an adaptable persistence service with personalities including EJB Container Managed Persistence (CMP), Java Data Objects (JDO) and Java Connector Architecture (JCA)**
- **JOTM (Java Open Transaction Manager), an implementation of the Java Transaction API (JTA) which works with a number of ORBs**
- **OpenCCM (Open CORBA Component Model platform), an Open Source implementation of the CORBA Component Model — similar to EJB but with support for languages other than Java**
- **RmiJDBC, a remote JDBC driver that allows access to a relational database through RMI**

- **ProActive, a Java library for parallel, distributed and concurrent computing, including mobility and security.**

Advocates of JOnAS, who can become quite competitive when arguing its advantages over those of JBoss, will no doubt be delighted that ObjectWeb now hosts Enhydra as well as this long list of predominantly French projects. It seems likely that further integration between these packages will take place sooner, rather than later.

Although less well known than JBoss, Enhydra and JOnAS, at least three other Open Source application servers deserve mention. The first of these comes from the ExoLab Group which is based in San Mateo, California and leads a number of projects including:

- **OpenEJB, an EJB server**
- **OpenJMS, a JMS implementation**
- **OpenORB, a CORBA ORB**
- **Castor, a framework for mapping between Java objects, XML documents, databases and Lightweight Directory Access Protocol (LDAP) directories**
- **Tyrex, an implementation of the Java Transaction Service (JTS).**

Similarly Caucho Technology of La Jolla, California, offers its Resin Server in two editions. The basic product supports JSP, Javascript and XML with a strong emphasis on performance. Resin-CMP extends the basic server with support for EJB Container Managed Persistence.

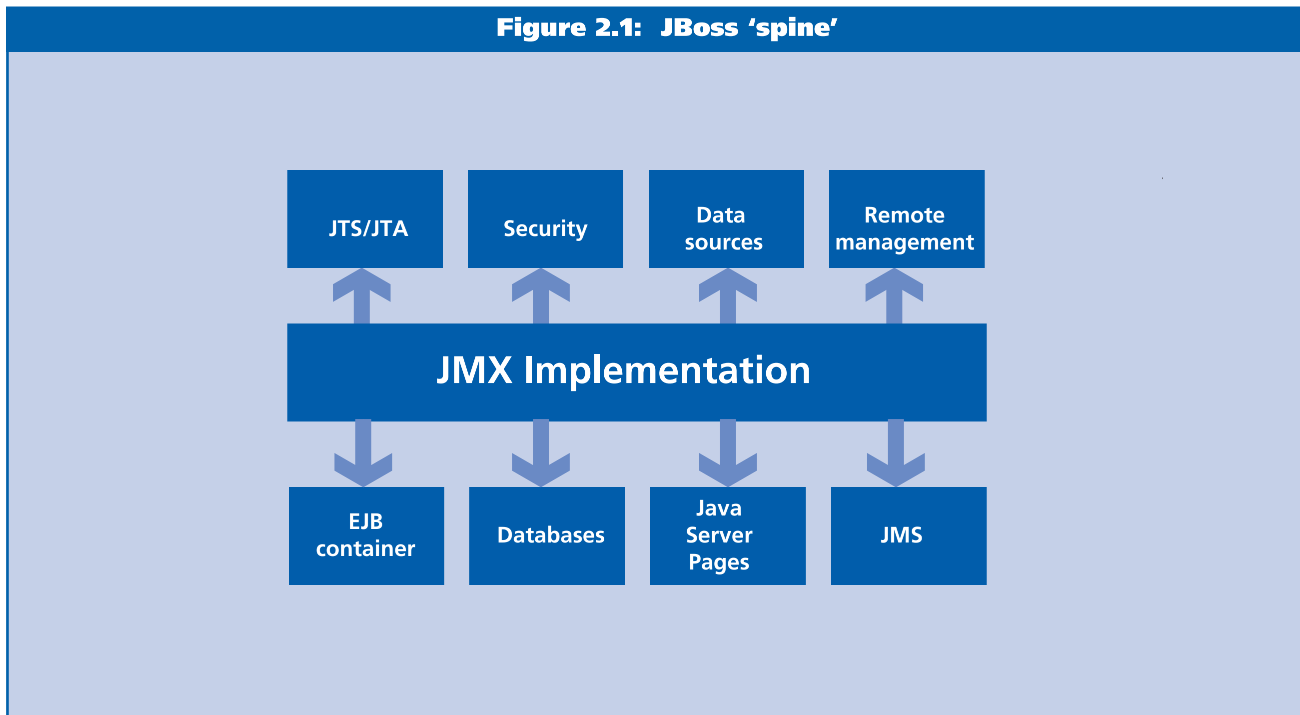
Demonstrating that not everyone is committed to Java, Zope Corporation's eponymous application server is based on Python, another object oriented language with an enthusiastic following. The core server is freely available for download, but there are a number of Open Source and commercial add-ons that enhance it with extra features, such as content management.

Database Management Systems

Production-quality database support is one of the features that traditionally commands a significant price premium in commercial software. So it is not surprising that licensing issues need to be carefully examined when choosing an Open Source database management system. Having said that, there is plenty of choice, with as many as 150 packages available that are described (loosely in some cases) as 'free databases'.

Modern databases are so complex and feature-rich, though, that there is a distinct reluctance among corporate users to adopt them. It should also be kept in mind that most free databases lack features that would be on the essential check-list of any leading vendor — for instance — functions like:

Figure 2.1: JBoss 'spine'



- **transactions**
- **triggers**
- **stored procedures.**

MySQL and PostgreSQL are among the most popular Open Source relational database management systems (RDBMS), and can be distributed without runtime charges. Given the importance of RDBMS in modern programming and the number of Open Source Linux projects in progress, it is not surprising that many applications and Web sites make use of them.

MySQL received the ultimate accolade in 2001, when Oracle announced a migration kit for users wishing to move their applications from MySQL to its own RDBMS. It is developed, supported and marketed by the Swedish company MySQL AB, and is offered free of charge under the GNU General Public License (GPL). This, however, forbids inclusion of the code in any non-free product. A commercial license is available for those who do not wish to be bound by the GPL.

This is fair enough. MySQL is free when used in free software, but costs money when used in software that is to be sold. With over 2 million installations worldwide, it is certainly the most populous Open Source database and it has a reputation for good performance.

Strictly speaking PostgreSQL is:

- **an object relational DBMS (ORDBMS)**
- **an example of the healthy relationship that has evolved between universities and the Open Source community.**

It is an evolutionary successor of Ingres, the pure RDBMS which was developed at Stanford University before entering the commercial arena as a leading player in the 1980s. (The name Ingres, by the way, refers to the French painter — in line with a whimsical custom in the database field). Researchers at Stanford followed up with Postgres, which had SQL capability added to it shortly before it went Open Source in 1996 — hence the name, PostgreSQL.

Beyond the 'big two', any choice of free databases is bound to be arbitrary. The third most important has traditionally been mSQL (alias Mini SQL), which is available from Hughes Technologies in Australia. As well as having less functionality than MySQL, it cannot be used without a paid-for licence (although the source code is freely available).

Other interesting or popular free databases include:

- **Firebird: this is an Open Source version of Borland's Interbase, which, because it is based on a successful commercial product, makes Firebird one of the most complete and reliable free databases**
- **Hypersonic SQL (HSQ); this is a small, lightweight database written in Java which can be linked directly into applications and accessed with JDBC**
- **SQLite; this is a C-based subset of SQL that provides many basic database operations**
- **Berkeley DB; this is a programmatic toolkit for embedded systems from Sleepycat Software; it is optimized for compactness, performance and reliability**
- **GNU SQL Server; this was developed as part of the GNU project, with help from the Russian Academy of Sciences (ISPRAS) and the Russian Foundation for Basic Research, and is a free portable multi-user relational database which supports SQL89 and some of SQL92 and is designed to work with C on UNIX or Linux.**

There are multiple other Open Source or free databases — such as University Ingres, ISAM Database Manager, TimeDB and MDBMS. The list is long, as any search of the Web will reveal.

Development tools

The final letter in LAMP stands for Perl or PHP or Python — three languages that are much less familiar than, say, Java or VisualBasic. But the Open Source community now possesses a multitude of programming languages and tools:

- **C is the default language of Linux, and C compilers and tools come with all distributions**
- **for the more ambitious, or object oriented, C++ is available**
- **other languages include Eiffel, Smalltalk and Java, as well as Tcl/Tk, Perl, Python, JPython and Object REXX.**

Perl may be the programming language that most faithfully reflects the spirit of Linux. To paraphrase its author Larry Wall, Perl is a mess because it is a tool for coping with reality — which is also a mess. In one celebrated passage, he writes "*Of course, in Perl culture, almost nothing is prohibited. My feeling is that the rest of the world already has plenty of perfectly good prohibitions, so why invent more?*"

To start with, Perl was regarded as an excellent language

for text processing — which helps to explain why it became so popular among Web masters. It was just so much easier to do everyday, practical things with Perl than with C, C++ or Java. For this reason, it has been dubbed 'the Swiss Army chainsaw of Linux' and there is a huge and steadily expanding library of re-usable modules at the Comprehensive Perl Archive Network (CPAN).

Since its creation around 1990 by Guido van Rossum, Python has gained a substantial following due to its unusual combination of power and simplicity. Interpreted, portable and interactive, it somewhat resembles Java and can run on a standard Java Virtual Machine (JVM). Unlike Java, though, it is also highly suitable for impromptu scripting — a role fulfilled in the Java world by Javascript (now standardized as ECMAscript).

In the same recursive style introduced by GNU, PHP stands for 'PHP: Hypertext Preprocessor'. It was created by Rasmus Lerdorf as a way of tracking everyone who read his online resume. Today, PHP is a powerful tool for serving dynamic HTML pages, very much like the Common Gateway Interface (CGI), Active Server Pages (ASP) or Java Server Pages (JSP). It can be installed:

- **either as a standalone CGI application, which works with any HTTP server**
- **or as an Apache module; the latter is more efficient as a new process does not have to be created for each invocation.**

IDEs

In contrast to the proliferation of language compilers and interpreters, the Open Source community has 'managed' for years to do without free integrated development environments (IDEs). Many programmers took the view that such luxuries were only needed by coddled, relatively unskilled corporate developers. Besides, IDEs are expensive to create and maintain, diverting resources that could be applied to better use.

This state of affairs began to change when big commercial vendors decided to invest in Open Source software. Sun, which had always been at pains to avoid competing with its Java licencees, decided in 1999 that it had to enter the development tools market — if only to prevent developers from going over to Microsoft's attractive VisualStudio. Accordingly, it took over NetBeans, a small Czech company whose eponymous IDE generated code for the Java Foundation Classes (JFC).

Two and a half years later, NetBeans has evolved into a Java framework for building and assembling development tools. Dozens of companies have used NetBeans in their own products, including such leading players as BEA, Compuware, Embarcadero, Iona and Thought. Sun, too, 'eats its own dogfood' by basing its Forte for Java toolset entirely on NetBeans.

The NetBeans framework offers considerable practical advantages — not least that products built using it readily interoperate with each other. Principled Open Source advocates, however, have some concerns about the licensing model and the dominant position held by Sun. Attractive as they may be, projects like NetBeans are a long way from Richard Stallman's original vision of a parallel world of free, unencumbered software that would be part of every citizen's heritage.

In any case, Sun was not left to enjoy the success of NetBeans for long. At the end of 2001 IBM announced a rival Open Source application development community known as Eclipse. Over 150 tools vendors signed up to participate right at the outset, including:

- **Bowstreet**
- **Computer Associates**
- **Embarcadero**
- **Mercury Interactive**
- **Rational**
- **Telelogic**
- **TogetherSoft.**

The Eclipse framework, developed by IBM's Canadian subsidiary OTI, is broadly similar to NetBeans — but is, of course, incompatible with it. IBM has already used it as the basis for a new IDE, WebSphere Application Developer. There is also a free Eclipse IDE which vendors like Rational are already shipping with their own proprietary products.

Naturally Sun is unhappy at what it sees as IBM's treachery. From the user's point of view, the consequences are rather more mixed. Certainly it is unfortunate that there are now two rival Open Source Java IDE frameworks, with the resulting dilution of support for either. On the other hand a measure of competition is healthy, keeping both groups on their toes.

Messaging

Until recently, messaging has not been a technology readily linked with Open Source. This is because there were no standards and the leading products were proprietary cash cows — like IBM's MQSeries or TIBCO Software's Rendezvous/TIB.

This is no longer true, but free messaging products are still quite a mixed bag, and often come as part of a larger package. For instance, any application server that complies with J2EE 1.3 or later is required to provide not just a Java Message Service (JMS) interface as previously, but a working implementation. This means that products like JBoss, JOnAS and Enhydra come with built-in JMS support. The ObjectWeb Consortium also supplies its Open Source JMS package, JORAM, separately from JOnAS.

Organizations that have already started work with a free CORBA object request broker, like TAO, may find that it includes an implementation of the CORBA Messaging Service, a relatively obscure part of the latest specifications.

Meanwhile, the spotlight has moved on to the latest fashion in messaging — Web Services, as exemplified by Simple Object Access Protocol (SOAP) and XML-RPC. There are many free Open Source packages in this space, most notably the Apache Group's XML-SOAP. Commercial vendors like IBM and Microsoft currently offer SOAP toolkits and the like free of charge, but these should still be discounted as they are usually alpha or beta software (at best) that may be withdrawn at any time if corporate strategy so dictates.

The impact of Open Source middleware

There are already many data points that show that Open Source software can save companies amounts of money that substantially improve their bottom line — without causing any adverse side-effects. Amazon.com, for instance, cut its technology costs by $17 million (25%) in the third quarter of 2001, largely through replacing UNIX servers with Linux. A spokesman gave a broad hint as to why so few of these success stories are reported in the media. "*We've always been pretty close-mouthed about technology on the back-end stuff, partly because it's pretty steep tuition and we don't want other people going to school on our tuition.*" In other words, the successful adoption of Linux was seen as a competitive coup.

In many cases cost is not the only — or even the primary — reason for using Open Source middleware. Organizations that employ skilled, experienced developers see advantage in having direct access to the source code. They can fix bugs, add or customize features and understand exactly what the software does (and why). Moreover, Open Source projects are often more responsive to user feedback than those from commercial vendors. They concentrate on the features that are in greatest demand, unhampered by marketing considerations.

Last but not least, Open Source middleware benefits from the continuous scrutiny of a global community of experts in which users are often developers and vice versa. Quality problems, bugs, security holes and even architectural shortcomings are quickly — and ruthlessly — identified.

The commercial implications (of so much free middleware)

At first glance, the revelation that so much free Open Source middleware already exists might seem to threaten the middleware industry's commercial foundations. This is unlikely to be the case, thanks to the huge (though often-overlooked) inertia of the IT market. Experience shows that new arrivals, no matter how much better, cheaper or easier, rarely succeed in displacing older technology altogether. Instead, a new equilibrium is usually reached after a while — just as the vaunted 'open systems' tide of PCs and UNIX servers failed to put an end to the mainframe era.

In addition, Open Source has its limitations, too. Indeed, on first inspection, it is even beginning to look as if it may actually be synergistic with commercial software enterprises rather than competitive. A surprisingly large number of Open Source developers work for commercial software vendors — often with the explicit approval of management (IBM is the most obvious example of this syndrome).

That said, some argue that the writing is on the wall for middleware as a commercial proposition. If functions as diverse as databases, IDEs and development tools as well as messaging and Web infrastructure utilities can be acquired for a modest maintenance burden, this is going to put pressure on traditional vendors. The most explicit example so far is what Sun has done with its Sun One Application Server (which used to be known as part of the iPlanet software suite). This was once a chargeable item. Today it is essentially free, if not exactly Open Source.

At a stroke, one commercial vendor raises questions about the pricing of IBM's WebSphere or BEA's WebLogic Applications Servers. The likely result is that the JBoss and other Open Source implementations will be the winners. And this could spread to all areas of middleware. In the instance of middleware it may only need one initiative to eliminate proprietary barriers.

Management conclusion

Of course, all is not sweetness and light in the Open Source world. All work must be funded in one way or another, and this inevitably leads to compromises. The ongoing ideological struggle between free software purists and pragmatists

rumbles on under the surface. Above all, the user must take responsibility for mastering a greater volume of technical detail. But perhaps that is not too great a price to pay for middleware that is free of charge, technically advanced, wholly transparent and continuously audited for quality.

The strength of Open Source software lies in its freedom to innovate without being constrained by marketing pressures, continuous audit and sheer force of numbers. These can complement the strengths of the commercial establishment. If these can co-exist, we will see a spectrum of Open Source middleware, from the conviction politics of Richard Stallman and the GNU project through to free versions of leading commercial products that are increasingly being made available for Linux and other environments.

An alternative, however, also exists. This would see middleware — from databases through development tools through IDEs, Web infrastructure, messaging and application servers — going the way of Open Source. While this might reduce originality, it might also speed acceptance, quality and commonality. Future years will be interesting in the middleware arena.

Of rules and middleware

Steve Ross-Talbot
Chief Scientist
Enigmatec Corporation
and
Said Tabet
Co-founder of RuleML

Management introduction

What role do rules play in middleware? It can be said that rules are just another middleware component. The positioning of standards such as JSR94 (Java Rule Engine API) and RuleML certainly encourages this view.

While rules have had a checkered past, there has never been more focus on rules. This applies to:

- ***support of greater personalization***
- ***business process management***
- ***Web Service composition***
- ***intelligent networking.***

In this analysis, Steve Ross-Talbot and Said Tabet examine rule technology and its role in middleware. They look at some of the standards, describe many uses of rules in a middleware context and provide examples that bring this topic to life.

All rights reserved; reproduction prohibited without prior written permission of the Publisher.
© 2002 Spectrum Reports Limited

Infrastructure, middleware and applications

Where does infrastructure end and applications begin? Interestingly enough this is one, quite good, definition of middleware.

Middleware occupies that large space that includes databases, web servers, messaging and lots of glue based frameworks. Middleware is the software that utilizes the infrastructure (the network and computers, if you will) to deliver an application. In a well-layered architecture you should be able to point to the infrastructure and you should be able to see the business logic.

As we move to greater intelligence the distinction between the layers becomes increasingly blurred. In the February 2002 **MIDDLEWARESPECTRA** there was an analysis entitled **Network computing and the middleware for the next generation**. This argued that:

- **network providers want to move up the value chain to offer more services, such as application hosting**
- **middleware vendors want to move up to provide better tools for building the business logic that we traditionally term 'the application'.**

The advent of Web Services, while full of marketing fizz (if not hype), is set to fuel growth in rule technology. The prediction — by Gartner, Hurwitz, Celent and Yankee, amongst others — is that business process management and its attendant business rule mechanisms represent one of the highest growth areas.

This makes sense as can be seen in the investment being made by companies such as BEA and TIBCO as they look to provide higher forms of business logic via rule technology. This applies also to the myriad of standards for Web Service composition — such as Xlang, BPML and WSFL.

With so much talk, is this a replay of Web Service-like overenthusiasm? Are rules hype? Or, can rules play a significant role in middleware in achieving the aims of just in time integration through just in time software?

What can rules do for me?

To understand what rules can do we must first understand what rules are. This may seem a trivial question but rules have a history that dates back to the days of Artificial Intelligence (AI) in which it was often said that rules-based solutions knew the value of everything but the cost of nothing.

Rules have a come a long way since the 1980s. Indeed advances in core technology and middleware have fueled a new generation of rules-based approaches. In our view, rules can be divided into three:

- **the deductive**
- **the constrained**
- **the reactive.**

The first is the classic deductive notion of a rule. Deductive rules have a high degree of relevance to any large data or knowledge base because they can be used to 'plug' the gaps in information by deducing (or deriving) conclusions from existing data and knowledge bases.

A simple example is a database containing customer buying records. A deductive rule might be used to show that customers are premium ones when they have spent more than $10,000 on luxury goods in the last month.

The definition of a premium customer is not in the knowledge base but is encoded as a rule. That rule iterates over the knowledge base and deduces those customers who fall into this category.

The second sort of rule is a constraint. Typically we find these in databases as triggers which are used to constrain values. These can include examples like:

- **'the age of an employee must be between 18 and 65'**
- **the dependence of relationships in which you state that 'all employees must belong to a department'.**

This is the classic notion of a trigger or referential constraint.

Reactive rules

The third type comes out of active database and distributed computing research and is called a reactive rule. A reactive rule is one that is event driven. It reacts to events and then does something.

This contrasts with the deductive based rules, which are:

- **primarily data-centric**
- **targeted at deducing facts from a large body of data.**

Reactive rules in active database research are a more generic form of trigger and are a superset of constraints.

Reactive rules are targeted at describing reactive behavior. A simple example embeds a reactive rule into a database to ensure that all employees belong to a department — a trigger. The events that cause such a rule to fire could be:

- **a delete event**
- **an update event**
- **a create event.**

When a department tuple is deleted the deletion event would be detected and would fire the rule. A condition is applied to the deleted tuple and, based on the outcome of that condition, an action fires to delete the associated employees (or perhaps to transfer them into a different department since deleting a department is not often associated with removing the employees). A more generic reactive rule in this context might be to inform an administrator, by calling out to some pager and blocking the delete.

Reactive rules are all about describing the behavior we want when change occurs. They can be used to:

- **deliver business transactions, which are nothing more than some specific set of changes**
- **monitor changes, and then perform an action to inform interested parties — a sort of event consolidation**
- **provide semantic routing in network elements.**

The advantage — of rule-based technology — in all of these cases lies in the code that you do not have to write. They complement SQL completely. Like SQL they state what you want, not how to do it. They are declarative.

Of course you can always write code or add more data to a system to do the same thing. But this:

- **costs more to do**
- **takes longer.**

The advantage of rules is that they can be used alongside the normal IT collateral to provide better business agility. This reduces the time and cost to deliver business flexibility.

Rules and middleware

In the area of rules there are really only two standards that seem to have significant value:

- **JSR94**
- **RuleML.**

The first (JSR94) comes from the Java stable. It is an interface to deductive rule engines.

However, the interface it provides does nothing to aid understanding of rules across heterogeneous platforms. On the other hand it does attempt to provide a plug-and-play interface between deductive rule engines and the entire J2EE platform.

RuleML is targeted at providing rule interoperability through interchange. In heterogeneous environments it is entirely probable that you will want to bolt a rule engine onto a Web server or application server or even a database.

The rules that describe what you want (to do) require more and more personalization. This in turn prompts the debate about rule interchange. It is not enough to provide plug-and-play if you cannot exchange rules.

For example you might be using IBM's WebSphere with iLog's JRules and BEA's Weblogic with HNC's Advisor:

- **JSR94 would allow you to plug the systems together but not exchange the rules**
- **RuleML would enable the exchange to happen and so provide the interoperability.**

The activity around JSR94 and especially RuleML is indicative of the primary role that rule technology is starting to play:

- **within the Java based middleware community (through both JSR94 and RuleML)**
- **beyond, with the wider non-Java based community (through RuleML).**

The primary target for personalization is through Web and application server integration using rule-based solutions. This is clear from JSR94.

The prominence of reactive rules within RuleML is that it possesses a clear role to play alongside — and as a complement to — messaging middleware, and not just message oriented middleware but also that being deployed to support the wider world of Web Services.

Web-izing business logic

While it is easy to wax lyrical about the theoretical benefits of rules, concrete examples offer a sounder base for optimism. With the arrival of Web object and distributed architectures there emerges a new challenge for smart business component deployment. The reason is that these could add

functionality to heterogeneous multi-organizational information systems. Flexibility is the key. RuleML, for example, can provide the standard representation of:

- **business rules**
- **policies**
- **regulations**

and facilitate their interchange across organizations and components of middleware systems.

In today's economy, corporations are under pressure to adapt quickly to new business needs and manage the growing complexity of their information systems. To prevent another collapse of the magnitude of Enron or prevent another Barings or Allied Irish Bank disaster, global accounting rules and real time compliance are needed.

A rules-based approach is well-suited to providing such a solution. Pre-trade compliance integrated with intelligent risk management and reactive rules is a solid example where business rules can take a middleware architecture to the next level — in effect making it the intelligent infrastructure. In such an instance, reactive rules are used to:

- **integrate co-operating components**
- **listen to relevant events**
- **feed data and information to a deductive rule engine where inference takes place.**

Derived data is used in complex rules for further inferencing, transformations and data processing. Figure 3.1 shows a compliance monitor screen where warnings and violations are recorded. In pre-trade and post-trade compliance, rules can be:

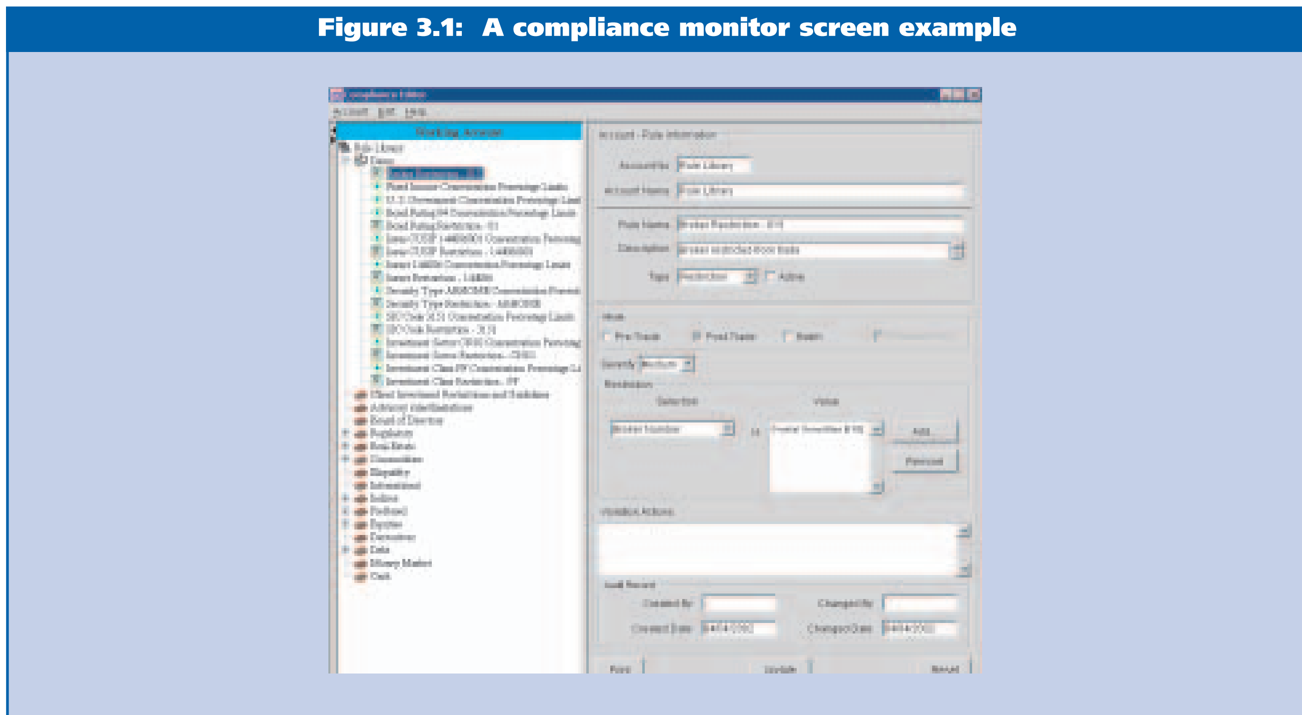
- **global**
- **regional**
- **local**
- **institutional**
- **or client-specific.**

Systems can be distributed across the globe. To ensure transparency, 100% visibility is a 'must have'. This is delivered in RuleML (but not JSR94) where London, New York, Chicago and Singapore can take advantage of the same rules, share them and re-use them as if they were local to each system. Equally, RuleML — because of its support for heterogeneity — will work with (say) HNC's Advisor/J in one location while another location might use iLog's JRules ... and so on.

Personalization and individualization represent added capabilities, where reactive rules combine with deductive rules. This can prove highly efficient. Equally, on-time multi-channel delivery of information from various components of middleware can:

- **not only be guaranteed**

Figure 3.1: A compliance monitor screen example



- **but also be augmented with sophisticated explanations and recommendations.**

Rules on the bus

The most interesting use of reactive rules (that we have seen so far) is to use them as agents that police higher-level business logic. A typical example might involve an agent, managing business transactions in a distributed environment.

In this situation you might wish to have a complex decoupled architecture in which the business is organized around front, middle and back offices connected by some message oriented middleware solution where:

- **the front office is responsible for booking trades**
- **the middle office provides services to ensure that the right price is struck, the necessary pre-trade compliance is followed and the counter party on whose behalf the trade is enacted has enough credit to complete the transaction**
- **the back office services ensures that the trade is recorded, reported and settled correctly and that all necessary documentation is sent to the new owner on a continuing basis.**

Investment banking has changed the fundamental architecture for addressing this. Solutions moved:

- **from the traditional monolith built on a request/response architecture which serialized all of the necessary steps to complete a trade**
- **to a highly distributed architecture, decoupled, which uses message oriented middleware to connect the various transactional islands (front, middle and back office).**

This has enabled trading in the front office (and middle and back office) each to work at its own pace. They no longer have to wait for unnecessary services (to them) to complete in the other offices.

Transactional islands in investment banking

The net gain has been:

- **an increase in trading throughput**
- **a decrease in error rates**
- **steady progress towards towards STP (straight through processing) and T+1 (next day settlement).**

But the decoupled nature of the new architecture requires the distinct handling of each business transaction. In the days when all of this functionality was monolithic, a database transaction was used to reflect the business transaction. This is what led to all the unnecessary services participating in the same database transaction — which also slowed everything down. Delivering this on a global basis causes further problems: try running a distributed 2-phase commit across a global operation.

In contrast, the management of business transactions in a world of transaction islands ends up becoming a perfect example of reactive behavior. As such it lends itself to being described declaratively. The desired behavior is to monitor events from the front, middle and back office. You are looking for all those events that make up the business transaction.

In a simple case this might be the correlation of new trade events with settlement events. A trade is only settled when all the settlement events that correspond to it have been received. The correspondence is expressed as the sum of the settlement events being the same as the amount in the original trade event. Of course, to do this properly, you need to have some temporal dimension in which to constrain the occurrence of the new trade event and its settlement events; this might be set at (say) 60 minutes, within which completion has to occur (Figure 3.2).

Figure 3.2: Including time

```
RULE: BusinessTransactionFlow
ON EVENT trade(?tradeId, ?tradeAmnt) FROM ?tradeSys
  FOLLOWEDBY settlement (?settlementId, ?settlementAmnt)
EVENTS FROM ?settlementSys
  WHERE ?settlementId EQUALS ?tradeId AND
    SUM (?settlementAmnt) IS LESS THAN ?tradeAmnt
ACTION
  SEND transactionComplete (?tradeId) EVENT TO ?tradeSys
ON TIMEOUT 60 minutes
  SEND transactionFailure (tradeId, ?tradeStatus) EVENT TO
  ?tradeSys
```

The source of the event is only relevant in so much as it may be used further to identify the event. So an event may arrive on a message bus or be delivered as a property change or even as an event arriving through some connector-based architecture from a legacy system. All that is important in the world of transactional islands is that the business meaning of some collection of events is correlated over time. It might be a business transaction or it might be used with a deductive rule engine to infer a collapse of some counter party.

```
BIND TransactionFlow.trade TO IBM:MQSeries
        ?QueueName ("Settlement.Trade.Queue")
BIND TransactionFlow.settlement TO ShinkaTech:WMS
        ?QueueName ("Settlement.Trade.Queue")
BIND TransactionFlow.transactionComplete TO TIBCO.RV
        ?TopicName ("Trade.Status") AND ShinkaTech:WMS ?QueueName("Log.Info")
BIND TransactionFlow.transactionFailure TO Sonic.SonicMQ
        ?TopicName ("Trade.Status") AND
        ShinkaTech:WMS ?QueueName("Log.Info")
```

Figure 3.3: Single binding of event names

A reactive rule, therefore, needs to be bound to event sources so that the rule can run. This can be achieved by a separate binding of the event names to specific sources (Figure 3.3). The use of reactive rules for managing a business transaction is only the start. They have relevance in the monitoring of any change represented by an event. The example shown in Figure 3.3 only deals with a business transaction. A reactive rule could:

- **be applied to the monitoring of risk over trading activity in which a correlation of trading activity is set against changes in risk**
- **watch the hardware platform upon which a business transaction runs, thereby tying the business intent to the delivery vehicle and so providing holistic management.**

The use of reactive rules to monitor and inform introduces higher semantic intervention into a distributed architecture. It enables real time sophisticated monitoring of the system that can be used to manage or police change as it relates to a business.

This is nothing new. The great and the good write code to do this.

What is new is the ability to do it and change it on the fly while systems are running without the need to change any code, and without restarting processes. This is the agility required for STP and T+1 in the financial sector. But it applies just as much to the huge range of applications and integration required in other industries.

Management conclusion

Given the described usage of rules above, it is clear that they need to work with current middleware components. They certainly need to work with Web servers, databases and message oriented middleware solutions. They also need to be made embeddable such that the rules can interact seamlessly with their surroundings.

The advent of the Java Rule Engine API (JSR94) in providing plug-in integration of deductive rule based solutions to the J2EE stack is evidence of the importance of rule-based agility to the range of middleware components applicable in the Java environment. The use of RuleML and the gathering pace of its acceptance suggests that plug-in integration is only a first step — and that rule exchange (where rules can execute across heterogeneous rule execution platforms) will soon be with us.

Furthermore, the support for deductive, reactive and constraint based rules in RuleML recognizes that interchange between rule types is just as important between rule systems. Indeed if Web Services are ever to achieve their ambitious goals, of true component re-use across the Internet, they will require such interchange to enable different components offering the same basic service to be used interchangeably.

In many ways, attaching agility through rules and applying standards — such as the Java Rule Engine API and RuleML — will lead to an explosion of Web Service component offerings by enabling a true market based approach for the services that they purport to offer. Indeed, a rule based approach will enable the consumers of such services to remain in control by being able to state their requirements as a set of derivation, reaction and constraint based rules which can be delivered to components as a statement of intent which they then promise to deliver.

Software — at your service

Dr Keith Jones
IBM Software Solutions Worldwide

Management introduction

There has been much speculation recently about the application of Web Services technology to key business problems. This raises a number of questions, amongst them:

- ***is this technology really mature enough for prime time deployment?***
- ***what are the risks of discovering security breaches?***
- ***can I be confident that Web Service transactions will be delivered once and only once?***

In this analysis, Keith Jones:

- ***takes a step back, to look at the origins and aspirations of a service-oriented approach to developing and delivering software***
- ***places Web Services technology into the context of an evolving scenario that is taking time to play out.***

At the same time, there is immediate value in the technologies now available from leading middleware vendors (including IBM, BEA, Microsoft and others) — particularly if applied to:

- ***otherwise costly integration problems***
- ***positioning of new systems for the future.***

All rights reserved; reproduction prohibited without prior written permission of the Publisher.
© 2002 Spectrum Reports Limited

Service-oriented software

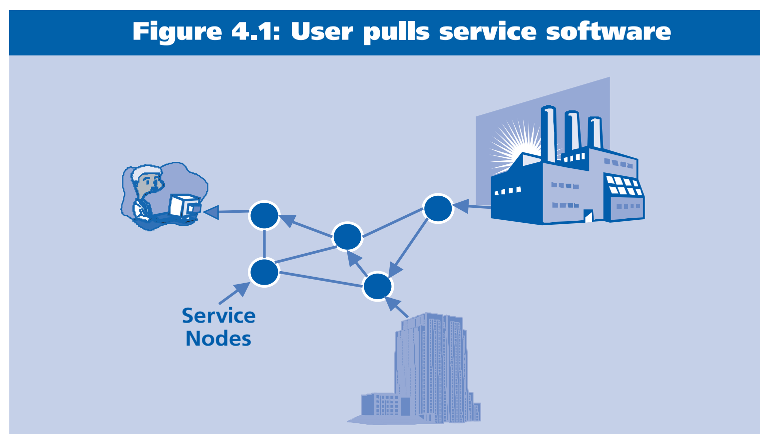
As the value of the Internet was becoming obvious to email and 'WWW' users in the mid-1990s, research workers in academia were asking whether the underlying Internet infrastructure might offer solutions to some of the most intractable software development and delivery issues. This was prompted by the realization that software development continues to be costly and constrained by the lack of highly skilled professionals, despite the best efforts by the IT industry in recent years.

Indeed, software being delivered today continues to be plagued by:

- **quality (fit for purpose) issues**
- **over-packaging (bloating)**
- **lack of configurability (excessive custom usage).**

One result of this thinking about the Internet, and software production, was that a radically different approach (to the way in which software is developed and delivered) was needed. The result is service-oriented software. Its primary attraction is that it can be invoked using the Internet's infrastructure.

Figure 4.1: User pulls service software



While the research continued, the most controversial aspect of this 'service-oriented' approach received a great deal of attention — so much so that it almost became a distraction from the underlying objectives. This controversial dimension involved the concept that each service invocation might become a chargeable 'item'.

In addition, that users might be able to 'pull' software dynamically in future (Figure 4.1) to meet their needs, using already available Internet connections, was considered to be a major shift away from existing approaches to software delivery. That software services might be considered highly re-usable as well as easily assembled — to satisfy functional needs — and equally easily be discarded after each use was also seen as a threat. Much the same was thought of the notion of removing the constraints of existing 'push' approaches to software development.

Nevertheless, progress was made — although a number of significant issues remain to be resolved before service-oriented software can become commonplace.

In broad terms, to attract both consumers and producers, software services must be identifiable, addressable and available at the time they are needed — wherever they may exist on the Internet. In other words:

- **consumers (of software services) must be able to find those services which meet their needs**
- **suppliers (of software services) must be able to identify consumers if they are to secure and charge for the service usage.**

Simultaneously, while research continued, various collaborative initiatives have taken up the challenge of extending what software services might mean. Their efforts have concentrated on the practical, particularly the definition of relevant standards.

At the same time, middleware vendors have spotted an opportunity. They have begun to deliver infrastructure products which they hope will support early service-oriented software.

In this context and given that an emerging technology base exists for Web Services, the issues today are:

- **what value can be achieved now?**
- **when will a full range of the desired functionality become available?**

Software services

A software service is characterized by the interactions it supports — not by the programming language or platform used for its implementation and deployment. An interface is, therefore, defined for each service.

This contains:

- **descriptions of the operations supported**
- **the messages required for input**
- **the messages expected on output or on fault conditions.**

Figure 4.2 summarizes these elements. A substantial community of middleware vendors and open source organizations has already agreed open standards for:

- **the interface description (WSDL)**
- **messages content for services (XML)**
- **the protocol to be used to transport service-oriented messages over the Internet (SOAP).**

Each of these standards is defined as being both language and platform independent in order to maximize the attraction to, and participation in, the sought after service-oriented economy. The adoption of XML, for example, for message data is widely considered to be a natural choice — even though some media forms (like video) do not fit well.

Although more work is needed, the standards work already undertaken has set a reasonable stage for software services to make their debut. If so, what issues remain to be resolved?

The open marketplace for software services

Software service interactions are between:

- **service consumers**
- **service providers.**

In addition, if interactions are between parties widely separated by distance, culture or time, there is a further need for a third role to be played — that of a service broker. Such a broker must independently:

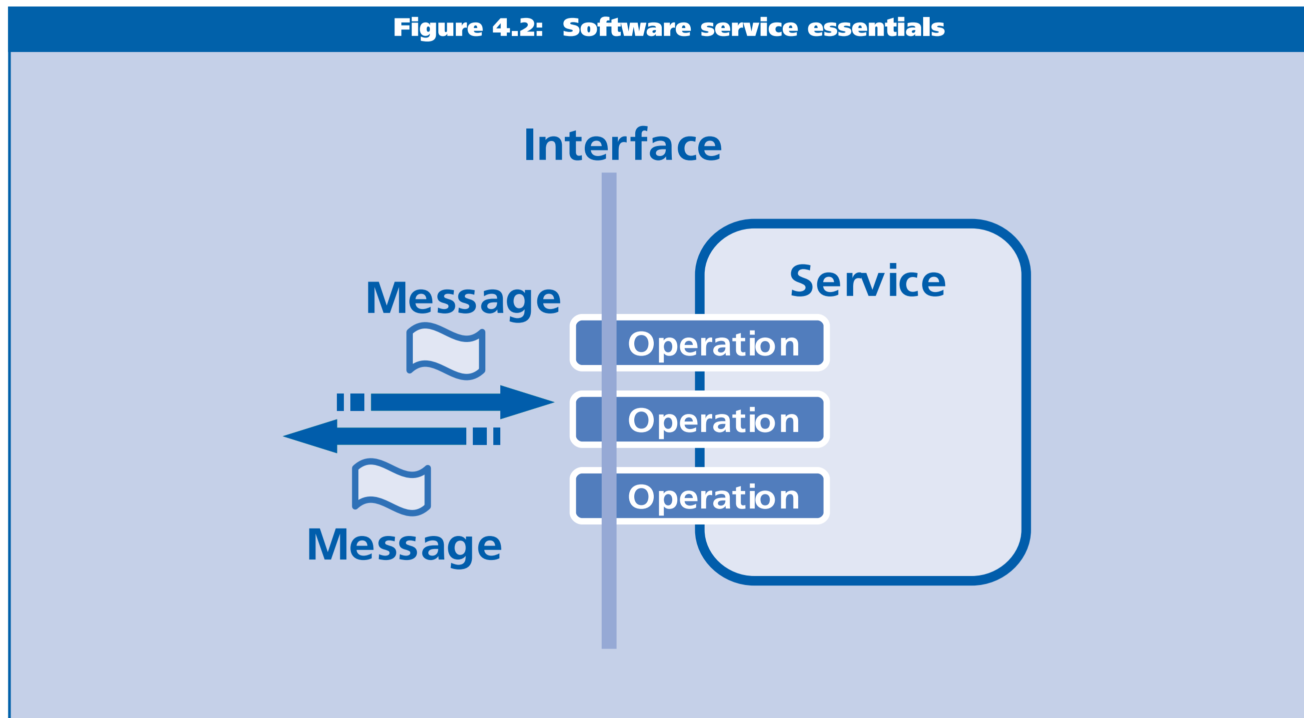
- **accept service descriptions from providers, when services are made available**
- **deliver service descriptions to consumers, when queries for service capabilities are initiated**
- **exploit standardized registration and query APIs (UDDI).**

When a need has been satisfied all references to services used must be discarded. Then the next cycle can restart.

For example, if the consumer — in an ideal scenario — was a garage employee ordering an automobile part, he or she might choose to use a proprietary service 'orderPart' to satisfy that requirement. The service provider (the garage enterprise) might implement an orderPart service by combining (composing):

- **several different services as offered by speciality parts suppliers**
- **specific cross-industry ones offered by local governments for (say) sales or other tax calculation.**

Figure 4.2: Software service essentials



It is the ability to combine different specialist services from different sources which makes Web Services so beguiling and attractive. In such an ideal world, the service consumer would:

- **first formulate his (or her) need for specific functions or data**
- **then contact his (or her) broker to acquire details of available services which satisfy that need**
- **choose from the available services**
- **compose the service request(s)**
- **start, and then complete, the usage process.**

This ideal scenario is one that cannot yet be achieved. While service brokers are already available on the Internet, they do not yet have details of many available services that can be accessed in this way. The choice remains limited.

What is needed is an open marketplace devoted to software services. Such a marketplace would offer a large collection of services which comply with the evolving sets of open standards. Figure 4.3 shows a number of possible categories for such services in such a marketplace.

Building software services

The service-oriented approach, that has been been outlined above, depends upon:

- **an ubiquitous infrastructure (generally supposed and accepted to be the Internet)**
- **large, open forums in which service providers can compete to make available the highest quality offerings.**

To ensure the greatest cost efficiency plus the highest software reliability, use and re-use of services must be as dynamic as possible. Unfortunately, the technology needed to achieve fully dynamic service consumer software is not yet available.

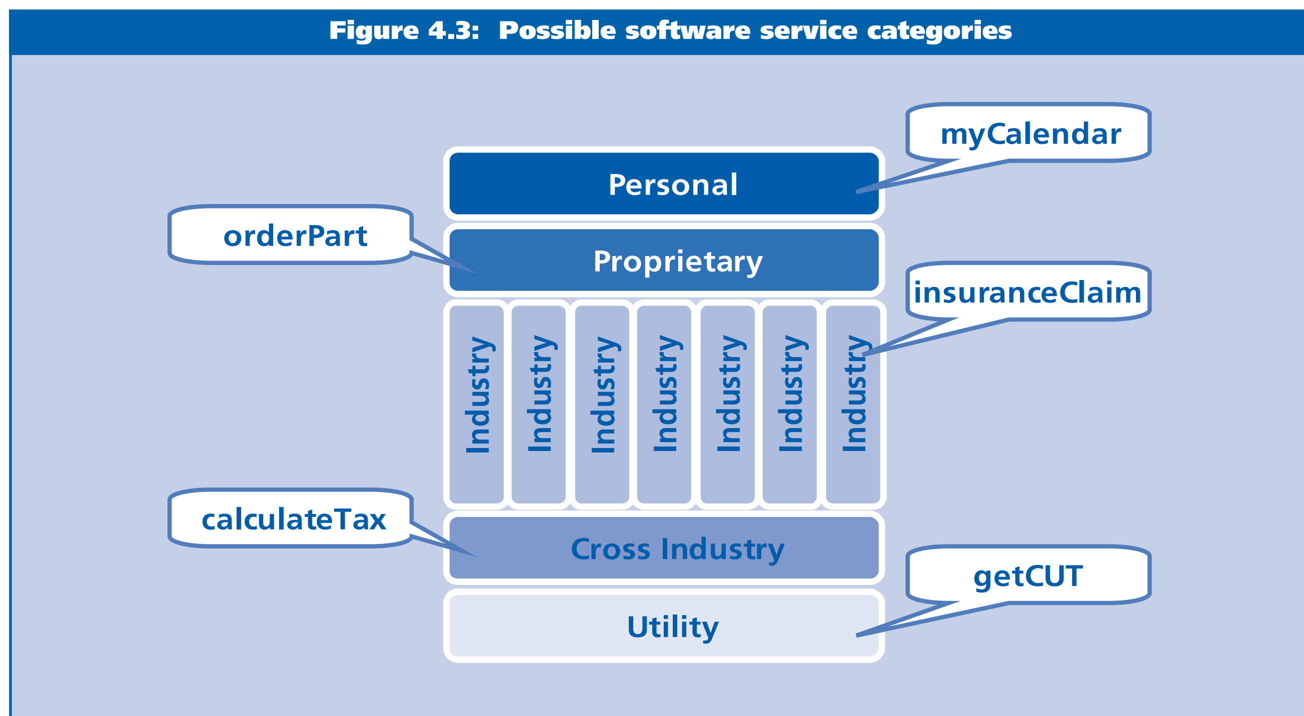
It is, however, already possible to build software that invokes statically bound services. For example, you can use IBM's WebSphere Studio development platforms to:

- **query actual service registries**
- **generate standard interface access bindings**
- **create service consumer components**

that will always use and re-use the same statically bound services. And IBM is not unique in this. Other advanced IDEs are being made available by competitors (BEA, Oracle, Sun, etc.) which offer broadly similar capabilities.

Service provider components can also be created that implement registered standard service descriptions. Again, the IBM WebSphere Studio platform can query actual service registries and generate component skeletons that can

Figure 4.3: Possible software service categories



be used to implement services. The resulting components can then be deployed in J2EE runtime containers such as IBM's WebSphere Application Server. As before, competitor vendors are doing much the same, for this is common sense.

Service descriptions and bindings

Service descriptions which comply with the WSDL standard define:

- **interface details (known as 'portTypes')**
- **the details of actual deployed service end points (known as ports).**

For any service end point there may be one or more available access bindings. This means that dynamic invocation of a service involves accessing portType proxies as well as selecting the most appropriate available end point bindings.

The combination of interface portType with selected and deployed end point binding constitutes a service connection between the consumer and provider components. This is shown in the service-oriented architecture (Figure 4.4).

The most often discussed port bindings are SOAP compliant (although service descriptions complying with the extensible WSDL standard may be based upon any well defined bindings available to both consumers and providers). Provisional service-oriented infrastructures have focused on use of synchronous SOAP bindings supported by underlying Internet protocols (HTTP, HTTPS, SMTP). For this reason they have been called 'Web Services' infrastructures.

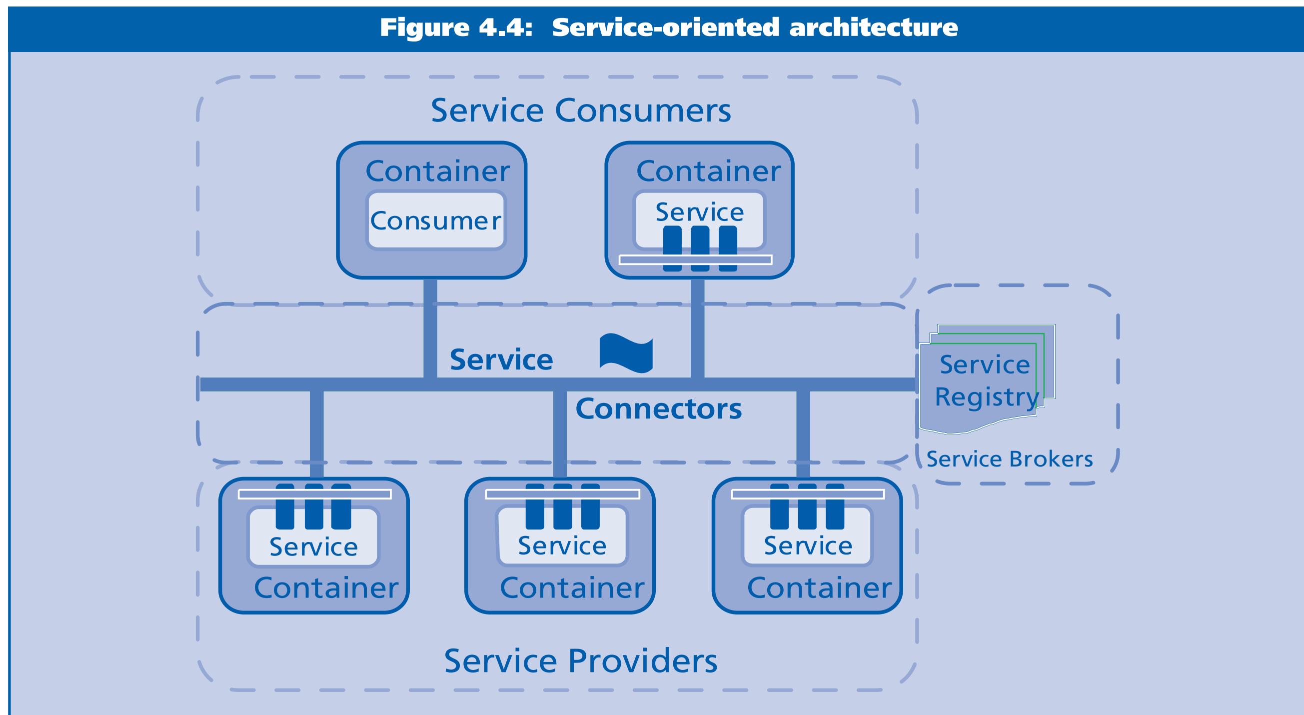
However, the lack of adequate support for security and exactly-once message delivery using these bindings has delayed adoption of Web Services technology. Indeed the search is on, as early users seek more reliable bindings for service access. This tends to be confined to trusted corporate networks or partnerships. For example, service connections based on SOAP-over-MQSeries bindings are already proving popular in some enterprise deployments. Usually, this happens where the implementer is an early adopter and transaction and once-only delivery matter.

J2EE infrastructure for services

At the same time, a number of middleware vendors have made products available that support development and deployment of service-oriented application software. IBM's WebSphere, DB2 and Lotus product families already include standards compliant support for services. Other leading vendors — including Microsoft, BEA, Oracle and Sun — have also laid out their plans for such support.

As stated earlier, the open standards for services deliber-

Figure 4.4: Service-oriented architecture



ately do not presume or target specific implementation platforms. However, the leading contenders — those that are attracting the most investment — are currently either J2EE or GXA compliant, with some additional investment by Open Source communities like the Apache Foundation.

Services designed for deployment in J2EE runtime containers (Figure 4.5) can be implemented using several different types of Java component. Simple Java classes, JavaBeans, Servlets or even JavaServer Pages can be used to implement simple services that run in J2EE Web containers. More sophisticated services can be implemented using session or message-driven EJBs that run in J2EE transaction containers.

In early 2003, the Java Community is expected to publish specifications for standard support for services as enhancements to J2EE 1.4. This will include first class support for service consumers, providers and connectors as 'managed runtime components'. The attraction for many enterprises is that this offers the opportunity to deploy one business logic which is shared by both Web users and service consumers.

In addition to standard J2EE service components, extensions are looking to support flow components which possess standard service descriptions that are accessible to local or remote service consumers. IBM is proceeding down one track. In competition, BEA, Intalio, SAP and Sun have announced the publication of the XML-based WSCI (Web Services Choreography Interface) specification.

Such flows may be arbitrarily complex a-cyclic networks of activities that implement managed stateful business processes. Flow activities may, in turn, be implemented by services that are deployed locally or accessed using Internet protocol bindings.

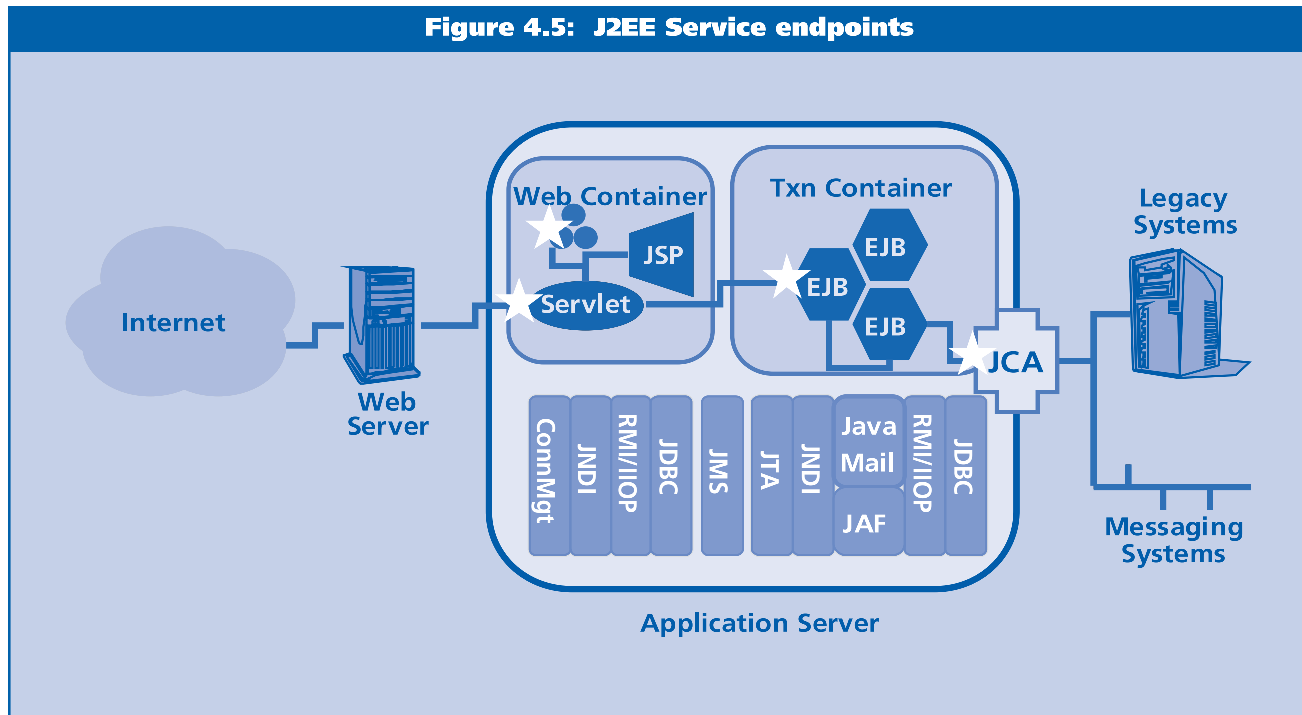
Flow components represent a powerful extension to J2EE. They embody ideas originally put forward in IBM's proposed open standard (WSFL) for the language to be used when composing elemental services into higher level services as would be required to implement the service-oriented vision. In addition, the proposed model is recursive — meaning that composed service flows, of arbitrary complexity, can be described.

IBM, or Sun and its allies, are not the only initiators in this area. Microsoft has made a similar proposal (XLANG). Work is now underway to agree an industry consensus on service orchestration descriptions. While many see this flow of services as a key step toward business process integration in large corporate scenarios, they are expected to bring broad relevance to general applications.

Re-using business logic as services

For many enterprises, integration of existing business logic

Figure 4.5: J2EE Service endpoints



running in different environments on different platforms presents a cost challenge that can only be staged for affordability. Use of open standard service-oriented technology for such integration projects has the potential to present an alternative (to traditional integration methods) in the future.

In such a scenario, selected business functions can be adapted to usage by service consumers by wrapping them in Java components (such as EJBs) that have standard service descriptions matching their invocation signatures. Indeed, certain IDEs already support the automatic generation of such wrapper components for key existing application resource types.

For other business functions it may prove necessary to use one of the many vendor-supplied J2EE Connector Architecture adapters to provide access bindings for service consumers. One example might see JCA adapters being made available for access to existing (say) CICS transactions — for use by service consumers in an enterprise scenario. Similarly, JCA adapters might be used to access existing corporate messaging systems and other legacy application systems.

Figure 4.6 shows how the deployment of service technology might occur as a logical 'bus' which provides the infrastructure needed for integration of existing business logic adapted for service access with new business service logic. Such a bus could support the flow of standard service-oriented messages between business processes acting as both consumers and providers (described using WSDL). For example, in the IBM product world:

- **WebSphere MQ could provide the reliable transport capability needed**
- **WebSphere Application Servers would provide the service connectivity.**

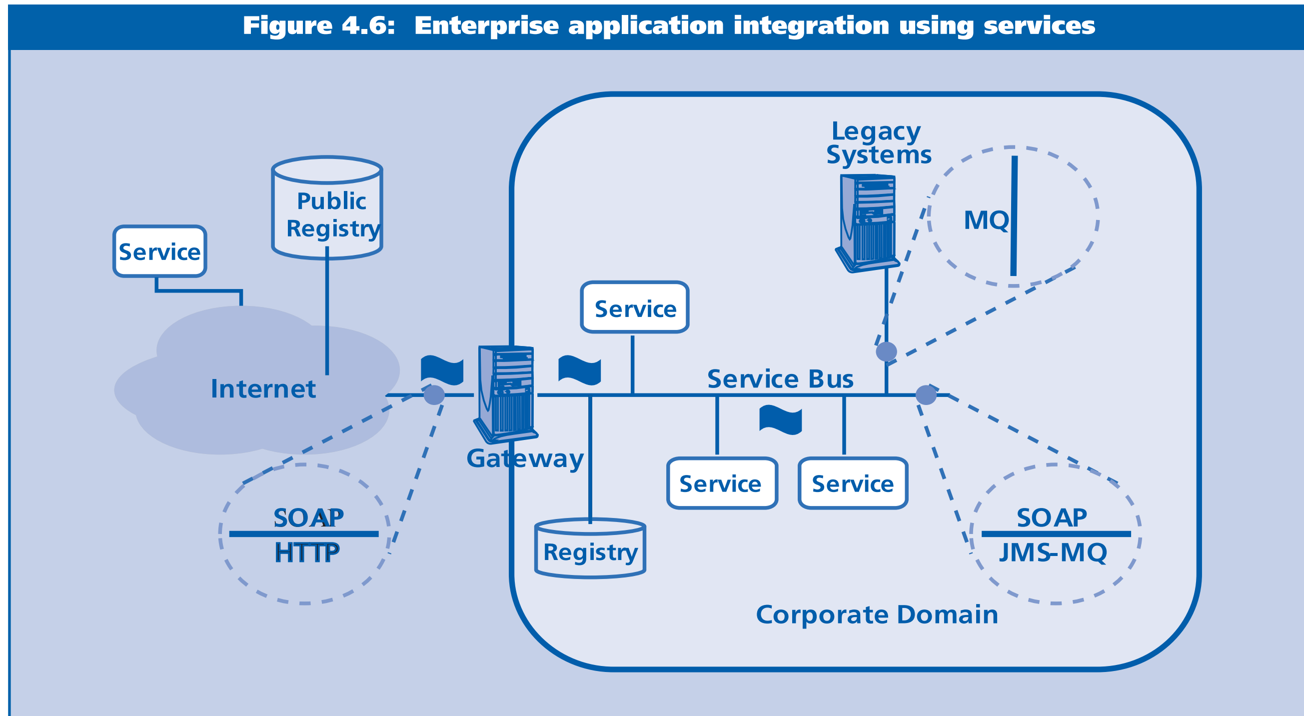
Within a trusted corporate network, service connectors can be optimized for high performance by static selection of lightweight proprietary bindings. Such bindings would be deployed in local service containers and described in local service registries.

Gateways and control points

In a large corporate network there may be several internal service registries serving different purposes. The service gateway provides a convenient mechanism for exposing only a subset of those services to external consumers — perhaps those needed to support restricted business partnerships.

Such a service gateway function is already included in several application servers products (including WebSphere) together with support for a number of different alternative service binding protocols. Figure 4.6 shows the inclusion of

Figure 4.6: Enterprise application integration using services



a service gateway ; this is located in an infrastructure to be deployed.

This gateway is transparent to service consumers and providers. But it does serve as a control point for access to enterprise services and as a protocol converter, when needed. In different scenarios such a gateway can be valuable. As a control point, the gateway can manage:

- **the application of enterprise level security policies**
- **the logging of service access for audit purposes**
- **the administration of licensing and billing for services**
- **other administrative activities.**

In some ways this is equivalent to a 'firewall' for Internet access to corporate business services. Even within enterprise networks a service gateway can be a control and integration point for re-use of business services. As a protocol converter, the gateway provides a proxy to 'internal' services that is capable of mapping from one service binding (for example, SOAP over HTTP) to another (for example, a proprietary service) without altering the substance of service requests or responses. Furthermore, service requests and responses can be intercepted for special handling using system level customization logic in a gateway.

What now?

As this analysis should make clear, many software service related issues remain unsolved in the research labs. At the same time, several key topics have yet to be agreed at the standards table, with security and reliability being the two most obvious (from a business viewpoint).

There is no doubt that Web Services technologies are garnering great vendor and press attention. But it is equally fair to say that widespread adoption has been stalled by the lack of agreement on security topics and the need for reliable, exactly-once delivery for services 'delivering value' over the Internet.

Undaunted by this, leading proponents of the service-oriented approach constantly seek to demonstrate that those service technologies that are already available can be applied to enterprise level problem spaces — such as B2B and EAI. Their enthusiasm is infectious but would-be investigators would do well to keep their implementations limited and small. Enterprise-scale expectations of service oriented software are premature, unless you wish to build everything yourself.

Management conclusion

The availability of Internet protocols on most computer systems prompted computer scientists to propose a new paradigm for developing and delivering software that addresses, at least in part, some of the major outstanding issues in software engineering. The attraction of service-oriented software is its dynamic assembly, which can then be torn down by consumers using the Internet to connect to services made available in an open market place.

While open standards have already been established for key pieces of such a service-oriented infrastructure, much remains to be finished. Security and reliability stand out as issues without which most aspirations will fail when they have to be delivered in a commercial world.

Nevertheless, leading middleware vendors — from Microsoft to IBM to Sun to Oracle, and many others — have developed standards compliant product implementations for early users who are willing to experiment and keep their initial efforts containable in size. The service-oriented approach to software already shows promise. But it will take many more years to satisfy the dream — and this will never occur if the security and reliability issues are not resolved in a way acceptable to enterprises.

Storage and middleware

Geoff. Norman
Consultant

Management introduction

Although the influences that have led to its advent have been evident for at least five years, the emergence of a distinct software product category called 'storage middleware' may yet turn out to be one of the more notable IT phenomena of the 12 months. Equally, given the IT industry's predilection for smoke and mirrors, it is reasonable to ask whether or not 'storage middleware' is a valid descriptor.

Adding to the confusion is the reality that describing middleware is rather like a sports car: while it is relatively easy to decide whether or not a particular auto fits into the category, defining it concisely can be difficult and generally involves a list of principle characteristics. In this context, the distinguishing features of middleware include:

- ***the ability to cope with, and provide linkage between, differing computing platforms or domains***
- ***dependence on effective message handling***
- ***an emphasis on interfacing, often at both the application programming interface (API) and call level interface (CLI) levels***
- ***the need to demonstrate industrial-strength qualities, at least up to the level of the associated platforms.***

The question is: does 'storage middleware' satisfy these characteristics? In this analysis, Geoff. Norman:

- ***looks at the evolution of storage and middleware***
- ***locates the relevance of eXtensible Mark-up Language (XML) and Simple Object Access Protocol (SOAP) in recent storage management initiatives***

All rights reserved; reproduction prohibited without prior written permission of the Publisher.
© 2002 Spectrum Reports Limited

- ***examines examples of storage middleware, including those from EMC (WideSky), IBM (Storage Tank), as well as HP, HDS, CA, BMC, Veritas and others***
- ***suggests how the product category will evolve.***

Enterprise storage evolution

Storage middleware owes a great deal to the emergence of networked storage, in the form of storage area networks (SANs) or network-attached storage (NAS). These have come to prominence in the last two or more years. By facilitating the physical pooling of resources across the storage network fabric — both 'downwards' to storage devices and 'upwards' to application servers — networked storage has broken the previously exclusive link between the physical pooling and the network fabric. In turn, this has opened opportunities.

To trace fully the whole history of enterprise storage is a subject in itself. What will become clear here is that storage middleware has emerged as part of an evolution of large scale subsystems. Today, the demands placed upon storage by enterprise computing can be summarized as:

- **achieving optimum total cost of ownership (TCO)**
- **while also providing improved management of multiple resources, including both hardware and software.**

The second of these can be broken down further into support for:

- **intra- and inter-device functions, including disc array configuration, local and remote data mirroring, local and remote data replication, and local and remote snapshot copying (at the volume, file and block level)**
- **centralized data administration (also known as console consolidation)**
- **increased levels of management automation**
- **assured availability and recoverability**
- **storage pooling**
- **application server pooling**
- **data sharing between application servers**
- **freedom of data movement, between storage devices.**

The similarity to that expected from traditional middleware — inter-domain linkage, message handling, interfacing and industrial strength operations — should be clear. In addition, the effective management of heterogeneous storage environments itself demands middleware to operate. Conceptually, at least, the basic similarities are substantial.

However, the connections between storage and middleware are only just beginning to be made. In the past storage possessed both a certain lack of maturity and the luxury of being sold in a sellers' market. This enabled storage developers to adopt a piecemeal approach. Consistency was not yet important.

That has changed. As I shall discuss, the move from a sellers' market to a buyers' one has seen customers demand more sophisticated solutions with which to draw hitherto disparate functions together into a single storage spanning mechanism.

From application layer to storage subsystem

The movement of at least some storage management function into the storage subsystem (rather than the applications layer) has been coupled with a desire to centralize storage administration. This depends on:

- **links between a multitude of storage device management components**
- **the selection of at least one of several overall storage — and possibly systems — management frameworks that are available from several major vendors.**

But centralized operation depends on more than merely providing links to an operator console. In common with other middleware, there is a wide range of data conversion which needs to be carried out. This includes aligning the content of messages and translating (necessarily) common input from operators (or from applications that support operators) into forms suitable for each of the attached devices.

This extensive mapping requirement explains why — in common with conventional middleware — storage middleware is moving towards a commonly-accepted metadata standard. As might be expected, XML increasingly crops up as part of the description of storage middleware.

In addition, 'virtualization' has become the flavor of 2002. If fully implemented, virtualization will enable all storage resources (across all attached devices) to be made available to any application server as a single logical pool. While no proposed virtualization scheme quite delivers this in prod-

uct as yet, most storage products are moving to create an ability to interface to a wide variety of storage components. Virtualization is going to provide still further opportunities for storage and middleware to come together.

Falling hardware costs

There is one further pressure that is pushing storage to the fore. This is that the unit price of raw disk drive capacity is decreasing at something like 30% per year.

The consequence is that an decreasing proportion of the value in storage consists of the commodity components that vendors sell as part of their underlying architecture. Profit opportunities, and margins, from hardware are becoming limited.

This is obliging vendors to look for product differentiation elsewhere. Software has become one such favored source of new margins. This should not be surprising. After all, EMC showed the way in the mid-1990s.

With more software comes services. All the major suppliers now expect services revenue to replace hardware margins. Software materially assists the sale of services and I expect a move:

- **away from the current support and services offerings, which have a lot to do with the mechanics of configuring, testing and implementing storage environments**
- **towards an emphasis upon storage management products that embed software and middleware and require services to match the storage to the demands of each particular customer.**

Routes to satisfying user demands

Satisfying the user requirements outlined above will depend on:

- **discovering the ways in which data is actually employed in user environments**
- **understanding that (data) usage.**

Initially this will determine the characteristics of hardware and software products. Subsequently, it will be carried on inter-actively and then pro-actively to match systems configurations to business demands on the fly.

For example, EMC has already showed its willingness to tailor storage subsystems to data characteristics. In April 2002 it introduced Centera, a simplified and (relatively) inexpensive disk array designed specifically to store fixed-content files such as:

- **medical records**
- **geophysical data**
- **video files**
- **e-mail attachments**
- **archived financial records.**

What has become clear is that increasing the diversity of storage devices also increases the complexity of the enterprise computing environment. In turn this emphasizes the importance of delivering linkages between the various domains within an enterprise computing environment.

Managing (and this includes concealing) this level of complexity means that systems management in general, and storage management in particular, will come to depend on middleware optimized for storage if it is to provide the logical and management links between components. Inevitably this must include interface alignment.

The emergence of storage middleware

Figure 5.1 offers a schematic representation of storage middleware. With this in mind, the following 'caveats' should be noted:

- **there is a continuing debate as to whether the management path should follow the data path (in-band) physically, or be separate from it (out-of-band)**
- **the storage administration platform and the application servers are the only server platforms that currently participate; expect other servers, such as those shown in Figure 5.1, to be included in future**
- **while the data path may not pass through the middleware layer, middleware's logical involvement is critical; inevitably it will also impose a performance overhead**
- **while application server A may run UNIX and read data in CIFS format and Server N may run Windows and read NTFS files, it is probable that some (or even all) of these data views will overlap**
- **storage middleware is definitely in the management path**
- **storage devices are shown as grouped within storage resource management domains, with each having its own API.**

A key conclusion should be that the number of interfaces involved — principally APIs, so this term will be used in a general sense — can reach such high levels that the writing, testing and maintaining of software which conforms becomes so onerous that the rate of functional progress ends up being compromised. Indeed, as with conventional applications, there is evidence that this is already occurring.

Part of the rationale for traditional middleware was that it was wasteful to have to write and test different versions of every API. This has little tangible benefit and significantly increases the cost and delivery time. It is no different for storage management. Far from being the useful differentiator that it once was, encouraging customers to stay within the bounds set by one vendor's particular combination of hardware and software products has become a hindrance.

Instead, APIs need to become transparent. In an ideal world, all the suppliers of storage devices will write to a single common API set. Although this is IT's perennial standards debate, 100% API commonality will not occur in the storage arena (although adoption of XML will assist). Nevertheless, extending the links from one domain to others within enterprise computing guarantees that coping with a variety of APIs will be a requirement for the foreseeable future. Indeed, given the consistent demand for interfacing — both within the storage domain and between one storage domain and other domains across enterprise computing — it was almost inevitable that some form of 'storage middleware' would appear, either by stealth or by design. In retrospect, several examples have existed for some time but it was an EMC announcement in October 2001 that first presaged storage middleware as a distinct product category.

That said, it can be difficult to discern boundaries. While EMC may have made an early move into storage middleware, this does not mean that all categories of storage management software should have the 'storage middleware' label applied. For example, operator support, storage administration tools and individual device management routines are not really broad enough in their current form to qualify as middleware, though the first two may well evolve to warrant inclusion later. On this basis, the following have yet to 'embrace' storage middleware:

- **Computer Associates' (CA) BrightStor**
- **Hewlett-Packard's (HP) OpenView**
- **Hitachi Data Systems' (HDS) HiCommand**
- **Tivoli's Tivoli Storage Manager (TSM).**

To summarize: storage middleware has three dimensions. It provides and enables links:

- **within storage resource management (SRM), for example between SRM entities**

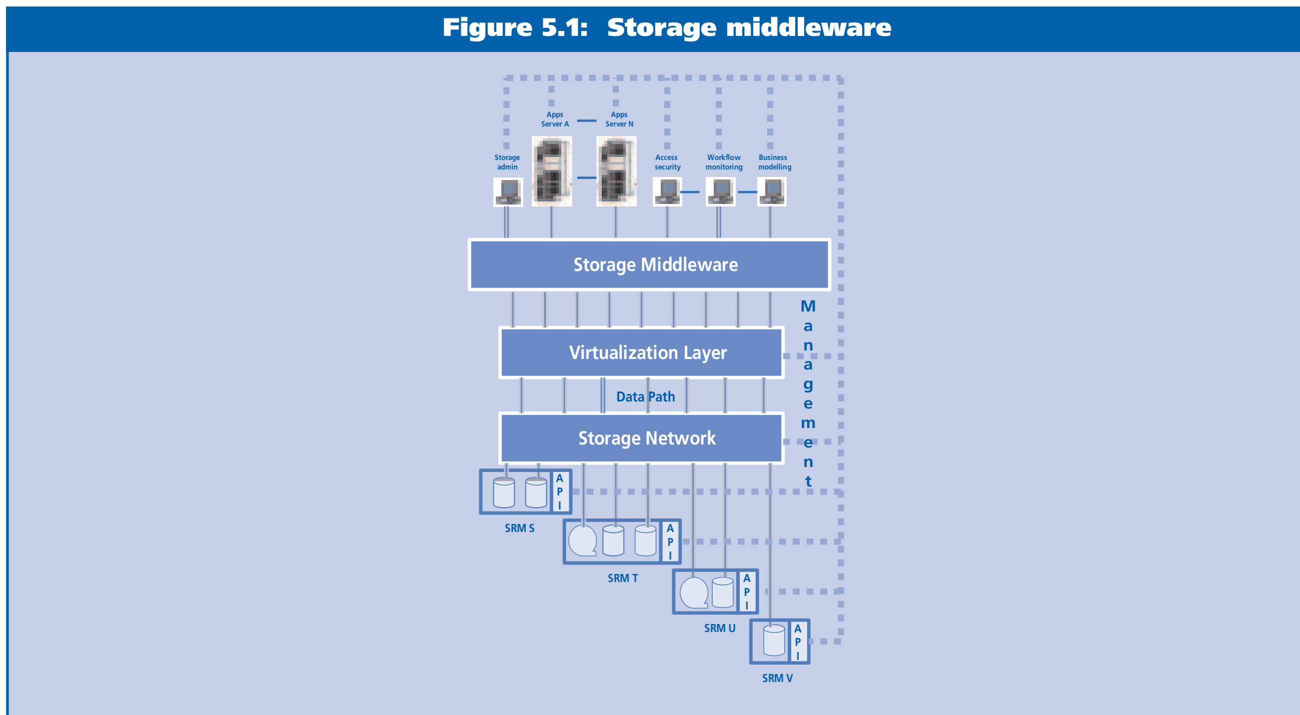


Figure 5.1: Storage middleware

- **within the storage management domain, for example between SRM entities and other storage management functions**
- **between a storage management domain and other computing domains.**

The next step is to look deeper at a couple of prominent storage middleware examples: WideSky (from EMC) and Storage Tank (from IBM). In addition, several others warrant a brief mention (including HP and HDS).

WideSky

EMC has pursued the theme of enterprise-level consolidation consistently since the mid-1990s. In October, 2001 it introduced Automated Information Storage (AutoIS). AutoIS extended that principle to encompass storage management by broadening the ability of EMC's existing range of storage management products to interact with non-EMC hardware and software products.

A central feature of AutoIS is the WideSky storage management middleware (Figure 5.2). This provides two way interfacing across heterogeneous storage environments and includes:

- **all significant storage devices, attached to any applications platform**
- **a wide range of systems components — including volume managers, file systems, network switches and DBMSs.**

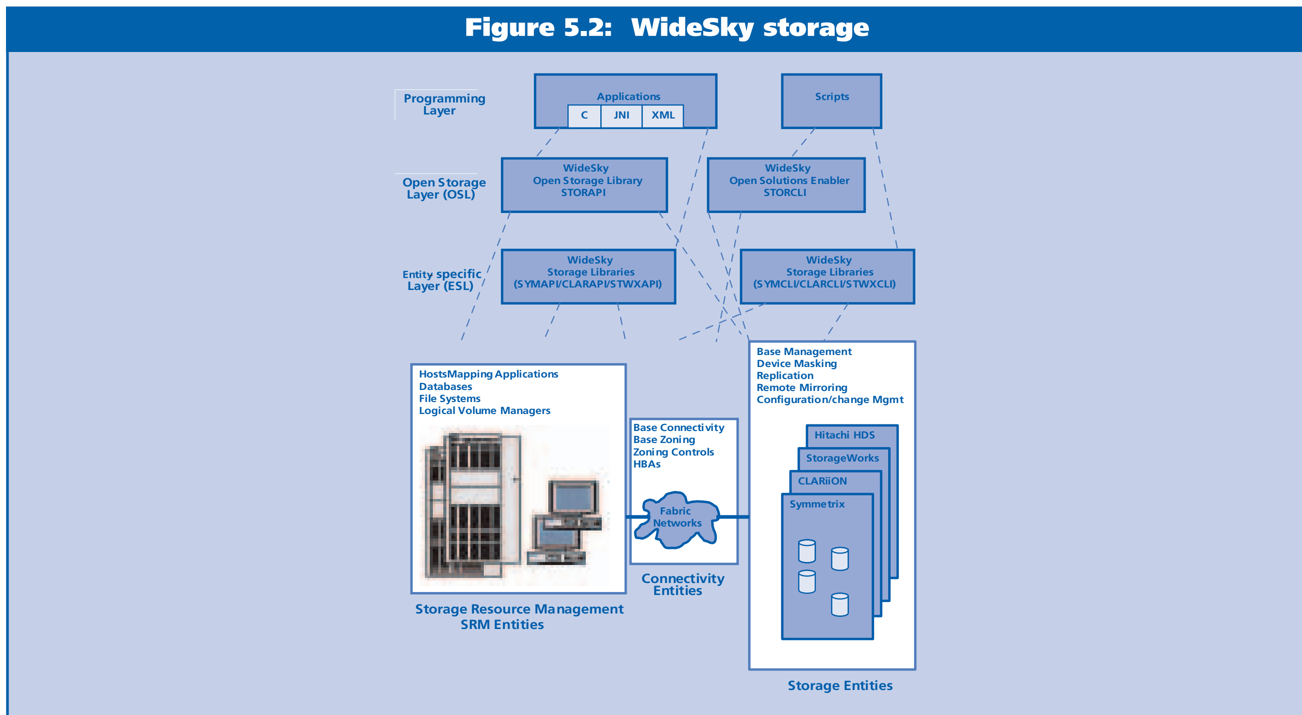
In March 2002, EMC announced the WideSky Developers Suite. As the name suggests, this Suite encourages third party vendors to build WideSky compliance into their products by providing a wide range of support through the development phase. Encouragement includes:

- **free membership of the Developers Program**
- **a free software development kit (SDK)**
- **distribution executed either through EMC or the third party vendor, with a mechanism for charging customers a run time licence fee**
- **direct EMC support, including use of EMC laboratory facilities and engineering.**

By mid-2002 EMC had already attracted 100+ partners, including suppliers of:

- **systems software**
- **database systems**
- **applications**
- **server platforms**
- **storage devices**
- **network switches.**

Figure 5.2: WideSky storage



To deliver its objectives, WideSky supports three categories of API, described as follows by EMC:

- **a Storage Resource Management (SRM) API; this obtains information about the total storage environment — from database composition down to raw physical storage — and identifies host objects as well as mapping databases, file systems and volume managers through layers of virtualization down to the physical storage**
- **a Storage API; this obtains information about heterogeneous storage arrays and uses a generic interface that enables heterogeneous storage management, provisioning, performance management and high availability**
- **a Connectivity API; this obtains information about the total storage network configuration plus it provides discovery among switches, storage arrays, HBAs, hubs and other connectivity elements: it uses zoning to control I/O path management.**

Not surprisingly, the SRM API was available immediately for EMC's own Symmetrix and CLARiiON disc arrays. The other APIs will become available in the second half of 2002. Similarly, support will also be extended to third-party devices.

EMC plans an incrementally widening coverage of storage devices, server platforms and storage management components:

- **initially through WideSky alone**
- **then via WideSky with vendor co-operation on APIs**
- **eventually through WideSky with vendor co-operation plus the adoption of industry-wide standards.**

EMC's underlying commercial aim is to enable, via storage middleware, an increasing range of storage devices, servicing an increasing range of applications servers. These will be managed centrally via EMC's own management and administration products.

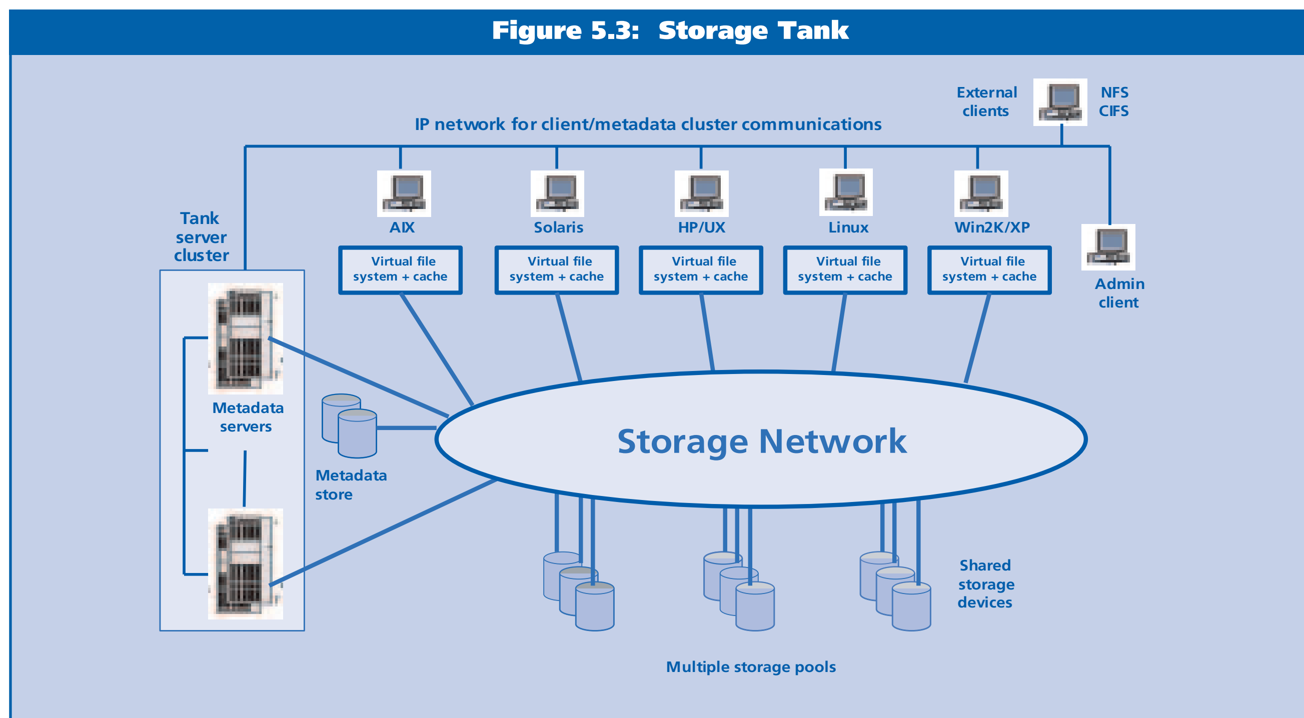
IBM's Storage Tank

IBM's storage middleware product, Storage Tank. Storage Tank is intended to operate in conjunction with IBM's Virtualization Engine (Figure 5.3). Both run on high-availability, Intel-based, xSeries Linux clusters.

The principal features of Storage Tank are:

- **the provision of a common file system**
- **a scope that will initially be SAN-wide, with extensions to cover heterogeneous enterprise environments**

Figure 5.3: Storage Tank



- **the inclusion of (complex) administrative tasks (such as file locking)**
- **the use of a metadata server, to move intelligence and processing workloads into the network and away from individual storage device controllers and individual application servers**
- **administrative consolidation (this is an intrinsic element of the structure and should aid policy-based, operational automation)**
- **the introduction of an approach by which each application server runs a Storage Tank client**
- **the adoption of a level of abstraction that enables physical and logical data to be hot-pluggable**
- **provision of NAS file server function via a NAS gateway accessing SAN-attached physical devices.**

In addition Storage Tank proposes to adopt established or emerging industry standards:

- **XML, to represent classes and methods**
- **the Storage Network Industry Association's (SNIA's) Common Information Model (CIM), to support a universal data representation and interfacing**
- **the HyperText Transfer Protocol (HTTP), as a transfer method.**

Storage Tank has a strange history. It was first announced in 2000, for delivery the following year. It then became the responsibility of IBM's Tivoli systems software company. Little more was heard of it until April 2002 when it was revived as an IBM Storage Systems Group (SSG) product — with initial delivery of some function postulated for 2003.

To underline the middleware connections further, both Storage Tank and MQSeries are developed at the same IBM Lab. (Hursley, in England). Prima facie, it would seem attractive to make use of the robustness and industrial strength of CICS or MQSeries within Storage Tank. Thus far, this would seem only to be a logical step rather than a real one.

That said, the Storage Tank concept is an ambitious one — not least because of its commitment to distributed and disparate file management. Somewhat surprisingly, however, it does not encompass either tape storage devices or mainframe applications platforms and file formats. As it evolves to encompass other features — such as allocating resources to data on the basis of business priorities and applying access security within the storage subsystem — it will have to exercise other negotiation and locking procedures (although it is not yet clear that Storage Tank development engineers are fully aware of the implications or that this is a necessity).

In a joint announcement, IBM and Hitachi — which sell large disk arrays through Hitachi Data Systems (HDS), Hewlett-Packard and Sun Microsystems — have committed to:

- **adopting industry standards (as mentioned above)**
- **using the same virtualization scheme.**

While adherence to an industry standard is conceptually good, it could slow down the introduction of some functions, especially compared to a large vendor pushing its own 'standard'. IBM has undertaken to be pragmatic in moving ahead of the standards bodies, especially if delivering function might be compromised.

In the IBM example, it is worth noting that IBM is exploiting its storage middleware experience as a middleware developer. IBM has an unrivalled background in transaction processing middleware (CICS, IMS/TM, MQSeries, etc.). It had also established, though failed properly to exploit, a promising storage hardware approach — attaching all manner of storage devices to the storage transport network via a common storage server built round its own AIX implementation of UNIX. While some of the lower-level functions within Storage Tank will take over the role of the dedicated storage server, albeit initially offering less functionality, it will be interesting to see how other conflicts are resolved.

Other examples of storage middleware: HP, Sun, HDS and others

There are many other examples of storage middleware from various types of vendor. Some of the more interesting are mentioned briefly below. The major end to end systems vendors — like Hewlett-Packard and Sun Microsystems —-have the same motivation as IBM and EMC for competing in the storage middleware arena — namely the need to improve margins over those obtained from pure hardware sales.

HP announced its Federated Storage Area Management (FSAM) in the second quarter of 2001, predating WideSky. It did not, however, try to establish the storage middleware category on its introduction. Perhaps this was sensible.

The initial launch of FSAM was unimpressive, to say the least. The picture given was of a slightly warmed-over

version of OpenView, ostensibly aimed at managing heterogeneous storage environments but looking more like a management-driven magnet towards HP storage devices. HP's acquisition of StorageApps, with the avowed intention of adding the latter's virtualization capability to OpenView, has strengthened its storage middleware credentials. Nevertheless, the most odd omission in FSAM is its lack of any emphasis on interfacing.

Meanwhile, Compaq had been developing its own Versastor software, benefiting from its success in offering networked storage scalability from a lower level than its competitors. How this will be affected by the HP takeover remains to be seen. It seems extremely unlikely that both Versastor and FSAM can survive or be combined.

Despite its protestations, Sun is not a leader in storage middleware even though Sun did start to promote Jiro, a Java- and standards-based storage management architecture, in mid-1999. Looking back, Jiro possesses many of the characteristics that would now be recognized as storage middleware.

Despite changing the name to Federated Management Architecture, Sun has not delivered any significant take-up. Simultaneously, the storage industry skilfully failed to pick up on an exceptional opportunity, one which had the added advantage of sharing the Java skills built up in the applications development area. While it is reasonable to argue that the time was not right for the major storage players to pursue a common API, Sun has been equally ineffective in its promotion of Jiro's and FMA's advantages.

HDS is primarily a hardware vendor that is turning to storage products. It needs to bolster its software portfolio if it is to support sales into enterprise accounts. While it was an original (prominent) supporter of Jiro, it now needs to add more to TrueNorth, its overall storage management architecture, if it is to obtain storage middleware attributes.

Software-only storage middleware vendors

Software-only vendors are also responding to user demands for storage middleware. As might be expected, the emerging products mentioned below reflect each developer's background.

Computer Associates clearly needs to extend its enterprise systems management portfolio, particularly its BrightStor storage component, to incorporate storage middleware functions. The BrightStor portal will move in this direction but has yet to do so.

BMC Software's Applications-Centric Storage Management (ACSM) makes use of intelligent agents — similar to those employed in BMC's Patrol systems management suite — to provide interfaces between applications and storage management functions. This puts it closer to the storage middleware category. Indeed, ACSM's announcement, in 2000, made BMC one of the first to recognize the value of middleware in storage — although its scope is substantially narrower than that of WideSky or Storage Tank.

Veritas Software has yet another different perspective. It has successfully established itself in a variety of categories but, in a middleware context, its most interesting products are the file system and volume manager employed widely on UNIX and Windows servers. Veritas also offers a wide range of data management, backup and restore functions. Like HDS, it was an enthusiastic supporter of Jiro. It was attracted by its simplicity and the ability to concentrate on developing software-derived function rather than working to a variety of APIs.

In April 2002, Veritas announced its Adaptive Software Architecture (ASA), with the specific aim of reducing complexity in heterogeneous computing environments. Details remain scanty, but ASA is certainly middleware. ASA confirms that Veritas has recognized the need to provide links to other computing domains in order to direct storage subsystem activities. This is important for Veritas because widespread use of Storage Tank (for instance) could substantially erode Veritas' revenue opportunities.

DataCore and FalconStor are relatively new companies. They offer storage management software aimed at providing enterprise-wide virtualization — with strong middleware elements in their products. Neither of these companies has achieved notable sales figures, though the former has made collaborative agreements with IBM, Hitachi and Fujitsu.

Storage and storage middleware futures

While it is not the purpose of this analysis to delve into the detail of how enterprise storage hardware is put together, its evolution will shape the direction taken by storage middleware.

For the immediate future, it is essential to remember that virtually every enterprise storage environment will contain a range of devices that are already up to three or four years old. It is desirable, therefore, to enable these devices (possibly through middleware) to contribute to more closely-interfaced storage and systems management

environments. Achieving this will lengthen the life of existing hardware in the front-line.

Simultaneously, storage management will expand to include a whole range of activities beyond traditional storage-specific management. Eventually these will enable the most advanced storage management environments to react directly to — and even anticipate — the influences and events occurring in other system domains (rather than only reacting to events within their own storage domain, as they do now). Such proactive storage management will place heavy demands on storage middleware.

While standards initiatives — such as SNIA's CIM — may impose a restricted life on some aspects of storage middleware (for example, coping with disparate interfaces), the wider range of inputs that will influence storage management in future appears to guarantee that storage middleware will have a significant function for the foreseeable future. Examples of other domains include applications themselves and applications-based middleware. In product terms, the load-balancing contributions of IBM's CICS or MQSeries or BEA's WebLogic have storage implications. These will be best served by storage management systems that benefit from close links to them.

Data recovery is yet another area for inclusion and evolution. Greater efficiency through work flow monitoring will be particularly useful in managing e-business systems. This also will depend on effective middleware. Indeed, IBM is already looking ahead to a time when storage management will:

- **interact with events and conditions**
- **influence storage components beyond the enterprise's own boundaries.**

This is a clear instance where the scope and importance of storage middleware will be extended.

The development of new types of storage devices will continue. It is quite feasible that optical disks could stage a revival when incorporated into a networked storage environment. This might, for example, combine well with the scope of virtualization activity — although it is likely to require further mapping and interfacing activity.

The long-term future of WideSky is open to debate. This is not meant as a criticism of WideSky, and EMC must be commended for putting development effort into an innovative product. Nevertheless, WideSky could be sowing the seeds of its own irrelevance. The more WideSky — or some other equivalent — achieves, the more likely the industry is to accept and pursue the benefits of a common API set. EMC envisages a future with industry-wide APIs, but every move towards them inevitably reduces the value of the EMC middleware.

Though a mechanism such as WideSky can be expected to add further value beyond a 'pure' linking role, its principal raison d'être may yet be diminished by its own success. As a means of attaching EMC's other storage management software products to a wide variety of devices, thereby increasing their effectiveness, it offers value to users. But that is a long way from convincing them to pay a realistic price for software that is 'only' an enabler rather than a source of distinct function. On the other hand, in the short-to-medium term, WideSky offers:

- **higher levels of interoperability sooner than Storage Tank**
- **support across a wider range of storage platforms, including, of course, those that conform to industry standards.**

Storage Tank is still some way from delivery, thereby offering more pragmatic solutions a head start. After that, the worst that could happen to IBM and Hitachi (and the Hitachi resellers) will be to find themselves the only adherents to open standards. While the long-term benefits of Storage Tank have the potential to be considerable, it is an extraordinarily ambitious project. It will test IBM's resolve and patience, and that of its customers.

IBM has already stated that Storage Tank agents will be no-cost items, though there will be opportunities for metadata and virtualization server sales. Being able to offer the highest level of interoperability among adherents should encourage storage device sales. But it is inevitable that lower levels of function will be available to non-conformant components.

Management conclusion

The way storage middleware is evolving suggests that there are two basic approaches. The first is to:

- ***accept (some might suggest 'welcome') the differences between management APIs***
- ***enable interaction between devices with operational support tools via software that manages the complexities of any-to-any interfacing.***

Value-added function can then be added on top of these interfacing duties. As currently implemented, this approach

is intended to draw mixed environments towards the middleware developer’s range of storage resource management and administration products.

The second approach is to embrace an industry standard — or standards — and expect that all attached devices will do the same. This has the advantage of leaving software developers to concentrate on adding function. Some vendors have compromised, with mutual interface-exchange arrangements. But these do not serve any long-term purpose, especially if they delay moves towards a comprehensive solution.

Storage middleware is already complex. It is costing hundreds of millions of dollars to develop, and it will save huge amounts of development investment once in production. Unfortunately, storage middleware has one other key similarity to traditional middleware. Even though the provision of effective storage middleware promises to make a substantial contribution to the overall function of enterprise storage environments, it is still not clear how storage middleware will generate sufficient revenue directly to pay for itself. As with traditional middleware, customers may come to expect such function to be a part of their environment for which they do not expect to pay ...

High availability considerations for messaging hubs

Nick Denning
Chairman and CTO
Strategic Thought

Management introduction

The design of highly available (HA) hub-based middleware systems is not simple. Various topology options are possible. For example, a hub approach also implies a single point of failure. Rarely is this acceptable — and the events of September 11th in New York and Washington have only reinforced the lessons already learned in London after the bombings of the NatWest Tower and the Docklands by the IRA.

As a consequence a message hub-based architecture, if it is to be deployed, must be designed to enable fail over by an appropriate mechanism, without loss of data or transactions. With any need to implement high performance and high availability features, it is vital that the underlying issues are understood so designers can take specific measures to deliver mission critical capable systems.

In this analysis, Nick Denning — the Chairman and CTO of Strategic Thought, based in Wimbledon, England — shows how subtle differences in behavior can occur as the sophistication of a messaging infrastructure increases. He uses IBM's WebSphere MQ (WMQ), Websphere MQI (WMQI) and other AIX and Solaris-based solutions to illustrate his points.

All rights reserved; reproduction prohibited without prior written permission of the Publisher.
© 2002 Spectrum Reports Limited

To hub or not to hub, that is the question

Let me start by comparing and contrasting the three principle topologies for messaging and determine the criteria for selection. The options are:

- **point to point, where each system has a specific connection to each other system**
- **bus-based, where all messages are placed on a bus and the bus moves the messages from the generator to the consumer through the most appropriate path**
- **hub-based, where all sources and destinations communicate directly to a hub.**

The critical difference between the hub-based one and the other solutions is that all traffic arrives first at a hub and is then dispatched, after processing, from that hub. This means that the order of all message processing is preserved and each system receives its messages in an identical order.

This is not the case in a bus architecture. Two systems communicating on a bus that are co-located or adjacent will typically be able to send and receive messages far faster than they will when communicating with any distant or remote system.

Thus the location of a single hub brings immediate advantages because it:

- **enhances the ability to monitor traffic**
- **offers a readily understood location from which to provide logging and audit facilities.**

But the downside of a hub architecture is that it introduces a single point of failure. Loss of a hub potentially affects all applications connected to it.

The use of a single hub for, say, an international business could result in complete disruption to the hub if it fails and therefore a disruption to the overall business. Similarly loss of the network connections between any site and the site supporting a single hub can take out all communications for a part of the system.

Use of a central hub also increases the level of traffic across the corporate network. While it is possible to provide a distributed hub approach — so that a local hub in each office brokers the local traffic and dispatches any messages for the rest of the network up to the corporate hub — this is not necessarily simple. But this does enable local systems to intercommunicate in the event that the central hub, or connections to the central hub, are lost — at the price of a more complex task when activities like central logging, management and auditing are considered.

This brings into sharp relief the different requirements for hub, bus and point to point:

- **point to point is only applicable for simple connections; it is straightforward to set up but the management effort increases linearly with the increase in the number of connections**
- **a bus architecture, such as that provided by TUXEDO or WebSphere, is more complex to set up and manage than point to point; once in place, however, it is relatively straight forward to add additional clients and servers into the architecture**
- **a hub architecture enables the management of messages through a single point to ensure order and equal treatment, particularly when one or more messages must only be processed and released by each application as a group under transactional control.**

Furthermore, while effective and efficient use of resources is enabled across a bus, little account of ordering is taken, without copious additional effort. Messages are outside the scope of transaction control. This is acceptable for client applications communicating with a server or set of servers, particularly when a transaction spans a number of calls and is designed to manage all work carried out by the servers across these multiple messages. It is most applicable where intra-application communications are needed.

In contrast, a message hub approach is suitable for both inter-application and/or intra-application communications. The strength of the disintermediated, or decoupled, approach lies in its additional capabilities, like:

- **transformation**
- **routing**
- **protocol conversion**
- **publish and subscribe.**

Having said all this about the advantages of a hub, it is now necessary to address the issue of the single point of failure. Furthermore, this illustrates that one size does not fit all except in the simplest of scenarios (where there is no requirement to implement some of the more advanced features I mention).

Simplification by design

In the event that two applications communicate via a single

point to point queue, then those applications can make assumptions about the behavior of (say) IBM's MQSeries. However, if a hub provides disintermediation between applications, then those assumptions may be invalid.

If applications implement the following features within their protocol, then the implementation of a reliable and highly available hub should be straightforward:

- **applications should ensure that message order does not matter**
- **messages generated should be atomic, and can be delivered in any order**
- **where order does matter, it should be the task of the application (or the adapter) to manage that order, either within the receiving or transmitting adapter or in the application itself; in a low load environment the simplest choice is to require an acknowledgement to each message before the next is sent but in a high throughput environment this is not acceptable and the receiver must carry the burden of marshalling the message order**
- **applications, or their adapters, should ensure that they can first detect (within the required timeframes) the loss of a message and then have the ability to re-transmit a lost message as well as being able to detect any duplicate message once a mislaid message is discovered and forwarded**
- **the receiving application should have an adapter instance connected to each queue manager containing a cluster queue instance**
- **if the transmitting application looses its connection to a queue manager, it should be able automatically to reconnect to an alternative queue manager**
- **applications should not use dynamic queues to return traffic (while this is not mandatory it does simplify the architecture)**
- **transformation data required by the WebSphere MQI should be delivered to multiple hubs independently and should have a time stamp showing when the data applies**
- **messages should have a timestamp that will enable the transformation applicable to the message to be applied (using time stamped transformation data) so that validation rules applicable to the data when it was generated can be used if necessary.**

If applications can fully implement these measures then the design of a high availability message architecture will be relatively easy. The cost of creating such an environment can then be minimized.

A hub architecture based on a WebSphere MQI cluster — implementing multiple queue managers and multiple instances of a cluster queue — means that all messages have multiple routes through the hub (Figure 6.1). Transmissions, therefore, never stop. In the event of a failure a re-transmission takes place automatically, but multiple transmissions are detected and discarded.

Possible alternatives

However, observing these requirements is onerous. Many may decide that they impose overly significant responsibilities on the application developer — something that many IT shops often prefer to avoid. Where it is not possible to deliver such measures within applications, the general responsibility for messaging must be born by the messaging infrastructure itself. At this point it is the responsibility of the middleware (not the applications) to reduce to an acceptable level the risk of losing or misplacing a message for an unacceptable period of time.

This almost certainly requires the deployment of some form of high availability approach and duplication of services within a distributed architecture, either in hot, warm or cold standby. In such instances, the architecture will be required to:

- **enable the automated fail-over of a hub, so that service is restored as soon as possible after a failure**
- **ensure that a system can automatically be redirected to an alternative, if a full site is lost**
- **minimize the risk of loss, or mis-placing, of messages where only a single copy of a message exists.**

High availability requirements

Before discussing the implementation of a high availability hub, it is worth reconsidering the overall requirements, as defined above. For example, do the applications actually justify the expense of upgrading the hub to possess high availability (HA) protection?

Our experience shows that it is expensive to implement high availability. If the applications are HA applications then it is clear that the hub must be HA protected too. If the applications are not themselves based on a high availability (HA) architecture then it might be possible to justify enhancing only the hub to provide HA facilities.

The difference is that when using a single central hub architecture, a failure of the hub may impact all the systems that use it, while the failure of one application has a far more limited impact. As a failure of the hub would impact all systems, it probably justifies HA protection even if any one application does not.

An alternative topology may be to consider multiple hubs for groups of systems:

- **all HA systems could use an HA protected hub and all non-critical systems a single hub protected by lower reliability measures; this would at least mean that the power of the HA architecture could be minimized**
- **increasing the number of hubs to multiple instances will decrease the impact of an individual failure, but will add capital and operational costs that may exceed the costs of a single hub with HA protection.**

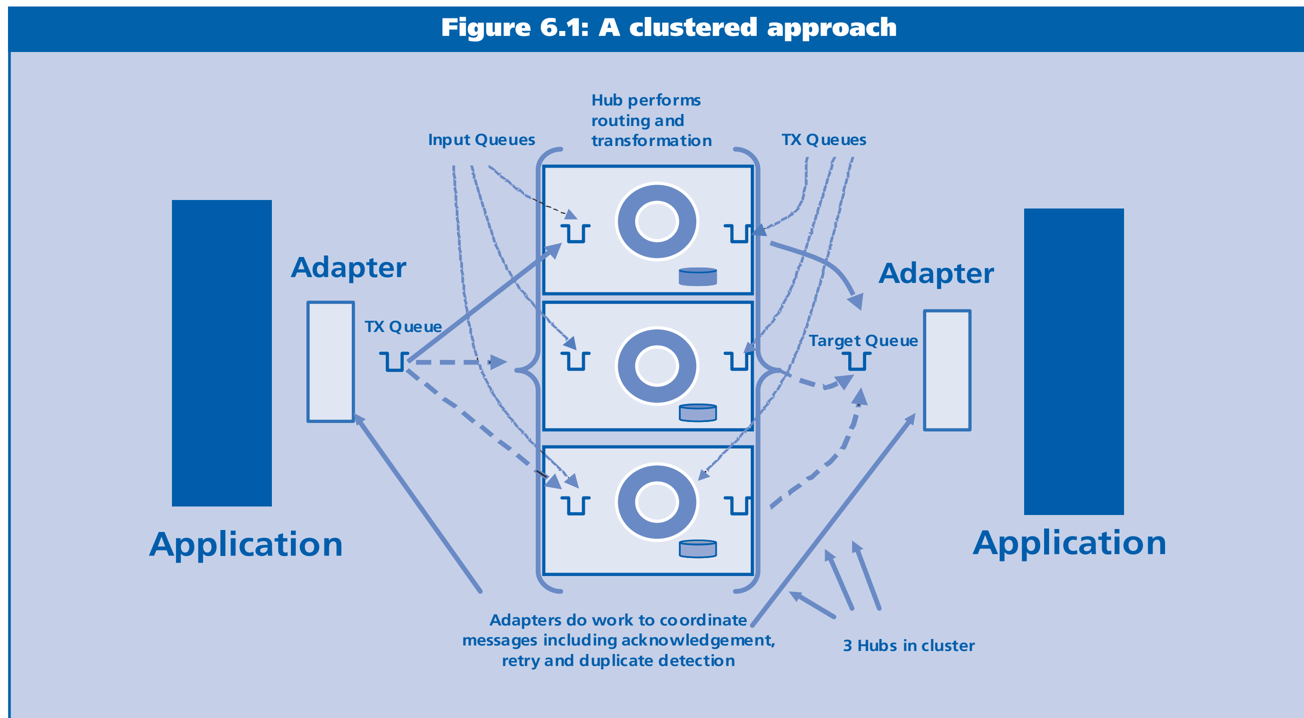
However, in a crisis following a failure, most want as simple as possible an architecture to manage. There is, though, a further complexity. Even if individual applications are HA protected, what happens in the event of a major failure of a system that requires putting the system into recovery mode? When a system fails and has to be recovered from a backup, there are several questions that must be answered:

- **what measures are there to recover and replay messages delivered to it after the backup was taken, the effects of which have been discarded when falling back to the backup?**
- **what mechanisms are available to ensure that any messages re-generated by the failed system — as a consequence of 'catching up' — are not re-transmitted (thereby generating double counting in the systems to which they have been dispatched)?**
- **are such regenerated messages identical to the previous instances of the messages?**
- **if not, is the net effect the same or have the communicating systems become logically inconsistent?**

The key point here is that if a system has to recover from a backup then it may be required to recover the messages injected into the system from the hub and replay them. But it must also be able to catch those messages generated by the system (to the hub) and stop these being re-transmitted and creating a double copy in the target system. Furthermore, there must be a confirmation facility to demonstrate that the new messages generated are identical to the original ones.

If these are not available, then the failure of one system may mean that the whole infrastructure must be shut down and 'good' systems disconnected, while the failed

Figure 6.1: A clustered approach



system is recovered using the hub. After that a reconciliation will be needed, and will have to be carried out across all participating systems. This is a major exercise, and is not a theoretical concern.

If the failure of an application does, therefore, require the complete system to be shutdown during recovery, you may well ask whether it is worth implementing a reliable hub. It is reasonable to ask this but there is a justification for continuing.

If all the applications of an enterprise utilize the hub, then it is important to be able to recover and restart the hub to enable applications to operate regardless of specific integrity issues for each application. In effect, you use HA measures to protect the hub, but you may need to expand the facilities within your infrastructure design if the process of recovering an application from a historic backup generates the types of issues identified above.

Options for reliable hub construction and deployment

The following, therefore, include the major choices for introducing high availability into hubs:

- **purchase of fault tolerant systems (if the messaging middleware is appropriately enabled)**
- **exploitation of a high availability architecture**
- **use of hardware-based cluster technology messaging middleware that is inherently reliable (guaranteed, once only, etc.)**
- **clustering and parallel brokers**
- **the use of messaging to provide reliability.**

It is, at the same time, vital that a management infrastructure is put in place to ensure that:

- **systems are monitored so that failures are detected and reported on, either on request or through alerting mechanisms**
- **mechanisms exist, with appropriate practices, to automate recovery operations, either automatically or when initiated by system managers.**

Each of the above has its various advantages, and disadvantages. These are discussed below.

Purchase of fault tolerant systems

There is the potential option of using full fault tolerant systems — like those from HP/Compaq's (Tandem) Himalaya product line. Such systems offer comprehensive redundancy, but only if the middleware can exploit such redundancy features. This is only practical if the middleware chosen takes advantage of the fault tolerant or non-stop capabilities. For example, WMQ was only introduced onto Tandem machines by a third party — Candle — developing this.

More often than not, most commercial middleware hubs do not exploit fault tolerance and so cannot, therefore, benefit from such fault tolerant, non-stop capabilities. Care needs to be taken here not make false assumptions or possess inappropriate expectations.

For example, while WMQ is supported on Himalaya, WMQI is not. Broadly speaking, fault tolerant systems and messaging hubs do not combine to provide HA solutions.

High availability solutions

A high availability architecture is one that enables the migration of a system automatically following a failure to a second system through the use of a machine in standby mode. There are two approaches:

The first is to use clustering. This requires the standby system to:

- **mount the drives from the failed machine**
- **start up all the required applications**
- **take over the IP address of the failed machine**
- **once running, continue (from the point that the failed machine was at when it stopped).**

This architecture can be enabled on Windows, Solaris and AIX. IBM's WMQ Support Packs detail a number of alternative configurations to achieve this objective.

In addition, the fact that the disks are dual mountable — but only one machine can mount them at any one time — means that when a machine fails, the other machine takes over control of all resources and continues operation (Figure 6.2). However, because there is only one copy of the disks, it is normal to locate the machines within the same site.

A second — more complex, and more expensive — approach is to have an advanced disk sub-system where all I/Os are queued in parallel to separate controllers in different locations connected by high speed networks. A combination of Veritas software and EMC equipment on Solaris can exploit such mechanisms and equivalent facilities are available on AIX. Once the controller caches are updated,

the I/O is guaranteed to be written to the disk without data loss.

Using this second approach enables separate disk sub-systems to be located in different sites, up to 20Km apart. This offers a robust distributed and replicated implementation for a hub although it should be noted that, with the Veritas/EMC solution, the replicated site:

- **operates in read only mode — and cannot be accessed or run**
- **becomes read/write enabled, taking normal control, only when the primary system fails over.**

Another downside is that, once the fail over has taken place, the failed system is not updated and becomes out of date. The resulting fail back operation is time consuming and non-trivial.

The key benefit of this second approach is that it provides a business continuity platform. However HA architectures mean that there is usually a loss of service for the period between:

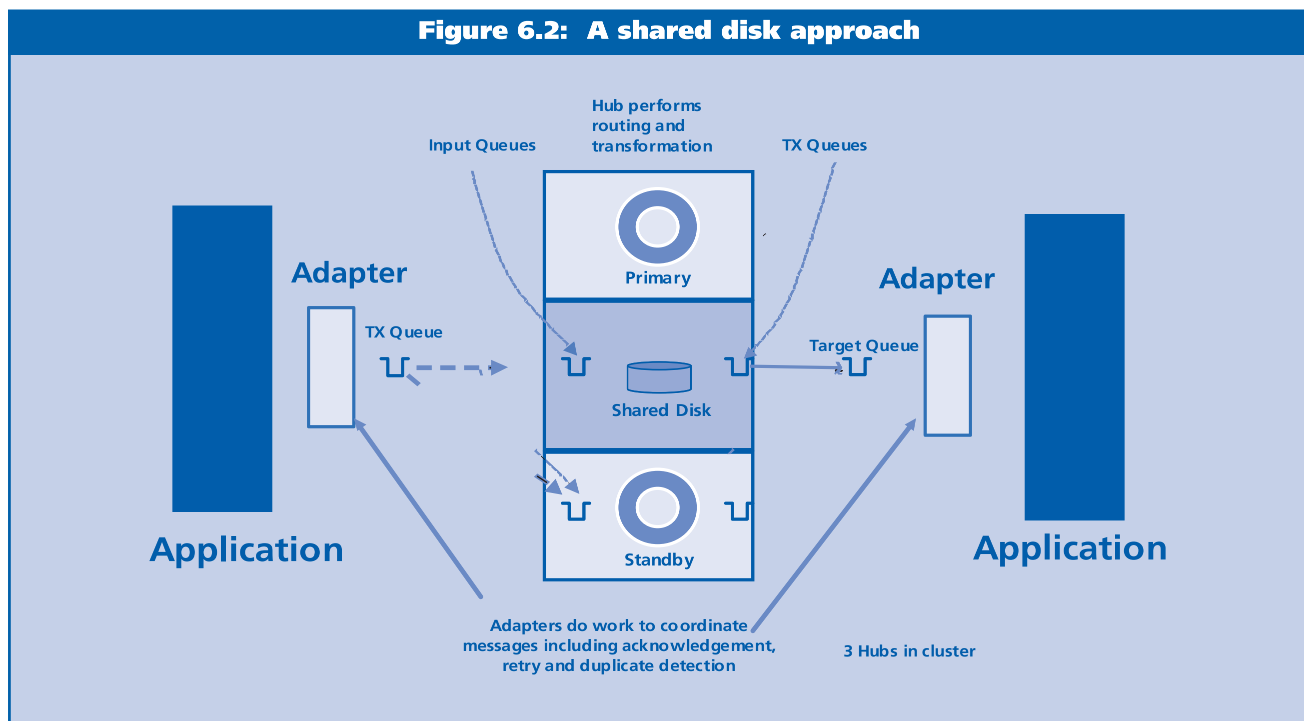
- **the failure occurring (and being detected)**
- **the fail-over successfully completing.**

With a messaging middleware architecture, this is generally not a significant issue provided fail-over is practised regularly. Not only is familiarity introduced but the time taken to fail-over to take place can be measured. This assumes that real time messages do not need to be passed through the hub (most message based systems will not fail or deny use to an end user if the hub is down). If, however, real time responses are required, a different alternative will be needed.

The principle downside to this HA approach is that its use of shared disks is only effective in the event of a hardware failure. HA does not protect against a software failure that cannot be resolved by a reboot. If something within the software configuration, or even data, becomes corrupt and the system cannot re-boot, then a serious problem will exist that will require the whole HA environment to be shut down and re-configured to resolve the problem.

This approach is available and supported now, even if it introduces delays while fail-over takes place. Provided no applications connect directly to the queue manager — but via intermediate queue managers — then applications are unaffected and will continue once messages start to flow. However, the distance between, or separation of, systems can present geographical limitations. It may not be possible to link (say) London and New York — because they are too far apart (physically) for this form of HA.

Figure 6.2: A shared disk approach



Designing reliable systems using hardware and operating systems is now common practice, not least because it has been proven in RDBMS installations. A range of hardware options exist with differing levels of reliability. These include:

- **redundant hardware in given systems (like a hub) to enable a single machine to continue in operation without interruption**
- **mirrored disks, to protect against disk failure**
- **hot swap facilities, to enable failed disks to be replaced and rebuilt without bringing the system down**
- **dual I/O channels to each disk so that I/O will continue in the event of a controller failing.**

Historically such technology was expensive. But it can now be found in relatively modest Intel-based servers from the likes of IBM, HP/Compaq and Dell.

Indeed, one can go further. Generally systems will carry on following failure of one of several CPUs, though a reboot may be required. Systems have now become increasingly reliable and such reliability may be sufficient for most organizations. The problem, that needs addressing, arises if a key component of a hub system fails which makes the machine fail.

In this context, it is necessary to consider the options to ensure that, by whatever mechanism, the service can:

- **either continue uninterrupted**
- **or be restored quickly.**

Figure 6.3: Reliable hardware

Hardware clustering

The discussion below relates to an approach that is currently not possible with WMQ.

For machines in a cluster to operate concurrently on the same resource it is essential that there is shared access to the I/O sub-system. Currently this is only available on the older OpenVMS OS or on UNIX when I/O sub-systems are not mounted.

On UNIX, concurrent activity is possible but only if the software — which enables shared access — can store its data in a raw partition and has a distributed lock manager that can control access across the systems. Such a capability is rare. It is is only currently known to be possible with Oracle's RDBMS.

The problem for other approaches, including middleware like WMQI or WMQ, is that they use files located on the UNIX file system. Therefore hardware cluster technology is not appropriate at the current time. This comes down to the twin needs to support:

- **raw partitions on shared disks and WMQ objects available within a raw partition**
- **a distributed lock manager.**

In concept, this architecture (if available) would present two distinct IP addresses, as both queue managers would be running concurrently within an WMQ cluster. This makes use of a mechanism similar to a prioritized cluster workload exit where messages would:

- **be dispatched to a priority queue manager**
- **fail-over to a stand by queue manager.**

This is conceptually attractive. Its advantage is that there is no loss of service as there is no requirement to fail over and re-start. Unfortunately, it is generally not available.

Software clustering and parallel brokers

Another approach is to architect multiple routing though a hub by enabling messages to enter the hub through a cluster (like that provided in WMQ). In such a cluster, multiple queue managers can be set up and duplicate queues placed on each cluster. Messages are then distributed to these various queues by the cluster, using load balancing techniques.

The difficulty with this approach is that, once a message has been dispatched and the appropriate queue manager selected, the message is stuck until that 'system' can be returned to service in sequence in the event of a failure of any of:

- **the queue manager**
- **the hub broker**
- **the host machine itself.**

If a hub broker fails, but (say) the queue manager is still operating, messages can be routed to the queue manager but cannot be processed. Under these circumstances it is essential to prevent messages being passed to the queue manager. This is possible by:

- **setting the queue to disable message 'puts' (if this can be done online when the queue is currently open)**
- **shutting down the queue manager.**

The important point here is that careful design of the messaging architecture is required. This must consider each failure scenario in detail. Once documented, recovery must then be practised. This does, though, become a viable option if there is reliability in the applications — as initially discussed.

When an WMQ cluster is used, care is required to ensure that there are no issues associated with message ordering. If there are multiple machines supporting queue managers that are formed into a WMQ cluster, and the power of these machines is not the same, then it may be that messages will be routed through them at different rates. Indeed this may be true in any circumstance if for some reason any one of the machines becomes more heavily loaded that other machines in the system (as found when a cluster queue has two instances on separate machines).

Though the order of messages through WMQ is not guaranteed, in most cases messages are processed in order and testing on a single server machine may not show an implicit dependency on WMQ to deliver messages in order. Only when a system is migrated to a software cluster, and a particular scenario occurs, might the failure to deliver messages in order be identified.

This approach is available. It is a simple solution applicable to circumstances where applications handle message reliability — and where middleware is appropriately enabled.

Use of messaging to provide reliability

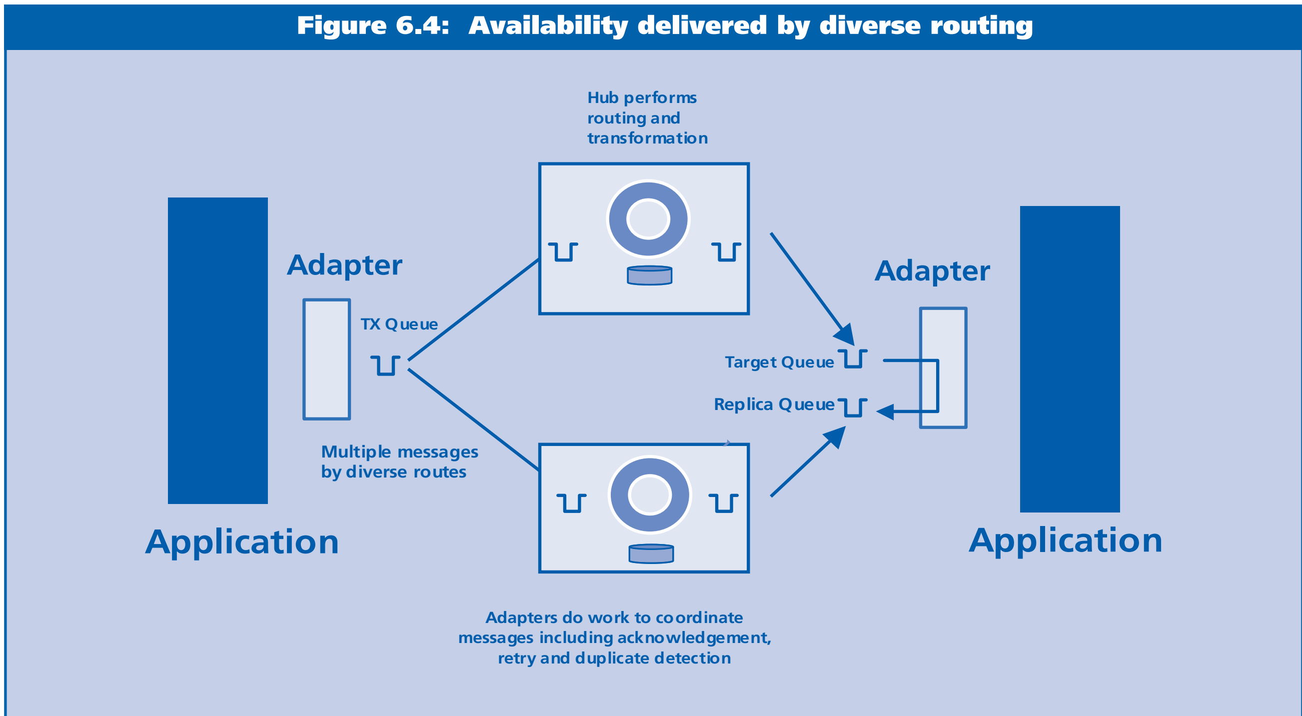
Here we mean the generation of two messages by the originating system. The messages are then processed in parallel along independent routes and only the message from the primary path is used when it arrives (or else the first one received is processed and then the spare is detected and discarded). However, in the event of a failure, the system can select messages from the alternate route and continue with subsequent purges of the message copies held on the failed system when the latter has been recovered.

This can be illustrated in the context of WMQ. With WMQ you can generate multiple messages and then route these messages in parallel by using:

- **name lists (as a defined object to enqueue a message)**
- **parallel MQPUTs (to enqueue messages to multiple queues at the same time).**

In WMQ Version 6.3 the ability exists to be able to implement message exits, in which MQPUTs may be used to generate duplicate messages and achieve the same objective. A parallel message acting as a backup can then be

Figure 6.4: Availability delivered by diverse routing



processed from start to end via a completely separate route. When the master is finally de-queued, the backup can be removed. However if the master never arrives a standby copy is available.

Another dimension to WMQ is the possibility of using a re-try mechanism. Within WMQ it is possible to cause messages to expire and to generate a report message when the expiry takes place so that the failure can be reported back to the source. This mechanism can be used to report back to an application that a message has not reached its destination so that a re-transmission by an alternative route should be tried. Further, a report can be generated, to an exceptions handler, about any failure to deliver a message.

The disadvantage of such an approach is that it requires more complex middleware (and data) structures to be deployed at run time. Nevertheless, this approach is worth trying with WMQ 5.3 (although it will almost certainly affect the detailed system design). It would also be complex to re-engineer into existing applications.

Recommendations

Until inherent HA facilities are available in middleware hubs, the following are recommended for designing an HA approach:

- **undertake a rigorous middleware design that identifies the requirements for performance (in terms of both latency and throughput)**
- **assess the significance of message ordering**
- **analyze the implications of, and costs of, losing or duplicating messages**
- **understand, document and practise HA operations**
- **introduce thorough auditing and logging.**

Where there is flexibility to do so:

- **implement a non-HA architecture based on a logically distributed hub; the failure of any one hub results in work automatically being transferred to the other hubs**
- **introduce appropriate measures in the applications themselves, or in the adapter(s), to enforce order**
- **distribute read-only repositories, especially one for each hub**
- **exploit a protected, updatable database that is replicated to a standby node using one of the advanced hardware techniques described earlier**
- **adopt protection mechanisms, like those described above.**

The very act of enhancing reliability (or performance) changes many previously valid assumptions. Principle amongst these is that, in a highly available hub, it is no longer possible to guarantee delivery in the same order as the messages were originally queued or presented to the hub. Therefore it is vital to:

- **identify specific requirements for message order within a highly available hub**
- **take deliberate measures to ensure that message order is managed at the appropriate point.**

That said, the ability to provide a high availability hub within system software without impact on the overall application programmer can be enabled if a multi-level approach is adopted. This would introduce:

- **message replication at one level in the architecture — to ensure that there are always two copies of any message in the system**
- **a mechanism to ensure that such duplications are automatically detected and removed**
- **a second mechanism to ensure that any side effects (such as processing of messages via transformations or rules) do not generate double counting of messages within any underlying database or log.**

However by increasing the responsibility of the application or adapter programmer this logic can be implemented outside the hub, probably by implementing a more robust, reliable and highly performing solution overall.

Do you really need HA?

The final decision here lies in the requirements. Business continuity requires an appropriate level of protection against:

- **the impact of software and hardware failures**
- **loss of the site and/or denial of access to the site.**

With respect to loss of site, this used to be focused on two major threats — namely flooding and explosions — each of which are local and can be protected from an alternate site within four miles. The game has now changed, after September 11th, 2001. Now business continuity planning must also cater for loss of an entire city — New York must

be replicated in London or Tokyo — and this imposes a new set of technical challenges.

Thus far, this analysis has emphasized the importance of considering high availability at the design stage. But it is equally possible that an HA hub is not needed at all, or that any expenditure should focus on improving the reliability of applications. This is often superior — but only if availability issues are incorporated into applications early enough in the design.

Netted out, the features of any middleware hub impinge on the operation of messages across a middleware infrastructure. The key needs are to determine:

- **whether message order matters**
- **if it does, how to deliver messages, in order, across the middleware infrastructure.**

Order can only be maintained by ensuring each flow only has a single thread. Messages can still be delayed or delivered out of order if a system goes down or a message 'gets stuck' in a queue — but typically we treat this as something more like a bug.

The moment, however, a messaging hub is enhanced to implement either performance features such as multithreading or high availability through the use of an WMQ cluster and multiple instances of WMQI enabling multiple paths (or both), simplicity disappears.

Management conclusion

As Mr. Denning describes, implementing a reliable, high availability hub architecture that is resistant to failure is currently a non-trivial exercise. However messaging hubs are being introduced by many organizations, all too often without sufficient consideration of the complex interactions that hubs embrace.

That said, Mr. Denning makes the point that provision of an HA hub is vital when a hub supports large numbers of applications and the failure of the hub would interrupt the service to those applications. You should not try to escape looking at HA and its different implications.

Members of the International Advisory Board

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

William Donner
Fenway Partners, Inc.

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

Ellen M. Hancock

Paul Hessinger
Vision UnlimITed

Pierre Hessler
Deputy General Manager,
Cap Gemini

Michael Killen
President, Killen & Associates, Inc.

Dale Kutnick
President, Meta Group, Inc.

Thomas Curran
Chief Technology Officer and EVP,
Bertelsmann Media Worldwide

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

Fiona A. Winn
Managing Editor & Publisher
Spectrum Reports

Additional contributors include:

Francis X. Dzubeck
Communications Network
Architects, Inc.

Jay H. Lang
Distributed Computing Professionals

Keith Jones
IBM

David McGoveran
Alternative Technologies

Will. Capelli
Giga Group

Amy Wohl
Wohl Associates

Martin Healey
Technology Concepts Limited

Mark Allcock
J.P. Morgan Asset management

Aurel Kleinerman
MITEM

Chris Cotton
Consultant

Ian Hugo
Year 2000 Taskforce

Yefim Natis
Gartner Group

Rosemary Rock-Evans
Consultant

Beth Gold-Bernstein
Hurwitz Group

Tom Heywood
University of Southampton

Eric Leach
ELM

Glen Macko & John Parodi
Digital Equipment Corporation

Randy Rhodes & Troy Terrell
Black & Veatch

Colin Osborne
The Tivyside Group

Roy Schulte
Gartner Group

Jim Johnson
Standish Group

Tom Curran
TC Management

Alfred Spector
IBM Corporation

Max Dolgicer
International Systems Group, Inc.

Peter Bye
Unisys Systems and Technology

Ely Eshel
MINT Communication Systems

Steve Ross-Talbot
SpiritSoft

Peter Houston
Microsoft Corporation

Jeff Tash
Database Decisions

Ed Cobb
BEA Systems

Bernard Abramson
Merck & Co.

Mirion Bearman and Kerry Raymond
CRC for Distributed Systems
Technology

Geoff. Norman
Xephon

Jim Gray
Microsoft Research

Jason Longo
PRL Scotland

Wayne Duquaine
Grandview DB/DC Systems

Steve Craggs
Saint Consulting

Tom Welsh
Consultant

Gustavo Alonso
Swiss Federal Inst. of Technology

Mark Whitney
Delta Technologies

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

USA and Canada:
Spectrum Reports, Inc.

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

UK and Rest of the World:
Spectrum Reports Limited

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

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

Email and Internet

Email:
spectrum@middlewarespectra.com

World Wide Web:
www.middlewarespectra.com

ISSN 1460-7220