MIDDLEWARESPECTRA

incorporating FINANCIAL MIDDLEWARESPECTRA

Contents

May 2003

Volume 17 Report 2

Evolving the EAI industry for the benefit of customers

Michael Kuhbock
Chairman, EAI Industry Consortium

Management introduction

Michael Kuhbock is Chairman of the Enterprise Application Integration Industry Consortium (EAIIC — which can be found at www.eaiindustry.org), which is based in Calgary, Canada but has a global and expanding membership. Although Mr. Kuhbock has been working with application integration for several years, he is not a long time participant. Indeed he set up the EAIIC as an 'outsider' who saw that the EAI market place was fragmented and inconsistent, particularly from a user's or customer's view-point.

Over the past two years he has been in a unique position to hear globally about what has been going on in the integration market place and what are the real concerns of people. In this interview Mr. Kuhbock discusses how and why the EAIIC was created. He describes what it is trying to achieve and why its middleware-related activities matter — to customers and vendors alike.

Setting the scene

My background is essentially entrepreneurial and I have worked in many different sectors — including for a major multinational organization. In the past ten years I have been increasingly involved with technology but I do not regard myself as a technologist — certainly not compared to people who have been in the software industry for decades longer than me.

I first encountered the application integration market place about three and a half years ago and I came to it with, I think, rather a different perspective — maybe a newer, brighter eye — most probably because I had not previously been immersed in it. In the past three years I have found myself undertaking a tremendous amount of research on my own, including starting the Consortium because nothing like it existed (which I found unusual).

How did I start to become interested in the first place? In essence, by chance. I was undertaking a business development, strategic alignment, consulting engagement with a company that was located here in Western Canada. This company was a professional services provider and some of its services related to the delivery of integration to user organizations. Indeed, I became so interested that I ended up taking over the company, Groundswell Revolution, where I am now the CEO. However, my interest in this was, and is, primarily from the business angle rather than any technological one.

Nevertheless, in the integration space you cannot avoid technology, even if you do come to it with a different perspective. I did my research and realized that there was no global advocacy group or consortium or association which existed to represent the interests of customers and users — or even vendors.

Integration, without being integrated

Yet I felt that there was strong need for business to possess this. One of my insights was that the vendor industry was saying that its activities were all about integration. Yet it was singularly failing to offer much integration amongst its own offerings. The reality is that application integration provision has been unacceptably disparate.

From my perspective, possibly the biggest challenge the integration industry faces is inconsistency. This may come from a technology perspective or an end user one, or it may come from the different verticals focused on by vendors or from the many different industry specialities that exist. Irrespective, integration should be about consistency, not inconsistency. Add to this the fact that there was no common interest grouping and it was not surprising that there was little incentive for vendors to try to create some form of coherency or consistency.

Why was this the case? In my view, this was a legacy from the speed and enthusiasm of the 1990s. Each and every vendor then felt obliged to create unique technologies that had to be marketed in differentiated ways — often using the same language. The result was multiple, conflicting marketing angles and messages with the emphasis on being seen to have the newest 'flavor of the day' in terms of technology and solution.

But this caused massive confusion in the market place. Users could not distinguish one approach or product from another. They, not unreasonably, became confused. With confusion came an unwillingness to invest in products and services, because they found it so difficult to work out what they were being asked to buy. Add this to the impact of the high tech. down turn and it should not be surprising that the application integration industry has encountered lean times.

The irony, however, is that the need for integration has in no way disappeared. If anything the need is stronger and greater than ever before. But the EAI industry had all but killed the golden goose in its pursuit of uniqueness rather than results. It forgot what its customers were trying to do and why they (customers) were needed to buy.

The legacy we now face is in establishing a consensus that is relevant to users — one which users can recognize. In my discussions with end users, this is one of the biggest concerns they continue to have.

The state of the integration market today

From a business perspective I find that there is a tremendous need for the services and solutions that the integration technology market can provide. But there is a hesitancy, a fear, about making progress. That fear comes directly from the legacy I have just discussed. It is now allied to a different fear — about which integration vendors will survive until the economic up turn.

Let me put this in a different way. Users are wary of the pure vending model of the past. This was the 'technology approach' — 'if we build it, you will buy it' or 'if we build it, customers will emerge'. The brutal truth, as seen over the past two years, is that they are not volunteering anymore — and certainly they are not buying in the ways they used to buy. In consequence most integration technology com-

panies have been forced to take a step back and ask themselves why customers are not buying what they (the vendors) are building — even when this is allegedly the latest and the greatest.

The challenge that most end users have is the tremendous amount of application integration software (of one form or another) which is either being used only a little or not at all. Shelf-ware is common. Much is disparate, even within an organization or a division of that organization.

Integration vendors are, therefore, in despair. They are searching for ways to leverage what they have built, to integrate what they currently have and then effectively utilize this to solve real business problems. Gradually they (the vendors) are understanding that possessing the newest, latest and greatest technology available is not a differentiator that customers value.

Instead a change is occurring. The more courageous of vendors are now starting to worry:

- **less about the building the best and fastest technology to sell**
- **more about engaging with customers on a continuing basis.**

Those adopting the latter approach are:

- **researching the market place**
- **finding out exactly what are customers' needs**
- **only then trying to deliver something which addresses those specific needs.**

This is one of the areas where the EAIIC has already made progress. We have facilitated the combination of end users, vendors, universities and other academic institutions. The EAIIC provides a meeting place where anyone can influence the direction that application integration technology should take. It enables common business drivers to be identified and seen to be what customers want. In so doing, I believe this will facilitate the development of relevant enterprise application integration technology for the future.

Shelf-ware

In all this, shelf-ware turns out to be a real problem, even though organizations have already sunk the cost. Prima facie, one might think that there would be nothing to prevent customers from using all the integration solutions that they had already purchased. After all they have already spent the money.

Yet I know of companies here in Western Canada, and in the US, that have bought CRM and ERP and integration software who are not using what they purchased. Indeed, many in their organizations still find it easier to work by using spreadsheets and hard copy. Their people 'prefer' to type out a spreadsheet and then fax this to another office rather than using the power of the integration and application technologies in which their organizations have already invested.

This mindset is a colossal challenge. A complementary fear is that, although these organizations have tremendous amounts of software, much of this does not fit into an overall enterprise solution. The result is disuse. Integration and application software just sits there, and no one seems able to find a way to utilize that software efficiently (or even its associated hardware).

Yet another factor comes into play. Many of these organizations have also invested in a path towards a full enterprise solution. But much of the purchased software turns out not to facilitate this solution, which makes use even harder.

The result is that users are sitting back going, 'OK, now we are hearing the same story again and we do have the need ... But this time we are going to wait until all the vendors who are going to be involved get on the same page'. But the vendors do not seem able to do this.

One of the best analogies I have come up with to describe this (and it seems to be well received as a metaphor) is to liken:

- **the vendors to a group of people on a bullet train**
- **customers to a group of people on a station platform.**

The vendors are standing by with their latest, shiniest, brightest technology. But the train does not bother to slow down long enough for those who might want to listen to what the end users want, or even to let the end users onto the train. The result is obvious.

Change is occurring

But change is happening. I find that vendors are increasingly trying to humanize some of their marketing efforts. I have even heard some of them starting to talk about the business drivers that affect customers. Nevertheless, too many are still using empty or opaque and wholly incomprehensible acronyms.

Some of the more advanced are beginning to think in terms of Total Cost of Ownership (TCO) and even of Return on Investment (RoI). But I still question whether there is yet really a deliverable behind the interest in this. I do fear that some of the use of these terms is just the freshening up of old marketing pitches, with these being warmed over with a dash of added 'RoI' or 'TCO'.

Yet what I also meet are plenty of customer organizations with problems to be solved. They have business issues to resolve. The vendors who are succeeding are those that go in consultatively and focus on the specific customer problem rather than on the technology that they may possess (or some other vendor possesses).

The good news here is that the focus is changing. The main thrust has become the solving of the business problem, not selling specific technology. This is what the customer wants — his or her problem solved.

There is nothing magical here; only common sense. I know from my experience in other industries that sales are made when vendors base their offerings on a driver and a need that comes either from within the business or which has been identified by the market place. But integration technology (arguably most technology) has never really lived under this discipline. It has always been 'too exciting' to 'have to' apply such common sense.

Think about it. When I upgrade Microsoft Office I only use about 5-10% of its total capabilities. I am paying for 90%+ that I do not want or need. The software industry cannot sustain this model, in my view, because its customers cannot afford it. Focus on the business issues, and not the technology, is really going to be what differentiates products and approaches in the application integration space.

Let me put this in a different way. I recently came back from the IBM Transaction and Messaging conference in Las Vegas. One of my roles there was to explain the EAIIC to attendees in a birds-of-a-feather session. I explained to everybody in the room what the Consortium was about and the challenges I have described above.

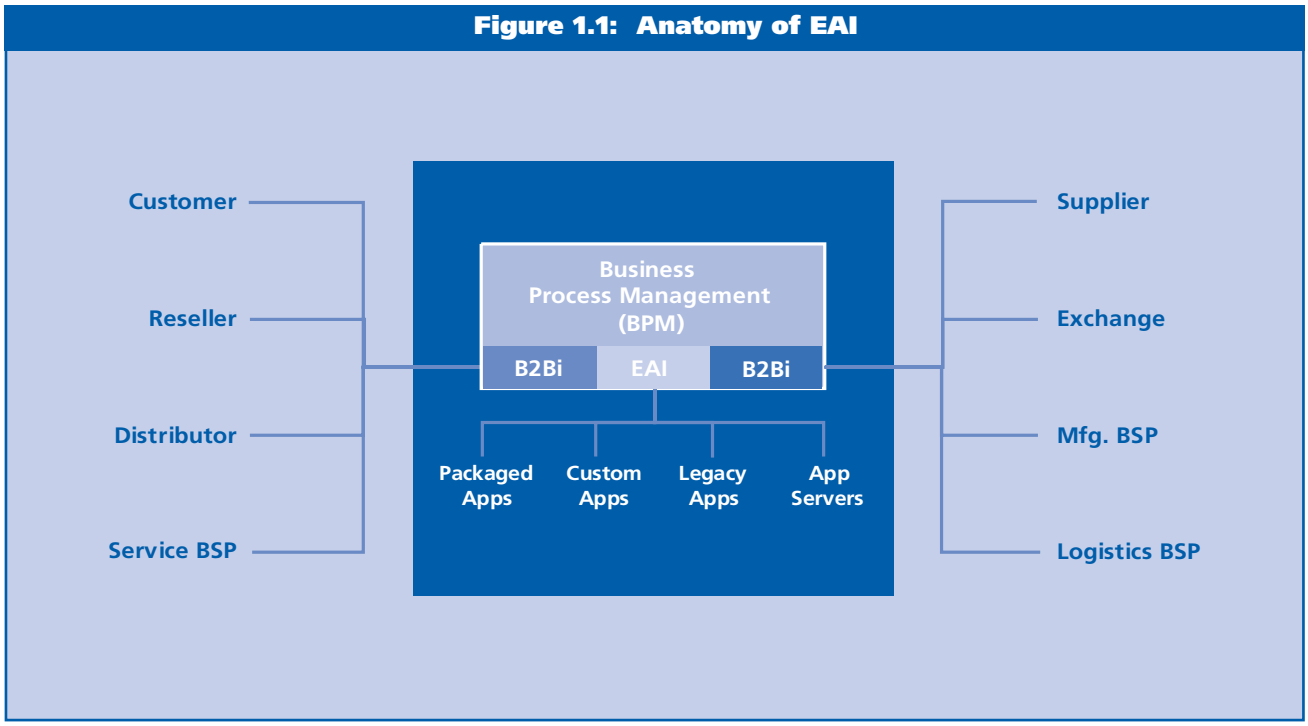
In particular I emphasized the lack of communication between integration technology and business. I said that I still did not believe they were on the same page. Technology players still seem unable to play together in the same sandbox as business players and it is this that causes problems for end users.

Two silos

I went further. I commented on how, in most organizations, there are two silos of people:

- **the business people (end users)**

Figure 1.1: Anatomy of EAI



- **the IT people.**

The end users are a challenge to IT people because they want to talk about the business problem. The IT people are a challenge for end users because they want to talk technology. In effect you have two vertical groups in the same business who seem unable to understand each other. They are challenged in terms of communicating with each other. Business has the need and technology has the solution — but neither realizes, or can comprehend, what the other possesses.

Having said my piece, I had two gentlemen stand up and confirm this. One said that his job was to 'do business analysis to support software acquisition and to build a budget'. He said that he had a doctorate (in technology) but no MBA. He did not understand what users meant, or how to satisfy their requirements. He also said that he felt unable to deliver a message to business users in terms they would understand. 'I don't know how to deliver the message. I know we need the software and I know we need to integrate but I have trouble communicating with our business users. Then my business users stand up and say the exact same thing, in reverse (I do not understand all this techie gibberish and what it does) even though I have business challenges that need to be addressed.'

In the EAIIC we are trying to simplify integration so that the technologists and the end users can find some common ground. If we can persuade them to communicate, they will move forwards with integration. If they cannot, they will not. In this context, vendors should be able to market on a basis that serves the of solving of business integration problems for end users.

The challenge for vendors

The key challenges for vendors are to:

- **'humanize' enterprise application integration**
- **orient the delivery of related-technology to what the market needs in business.**

Vendors have to engage customers on a deeper basis. They have to talk to business units. They have to describe a holistic solution which engages both the business and technology people. Vendors who do not do this, in my estimation, will almost guarantee their failure — however 'perfect' their technology.

They cannot afford to permit solutions to remain disparate. Without full integration there is no solution.

I have seen this with current clients. I receive phone calls probably every couple of days from end users who simply do not want to speak to integration vendors anymore, because they feel that the latter are not listening to what their needs are. In my experience such sales people are almost always trying to sell something that they have (in terms of a solution) before they even ask: 'what do you need and why?'

The key is going to be a greater focus on service and less on product. In most industries that have faced real market challenges in the past 15 years, although I cannot speak for Europe because I personally have not done a great deal of business there as yet (although, via the EAIIC, I am now listening to its needs), the answer has almost always turned out to be 'listen to what the business needs really are'. Look at manufacturing, the produce industry, insurance, real estate, the auto industry, professional services, etc. All have been through bad times. They recovered when they understood what the customer wanted. In effect they developed a consultative model to deliver what their products should be.

Technology has still to wake up. Enterprise application is no exception. It is still building product and selling on the basis that 'OK, we built it, which is why you are going to buy it'.

What the technology industry, and this is wider than just integration, is finally realizing is that the end user, or customer, will not buy like this. That is why revenues are down. That is why financial positions are weak.

The consultative model is needed. It is obvious. It has to be done, or vendors will go out of business.

To me this is going to be driven by the IBMs and the Microsofts of the world. While smaller vendors are starting to do it, it is the few majors who matter — because they end up dictating which direction the technology/business market place will go for the future.

In my own business I am seeing this. It is welcome but the ship is large and it will take time to turn around. This is what we hope the EAIIC will facilitate. It now has a platform and experience of the market place. Users and vendors are coming to us for guidance and understanding — so that they can help themselves.

In effect, we provide a buffet so that potential end users, looking for a solution, can come and investigate — without meaningless acronyms and other gibberish being thrown at them. They can then pick out what they want.

At the same time we try to humanize integration, by adopting a strong business-centric focus which concentrates on understanding the business problem before reaching out for some possible deliverable. This approach gives potential customers the opportunity to understand before they go out into the market place to look for a solution.

Looking two years out

If I look out a couple of years I hope we will see an increasing humanization of the language of integration. This will enable so much. For example, in most enterprises it is now a collaborative decision about whether to acquire an enterprise application integration solution. If all sides do not understand, even if the need is recognized, the purchase will not happen.

CEOs and CFOs are increasingly making integration decisions. Yes, the CIO prepares the budget and then operates what is bought. But increasingly the checks are signed off by operations, by the businesses.

So I am starting to see organizations within enterprises engaging together and starting to communicate. It is painful. They have rarely had to do it before, but they are starting to do this.

Business and technology people have to be on the same page if they wish to move forward. In the next two years, I am optimistic that this will gain ground and will become commonplace.

Similarly, the communication between vendors and end users will become simpler. The solutions that sell will be based on needs and requirements as expressed by end users.

If this sounds simple, it will not be. Disposing of the 'I build it, you buy it' syndrome is going to take time. Equally, vendors will have to appreciate that, even in two years, there will be lots of users who are still dealing with the purchases of two or three years ago (2000 and 2001). These users will be saying 'help me to take two or three steps back and create a solution with what I already have. Once that is complete, then I can move forward.'

At the same time, I hope that the next two years will see users unifying, and vendors responding, by starting to cooperate with other vendors in the 'same sandbox'. This will be quite different to the old ways of positioning or marketing — where, all too often, the creation of a new software language or changing an older one seemed the preferable marketing way forward.

If the end user has to say to himself or herself 'well, Microsoft has got X, IBM has Y and Sun has Z — and I do not understand', then no purchase will happen. A good example is Web Services. These have to work together or there is no point, from a user perspective, in buying into them. Web Services, as middleware, promise much good for integration. But they are not there yet from a user perspective.

The EAIIC

The EAIIC was founded in early 2001. It has grown organically. It is not sponsored by one of the major vendors.

The first year we spent building infrastructure, Web sites and content. We refined the mission and direction as well as produced a number of studies which looked at what the market sought. (The EAIIC is no different from any other vendor — we had to listen to what the needs and challenges were and then try to deliver a platform that facilitates solutions.)

We are up to 35+ confirmed members with about 100 that have engaged us and want to join. Currently about 30% are end users (the rest are vendors) but I expect the end user proportion to keep rising. Deliberately, we have done no real advertising. Instead people are finding us.

One reason is the clarity we offer. Try visiting a major vendor Web site to find out about integration and after half an hour you might have something, if only you could understand that vendor's product line and naming conventions. Try us and the experience is simple and straightforward. With the consortium, in one click, you are pulling off information on EAI or application integration.

Our attraction, we are finding, is that people are looking for answers and information, whether it be in white papers or in definitions, whether it is just a summary or a detailed explanation. Although EAI has been around for a decade, the vast majority of possible users are only just starting to learn what integration is about, and what solutions already exist.

An important point of principle for us is that we have a strong component of academics and students as well as professional services people. This gives us depth and consistency, even though we try to preserve a strong business-centric focus. We also work with other standard bodies for we do not want to compete with them in their work. We do not want duplication of effort.

Business Value Realization

One example of the way we think is the recent creation of our BVR Committee. This is the Business Value Realization Committee. It focuses on bridging the gap between the business and IT silos. Its first document describes Business Value Realization for Integration 101. It, in simple terms, explains:

- **what integration is all about**
- **where integration technology comes from**
- **where integration is going.**

The intent is that it will assist an end user to understand. This may be:

- **either the technology department printing it off and handing it over to business users**
- **or business managers finding it and discovering what integration can do.**

Our documents marry technology and business drivers. Business people can read them and comprehend what integration technology might do for them.

It really is this simple. The irony to my mind is that the piece of the integration pie we have been fighting over is a very small part of a huge overall integration pie. But we have been fighting over only a slice, not the whole.

The EAIIC aims to change this so that everybody benefits. If end users obtain integration solutions with which they feel comfortable, they will move forward. With that comes an invigorated and more complete market for all facilitators of integration solutions.

That is what we are driving towards. It is being rather well received — because we do have that business focus in our minds rather than the technology components.

EAIIC successes

The EAIIC web site is maturing into an integration solution center. Activity and visits have doubled every month since December of 2002 through to March 2003, the majority from end users of integration technologies or enterprises that are investigating integration solutions.

We are now receiving up to 27,000 hits per day from over 57 countries per month. This is just the tip of the iceberg of those companies interested and in need of integration solutions, which we will be assisting in their integration journey. This has included an explosion of interest out of Europe.

In addition, the EAIIC has successfully created strong collaborations between members and working partnerships which have arisen from EAI Industry Consortium participation. In the same vein, the EAIIC is creating solution opportunities via our thought leadership and educational initiatives. End users have contacted the EAIIC for direction and further education around their integration and business challenges.

In testing financial times, the EAIIC has enabled members to leverage their financial budgets, marketing dollars and intellectual capital by way of the members contributing intellectual capital and participation in conferences and events around the world. The EAIIC is already sponsoring and supporting over 25 conferences in 2003 and has been invited to speak or present at 20 events.

In my view, and I am necessarily somewhat prejudiced, the EAIIC has been able to establish a firm foundation for business-centric thought leaders to deliver their message into the marketplace. We are now starting to break down some of the barriers to secure the involvement and participation of the end user community.

Lessons learned

The most important lesson I have learned is that integration must be in tune with every business' bottom line. We are not there yet. EAI is still missing the mark. For example I still, in 2003, see white papers that fail to realize this.

We need to start communicating with everyone who might use integration technology, but in terms of business drivers. I have spoken about humanizing the language of integration. We need to engage end users in a more holistic methodology if they are to obtain the solutions their businesses need.

We are not there and this remains a challenge. If the EAIIC can produce a methodology and a solution and a thought process that matches this, we will have learned this lesson.

At the same time we really need to start opening lines of communication and understanding that not everybody is the same. Different businesses have quite different needs, even within an industry or niche. Vendors do not like this. But I have learned the hard way how true it is when applied to enterprise application integration. There are just so many variables.

Another lesson learned is that, if people do not understand, they will not buy. In my view they are quite correct. If integration is to succeed it must be understood. Without this why would any CEO spend hard won resources on a promise, on a desire?

We have to make sure that everybody understands the message clearly. Think back to school: very few people would raise their hands and ask a question — but 90% of a class often did not understand what the professor was speaking about. That is something that, in my experience in this market, rings true everywhere around the world. We need to start listening to the questions that are asked.

My next lesson is that we need to slow down as an industry. We are on that bullet train and we are passing the end users on the platform. We are not slowing down sufficiently to give them a chance to climb on board. Indeed, I would go further. The bullet train is on a circular track and going round ever faster. The customer sees the same train time and time again but can never get aboard. That is futile.

Following on from this, we must concentrate on consistency. We need it. But look at the evolution of EAI, Enterprise Application Integration. Three four years ago EAI was becoming prolific. People were using it, but then the market shifted from EAI to Enterprise Integration to Application Integration to Web Services. We must leave behind mindsets which confuse customers.

My last lesson learned is that we need to unify the message in the market place. Forget trying to differentiate by what form of Web Services you offer. That only distresses buyers, who thought there was only one form in the first place. Working collaboratively together, and making the pie bigger, is the way forward. I know this is not easy. But if nobody buys because they are confused, nobody wins.

Management conclusion

Mr. Kuhbock makes the point that integration is not an aspect of technology that is diminishing or about to go away. Rather he describes how the problem is more about expanding perceptions so that the true size of what needs to be done can be seen.

Yet, at the same time, he castigates the vendor side of the integration community — for failing to understand or even offer real integration. More than anything he feels this is what has been holding back customers and their organizations. It was to address this, and related issues, that the EAIIC was created. Its role is to simplify so that integration can take up the burden of providing automation between applications — the whole objective of middleware in the first place.

The economics of Grid computing and Web Services

Dr. Robert B. Cohen
President
Cohen Communications Group
and Adjunct Fellow, Economic Strategy Institute

Management introduction

Grid Computing and Web Services are likely to change the economics of commercial firms in a number of ways. They can reduce operational costs, due to labor savings, but they also reduce costs by bringing products to market faster and providing a means to outsource a larger part of production. The end result is an increase in both labor productivity and firm productivity. One consequence is likely to be higher levels of output and lower prices for products, due to the cost savings.

Grid computing and Web Services will also facilitate the overhaul of business processes and, in later use, provide a way to link partners and suppliers more closely to larger firms. This will facilitate product development and improve efficiency further.

In this analysis, economist Robert Cohen looks at how, besides the cost savings, Grid computing and Web Services will produce higher levels of output, lower prices and greater productivity for the industries that are early adopters. For the US economy, these two technologies will:

- ***increase GDP (by 3% or about $400B in 2010)***
- ***raise disposable income, personal consumption, private sector productivity and employment.***

In sum, Grid and related technologies stand ready to spur a new era of growth due to their impact on both productivity and profitability. As these new technologies become

more central to business operations, they will establish middleware innovations as a major factor shaping commercial growth and profitability, thereby changing the factors responsible for broader business success and economic growth.

Firm level impacts of Grid computing

The impact of Grid computing (and Web Services) will depend upon how firms adopt these applications and what they use them for. This section explores how firms have begun to use distributed or cluster computing in the past two or three years. It also describes what firms are planning to do with Grid technologies in the future and how their approach to such technologies is tied to Web Services.

In the past few years, firms have adopted cluster or distributed computing, largely as a means to gain greater computing power by linking the processors of a large number of devices. This early stage of Grid computing provided several benefits.

First, there is now less need to buy additional computing power because more computing resources are obtained from those computers that firms already own — whose processors have previously been underutilized. At the same time, a firm using distributed computing, whether in computer-aided design or another compute-intensive function, saves time in being able to complete processing a problem or analysis much faster than previously had been possible.

In an economist's view, this saves resources besides the funds that would have been spent on additional computers. Fewer hours of labor are needed to obtain the results to a problem and other resources, such as facilities and equipment, are also freed up.

As one might expect, labor productivity — the amount of work that a worker can do in a specific period of time — increases and the cost of completing a computation is lower than had previously been true. There are also cost savings due to faster time to bring products to market and reductions in the funds needed for product development.

In aerospace, Pratt & Whitney saved 60% of its development costs and cut its engineering time in half on an engine compressor project using Platform Computing's distributed computing software. In the pharmaceutical industry, according to Boston Consulting Group, using cluster computing can reduce the discovery phase for a typical drug from 6 years to 4 years — saving $264M in R&D costs. In drug firms, these savings are a result of lower labor and processing costs compared to what had traditionally been required in order to discover which new drugs have a high probability of passing through the drug approval process.

The crucial difference that distributed computing makes is not based solely on cost savings. Rather, as in pharmaceuticals, it is the fact that intensive computations can harness the insights contained in path-breaking databases covering protein and genome behavior. The result is a far more reliable identification of potential new drugs than had previously been the case.

The new results are qualitatively different from what was possible before. As a consequence, the use of distributed computing, while only an early stage in the evolution of Grid computing, has already resulted in new ways of discovering products that gives those firms, using such technologies, a competitive advantage over their rivals.

In effect there are two main impacts of the early use of cluster or distributed computing — the increase in labor productivity and the cost savings, such as the savings on labor, computers and equipment. Yet, embodied in cluster computing, is the power to improve operations and a firm's ability to compete.

For instance, Bayer, the large German-based pharmaceutical firm, was not developing as many potential new drugs as it liked. Once it realized the benefits of Grid computing it contracted with the Grid middleware firm, LION Biosciences, to establish a joint venture to identify 500 possible new drugs over a five-year period. This joint venture has been quite successful, generating new 'target drugs' faster than expected. This is one case where the new Grid computing technologies support more outsourcing and specialization in the drug discovery process.

Cluster computing has not been the only path for the early adopters of Grid-related technologies. Firms that wanted to improve the way they used their databases have linked them together in 'Data Grids' so that scientists can access information from a range of research centers easily and rapidly. Data Grids link data that previously existed only in isolated and difficult to access data 'silos'. They extend beyond research centers in the US to join centers throughout the world.

Large pharmaceutical firms — such as AstraZeneca and GlaxoSmithKline — have connected data centers in Europe and the US, thereby achieving economies based upon shortening the time needed to access and obtain important information. Having rapid access to this information:

- **saves researchers time**
- **improves their ability to evaluate new products**
- **reduces direct operating costs, by enhancing the efficiency and productivity of workers in a firm's R&D groups**
- **contributes (as a result) to improved labor productivity.**

The economic significance of Web Services

A third stream for the early adoption of Grid technologies is Web Services. Firms have adopted Web Services for two reasons. For some early users — including banks — Web Services reduce costs by permitting the Internet to substitute for brick and mortar facilities. For other firms, Web Services provide greater control over internal operations.

In the case of banks, using Web Services to establish free checking on the Internet, based upon secure Web portals that can be accessed by many consumers, saves considerable costs. This is achieved by computerizing the bank's transactions, of consumers and small businesses, moving the main contact with such clients to the Internet so that the number of customer visits to branches declines. One large bank estimated that it would gain 3-4% per year in productivity improvements over the next five years when using Web Services to provide Internet banking. The savings are due to the need for fewer tellers and facilities to serve banking customers.

In addition, the Internet provides banks with the ability to offer additional competitively priced services to customers by reading Web-based literature. This provides them with the potential to expand their share of the market for financial services since early adopters can utilize Web Services to offer inexpensive financial services to customers, even services that banks do not develop themselves but source from other financial institutions.

Due to the pricing advantage that banks using Web Services believe they will achieve on the Internet, they assert that such services will help expand their customer base since more traditional banks will be unable to compete on price. According to these banks, Web Services will offer them an opportunity to achieve significant competitive advantages in the marketplace.

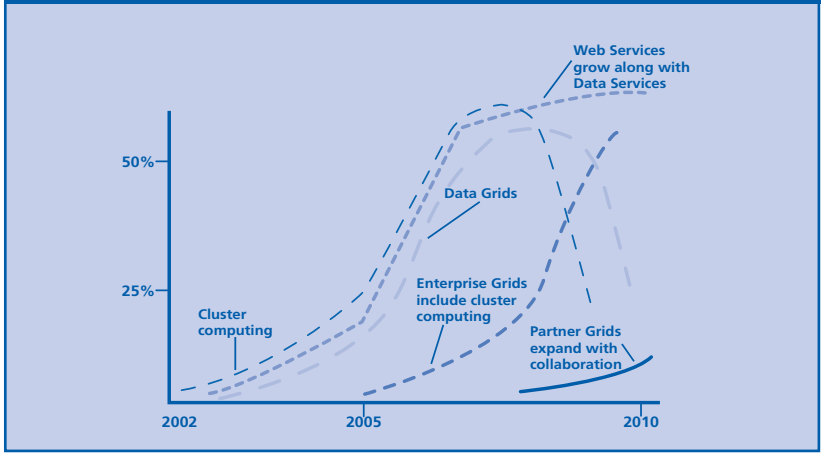
Economists who study industrial competition usually expect that when banks fear that they will lose their position in a market, they seek to improve their competitiveness. This should result in a broad adoption of Web Services in the banking industry. Since Web Services mean that banks must upgrade and enhance the links between data centers that distribute information about transactions, there should also be a concomitant increase in spending on the broadband connections between such data centers.

As a result, banks may deliver a large increase in spending (on broadband) in order to support their Data Grid-like structures which will link their distributed data centers (as these use Web Services). Doing this, banks will not only reduce their costs and enhance competitiveness, via Web Services, but they will also alter their operations because they will enhance the usefulness of those data centers.

A second reason for using Web Services is to improve the efficiency of existing operations, particularly complex operations, inventories and billing. For example, a Microsoft case study describes how Naptheon — the former information services group of Newport News Shipbuilding — built a Web Services application to manage its building, servicing and decommissioning of commercial and military ships. This system made it much easier to manage multiple contract teams from diverse organizations, thus reducing operational costs. By the same token, Goldman Sachs estimated that a large manufacturing firm using Web Services for its supply chain "could halve the amount of inventory and working capital ... carried on the company's books."

Other firms are using Web Services in concert with distributed computing and Data Grids because researchers find they facilitate secure access to needed information.

Figure 2.1: Adoption of Grids in the aerospace industry



In pharmaceutical and auto firms, researchers at different locations can easily obtain calculations about new drugs or designs even when the distributed computing is taking place behind a firewall. This meets the concerns expressed by the drug and auto firms that none of their proprietary information be readily accessible or subject to loss due to computer hacking. The main communications links in these cases are Virtual Private Networks or private lines.

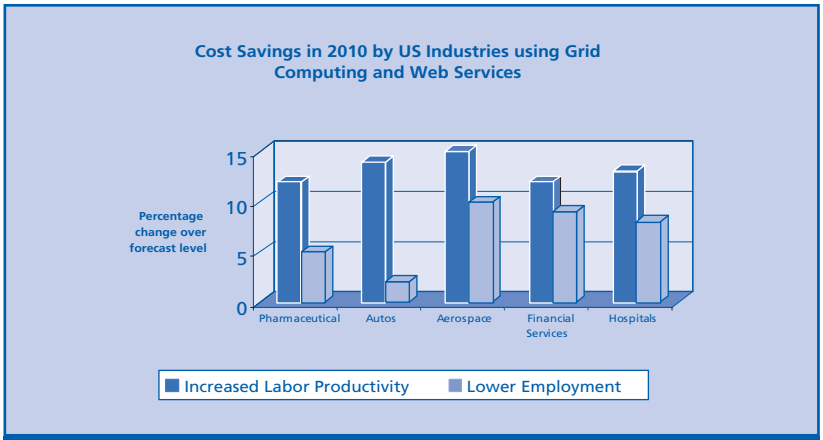


Figure 2.2: Cost savings in 2010

That said, these firms are using Web Services technology because of the time savings and greater ability to co-ordinate widespread teams of researchers and engineers. Again, access to information leads directly to improved productivity as well as reducing costs.

Future patterns

What is the future pattern for the adoption of Grid computing and Web Services? While it is difficult to know exactly how firms will adopt these technologies, firms that have discussed their plans expect the following stages:

- **first, there will be a more widespread adoption of cluster or distributed computing, Data Grids and Web Services over the next two to three years**
- **second, enterprise Grids will emerge in 2003-4 and there will be a rapid ramping up of their growth through 2007-9 (cluster computing and Data Grids are subsumed under enterprise Grids)**
- **third, partner Grids will begin to develop in 2005-6 — to link suppliers and partners to corporations using Grids and this will grow rapidly during 2007-8.**

What is likely to be the economic impact of enterprise and partner Grids? These Grids will open the way for new economies in production. According to executives that plan to deploy them, this will result from the following changes. To start with, they will link suppliers and partners more closely to a firm that uses them for products and services. This will let a larger firm more easily monitor drug testing by a Clinical Research Organization or collaborate on new part designs with a supplier. As a result, the larger firm will be more able to outsource many corporate operations to selected partners and suppliers. This should permit additional savings on operations, assuming that the partners' specialization in clinical testing or parts production exploits scale economies and a knowledge base that the larger firm does not possess. This trend to outsourcing is a major shift that many firms have discussed with us.

Enterprise and partner Grids will also permit the reorganization of business processes. In pharmaceuticals, several firms have noted that these Grids will permit them to manage the complex drug development process better. One result should be the screening out of potential new drugs that have a lower chance of making it through clinical trials since computerized screening based upon Grids provides much better information about how drugs are likely to react in humans. Success in this area should enable pharmaceutical firms to save the tremendous investment necessary to take these drugs through clinical trials. As a consequence, these firms would be able to focus their development efforts on drugs that have a greater likelihood of succeeding, thereby improving their chances of creating more approved drugs.

In addition, by using partner Grids to reach pharmacies, doctors and consumers, pharmaceutical firms may also be able to change the way their products are marketed. This might mean that groups at high risk for reactions to a specific new drug are pre-screened by doctors or pharmacies, making it possible to gain approval for drugs even if they create adverse impacts on small groups in the population. In sum, Grids begin to reach beyond the compute-power characteristics we assume for such Grids today. They will change the way firms will need to manage their operations and market their products.

Specific industry impacts

Experts at Grid middleware vendors — as well as at firms that are adopting Grids and Web Services — have esti-

mated how fast firms in five industries are deploying them. Using these descriptions, I have calculated how these five industries — pharmaceuticals, automotive, aerospace, financial services and healthcare — will probably deploy Grid computing and Web Services from 2002 through 2010.

For the aerospace industry, the adoption pattern appears in Figure 2.1 (which is similar to other industries that we analyzed). After an initial deployment of cluster computing and Data Grids accompanied by growing use of Web Services from 2002 to 2007, aerospace firms adopt enterprise Grids while Data Grids and cluster computing stop growing because they are part of the new enterprise Grids. Towards the end of the decade, aerospace firms begin to adopt partner Grids. Based upon this type of analysis, I have estimated the following major impacts on industries in four categories:

- **cost savings**
- **outsourcing savings**
- **labor productivity**
- **new Grids and Web Services.**

For cost savings, Grid computing provides many industries with cost savings. The largest savings appear to be due to the widespread use of Web Services by banks which reduces support staff costs by as much as 15-20% by 2005-6 and save as much as 30% by 2010. In the four other industries studied, the savings due to the use of Grid computing and Web Services are expected to lower costs by 5-10% between 2007 and 2010.

In the category of outsourcing savings, the auto and financial service industries' extensive use of Grids and Web Services permit greater use of outsourcing to reduce overall costs 4-7% in 2002-05 and by as much as 7-10% in 2007-10. The other industries studied will have smaller savings from outsourcing.

Labor productivity gains in the pharmaceutical, auto and financial services industry should reach at least 15% by 2010. Other gains are as described below.

For new Grids and Web Services, the five industries analyzed will spend, by 2010:

- **29% more on computers**
- **26% more on communications**
- **22% more on software**
- **8% more on communications services (mostly for Grid-related broadband connectivity)**
- **5% more on professional services**

than forecast in a national economic model. Using a model of the US economy, I have modified a baseline growth forecast to incorporate labor savings, changes in outsourcing and new investments and spending that are associated with the use of Grids and Web Services in the five industries analyzed. The main cost savings for these industries are shown in Figure 2.2.

Economic rationale for the adoption of Grid computing and Web Services

This analysis also identifies a competitive economic rationale that explains why firms will adopt Grid computing and Web Services. Adoption will be due to the competitive advantages that these technologies offer to early adopter firms.

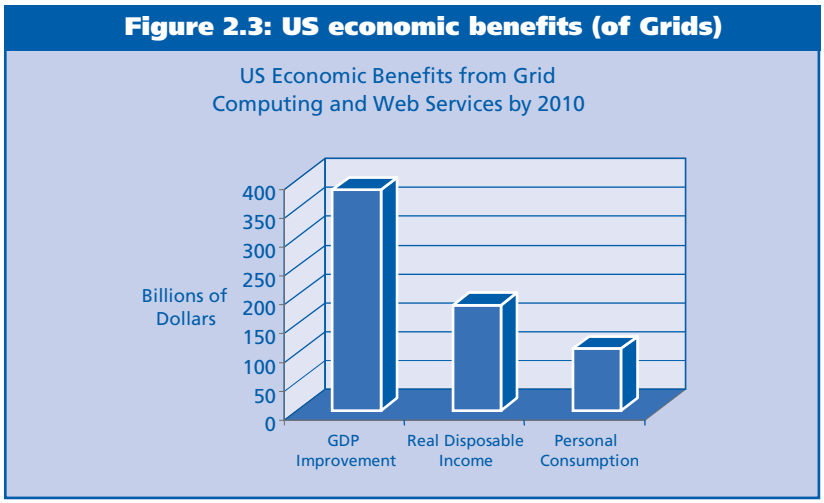
By deploying such technologies, early adopter firms will:

- **shift their competitive position**
- **put their competitors under considerable pressure to do the same.**

If firms do not do this, they run the risk of losing their ability to compete. If the pressure to change is great, it should result in many firms adopting Grid computing and Web Services to stay competitive.

If Grid computing and Web Services provide for improvements in competitiveness, this will spark a new technological revolution that will change the way businesses operate. By altering the way firms inno-

Figure 2.3: US economic benefits (of Grids)



vate, develop new products, manage their relationships with suppliers and partners and market their products and services, these technologies will force most industries to adopt them. Thus, they will not only provide impressive gains in computing power, but also alter business processes and competitiveness.

Impacts on the US Economy

This section summarizes an analysis of how Grids and Web Services are likely to affect the US economy. The analysis modeled how five industries are expected to adopt these new technologies between 2002 and 2010, using such changes to modify the expected forecast of industry growth.

The results (Figure 2.3) indicate that the adoption of Grids and Web Services will:

- **increase economic growth — expand the Gross Domestic Product (GDP) — by $384B**
- **increase real disposable income — by $184 billion**
- **add to personal consumption — by $108 billion (all amounts are in constant 1996 US dollars).**

After having an initial negative impact on US employment — because they reduce labor costs — Grids and Web Services will expand US employment because they stimulate overall growth. It is likely that if other industries adopted these technologies the economic benefits of Grids and Web Services would be larger than estimated here. Grids and Web Services will also impact additional industries because they are likely to require increased spending on software, computers, communications equipment and communications services. This should benefit US information technology industries, particularly the telecommunications services. Spending on Grids and Web Services should expand spending on broadband links and increase data traffic, thereby contributing to the revitalization of the US telecommunications industry.

Management conclusion

For those who see the Grid, and Web Services, as being about technology only, this analysis by Dr. Cohen places a wholly different perspective on them. If they are adopted as described, the macro economic benefits are going to be substantial, both in additional growth as well as improved efficiencies in commerce.

Although there will be an initial blip in terms of employment (attributable to that improved efficiency), the reinvigoration of the telco industry (by Grids requiring major bandwidth increases) and the IT industry (new applications, of all sorts, as well as hardware) will do much to offset those losses — over time. The dot.com bubble may have burst but Dr. Cohen shows how growth can continue, albeit in a different form.

The final implication is micro-economic. It is that firms of all sizes will need to consider how they are to invest in and exploit both Grid and Web Service technologies — if they are not to be left behind to become uncompetitive.

References:

- **Kim Girard, 'With Due Haste', Baseline Magazine, April 2002**
- **Microsoft, Visual Studio .NET Product Information, Case Studies, 'Newport News', March 25, 2002: http://msdn.microsoft.com/vstudio/productinfo/casestudies/newport/default.asp**
- **A.T. Kearney and the Stencil Group, 'The Emerging Web Services Market': http://www.stencilgroup.com/ideas_scope_200203atkws.pdf**

Grid computing — middleware but not as we know it

Mark Lillycrop
Consultant

Management introduction

Like the utility power grids which gave it its name, Grid computing is a pretty straightforward concept: it is, to borrow IBM's description, "distributed computing over a network, using open standards to enable heterogeneous operations."

That is not a bad definition. Most Grid computing offerings are aspiring to live up to these essential rules, although some vendors have hijacked the term to describe something more banal — small groups of locally attached systems that are more like clusters than grids.

In this analysis, Mark Lillycrop looks at how Grid computing requires middleware. But this is not necessarily middleware as we know it, as he describes.

Diversity inhibits understanding

The problem is that, from a simple definition, Grid computing has sprouted a huge diversity of interpretations and levels of scale. There is SETI — probably the most famous Grid application to date — where PC users allow the spare cycles from their idling machines to be harnessed over the Internet and put towards unlocking the secrets of the cosmos. Then there are medical applications calling for collaboration between specialists worldwide. The scope moves on to include biotechnological and petrochemical systems requiring vast but irregular amounts of number crunching.

In short, the potential of Grids is enormous. Whether it involves the sharing of processor power, data, storage or bandwidth, we are only just beginning to appreciate the long-term implications of Grid applications.

Right now, however, the Grid is still a pretty specialized business. Most of the applications that have made it into the headlines have been scientific in nature and processor-intensive. This is a far cry from the transaction management concerns of the average commercial user.

Looking further down the road, however, Grids are likely to provide the backbone for a whole range of utility-based, computing-on-demand, services. In the months ahead, Grid technology will have to make the credibility jump from the scientific world into the business mainstream, in much the same way that the Web transformed an established network for military communications into the consumer phenomenon of the late 1990s.

For this to happen, though, the focus will need to move:

- **away from the broader infrastructure of Grid systems, which has captured the popular imagination (up to now) through standards bodies like Globus**
- **towards the load-and-go application-specific middleware that is needed to adapt the Grid concept for the wider commercial market.**

New middleware players

In the last couple of years, a number of relatively unfamiliar names have gained prominence in the middleware sector. These are companies that specialize in some specific elements of Grid computing. Among the leaders are:

- **Entropia, which is positioned clearly at the SETI end of the market with a range of products for PC Grids: this Company's main marketing message is RoI and it stresses the poor utilization level of PCs and the benefits to be gained by capturing unused processing power; its flagship product, DCGrid, includes a number of tools to help users get compute-intensive applications up and running on PC networks**
- **United Devices, which sells MetaProcessor middleware products that offer a broad range of Grid computing options based on both Windows and Linux technology; MetaProcessor is designed for cross-platform applications, and offers a level of scalability that is missing from many PC-focused products; in addition it comes in a number of flavors, including an enterprise version for larger companies, an alliance version for third-party providers and an on-demand service which is claimed to provide over 8,000 CPUs and 14 teraflops of power**
- **Avaki, which has gained prominence in a short period of time because of its active role in developing industry-wide standards for Grid computing; its Compute Grid and Data Grid products (which can be acquired as a combined 'Comprehensive Grid') demonstrate the Company's two-pronged approach to Grid computing as well as careful attention to product packaging**
- **DataSynapse, which has achieved considerable success by focusing on Grid application availability and management and on so-called 'adaptive scheduling'; with expertise in a number of vertical markets, it has become a major IBM Grid partner**
- **Platform Computing, which has been and is a major contributor to the Globus initiative for constructing standards-based large-scale Grids; Platform is one of the more established players in this area and, partnering with most of the IT giants, it offers a considerable range of tools for managing heterogeneous Grids.**

Lacking the conventional middleware connection

A brief examination of companies like those listed above highlights one of the current drawbacks of Grid computing. There are several distinct approaches to implementing heterogeneous, distributed applications across a network, none of which can entirely claim to be standards-based.

The difficulty is that, while the leading Grid middleware suppliers are making considerable progress in packaging their applications in a useable form for a wider market, and focusing strongly on interoperability, the need to integrate

Grid middleware more closely with existing enterprise middleware has yet to be addressed. Too many of the product suppliers in this area are working, to a greater or lesser extent, with bodies like Globus, with its Toolkit for building Grid-enabled applications and its architectural arm, the OGSA (Open Grid Services Architecture, not to be confused with the Owassa Girls' Softball Association which, I discovered, owns the ogsa.org domain).

However, like all de jure standards, progress is slow, and the challenge facing the standards bodies is to create a set of minimum criteria for Grid developments and Web Services without undermining the value-add of individual participants. In the short term, therefore, there is clearly a need for the major IT vendors to pool their resources with the Grid computing specialists to work on much closer relationships between:

- **conventional middleware**
- **Grid technologies.**

If these relationships succeed and as Grid computing moves into the mainstream, there will be some very significant changes in the next generation of IT architectures. Indeed, as Grid technology continues to evolve over the coming months and years, we may well see some enterprise middleware slowly disappearing — to be replaced with a new breed of tools that use Grid principles and maturing Web Services to integrate inter-enterprise data and processing resources.

IBM and vertical Grid offerings

Such possibilities make IBM's recent announcements particularly interesting. In January 2003, IBM published a set of 'Focus Areas', with accompanying technical architectures. It identified ten applications within five target areas for on-demand Grid technology. IBM also took the opportunity to align its own management middleware products with offerings from the Grid specialists for each of these vertical Focus Areas, providing customers with a number of choices depending on their specific requirements (Figure 3.1).

These announcements achieved two important things:

- **they underlined the respective strengths of the new-breed Grid middleware specialists in various vertical markets**
- **they went further than any other initiative, to date, in positioning Grid middleware for use in mainstream commercial applications.**

The IBM Focus Areas and associated applications are as follows:

- **Research and Development ("Accelerate and enhance the R&D process by enabling the sharing of data and computing power seamlessly for research-intensive applications"); this includes Life Sciences where the offering is the 'IBM Grid Offering for Information Accessibility' ("Share data and computing power, for computing-intensive engineering and scientific applications, to accelerate product design")**
- **Engineering and Design which applies to Aerospace and to the Automotive industry (the IBM Grid Offering for Engineering Design/ for Design Collaboration)**
- **Business Analytics ("Enable faster and more comprehensive business planning and analysis through the sharing of data and computing power"); this is aimed at Financial Markets and Life Sciences under the title 'the IBM Grid Offering for Analytics Acceleration'**
- **Enterprise Optimization ("Optimize computing and data assets to improve utilization, efficiency and business continuity") which concentrates on financial markets (the IBM Grid Offering for IT Optimization)**
- **Government Development ("Create large-scale IT infrastructures to drive economic development and/or enable new collaborative government services" — the IBM Grid Offering for Information Access).**

These 'Areas' will expand in number and coverage over time. For example, IBM has since added a Grid-based gaming application.

Commercial middleware

These new vertical categories provide a useful clarification, and add credibility, to Grid computing as a means of meeting flexible customer demand in today's commercial IT environment. They are, though, still primarily aimed at:

- **large-scale processing projects**
- **research, design and analytical tasks.**

If we consider the real long-term destination of Grid technology — as a means of creating a massive, flexible pool of IT resources for on-demand computing — there are still a number of major issues that need to be addressed. Of the five Focus Areas, the most intriguing are the two so-called 'Efficiency Grids', Enterprise Optimization and Government

Development (the other three are described as 'Performance Grids'):

- **Enterprise Optimization provides a foundation for taking Grid capabilities into the area of server consolidation and the flexible allocation of IT resources to achieve maximum cost-effectiveness — in other words, the provision of capacity-on-demand type applications that will offer a larger, commercial role for Grids in the future**
- **Government Development, aimed at the lucrative and burgeoning e-government market which hints at another commercial use for technology that has hitherto been confined to scientific laboratories, with large-scale collaborative access to official data and resources; again, the initial applications are processor-intensive, but government business will also be a key driver for data-intensive Grid applications as they mature.**

New directions for Grid middleware

To make real progress in new business-related areas, particularly support for application integration and business continuity, Grid technology will need to gain far greater support for common transaction management and security services over the coming months. Part of this provision will no doubt come from increased exploitation of Web Services in general, but pressure will also be exerted on Globus and OGSA to provide new industry standards.

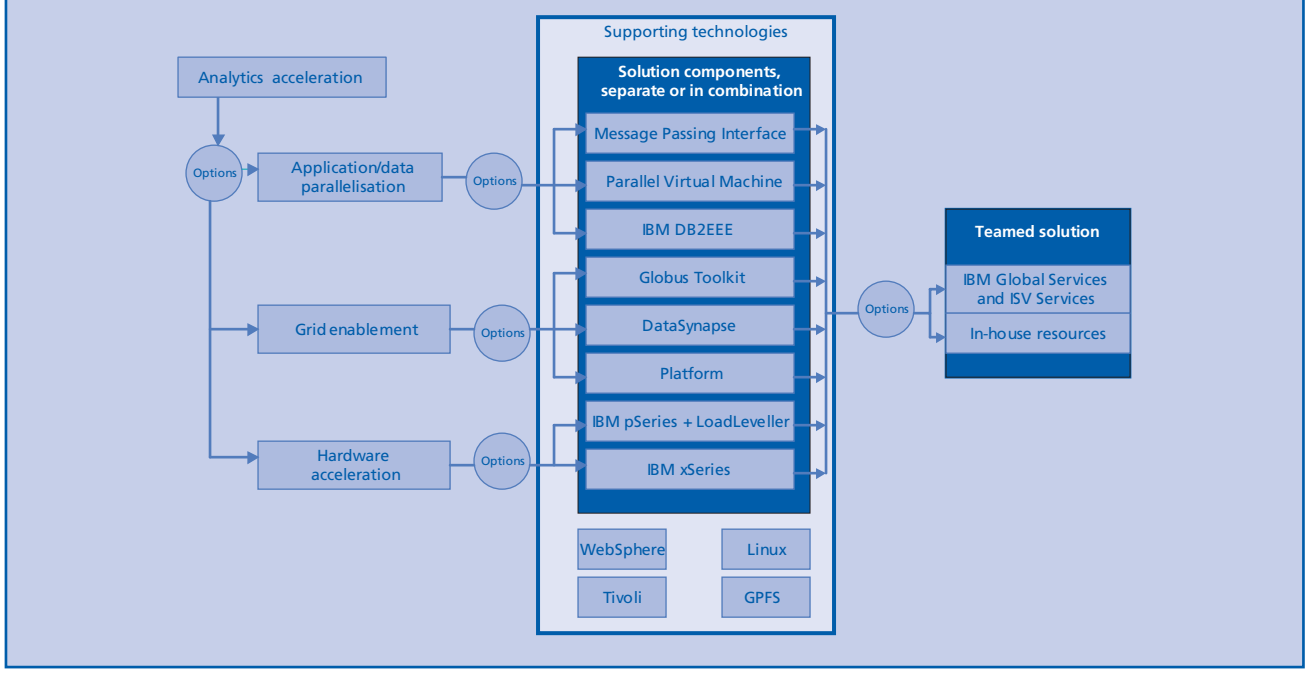
As far as transaction management is concerned, various proposals are being considered for managing multi-phase commits and handling synchronous and asynchronous messaging data across a Grid. Ironically, the Grid mechanisms designed so far have a lot in common with existing messaging middleware; systems participating in the processing of a vast calculation may need to be co-ordinated in a similar way to the inter-related transactions managed by today's messaging middleware.

But today the Grid focus has been far more on harnessing processing power, with relatively little attention paid to the commercial world which tends to be characterized by very large numbers of small data items. Future releases of products — such as WebSphere MQ — will position these for a Grid-style environment. Equally, products from vendors such as Platform and DataSynapse will increasingly feature data management facilities.

Security — the big issue?

In moving Grid capabilities into the transactional environment, security is the challenge that currently preoccupies Grid designers. Basic network security between participants is certainly manageable, but maintaining secure interac-

Figure 3.1: IBM's Grid focus areas and choices



tions between heterogeneous platforms within different organizations is another matter altogether.

Globus, not surprisingly, is coming to the rescue. The Security Architecture for Open Grid Services (published last summer) provides a blueprint for the services that will need to be standardized in order to manage secure transactions and data over a Grid.

Globus tends to describe the participating systems in a Grid as members of a Virtual Organization (VO). VOs are not new, nor unique to Grids. Ever since the late 1990s, when the Internet first allowed companies to interact more closely with partner organizations and customers through extranets and IP-based VPNs, we have faced the problem of how to define the relationships between these partners. (Watertight security policies are hard enough to establish within the boundaries of a legal entity; but break through those boundaries and security becomes a much more complex issue.)

According to the published specification for the Security Architecture for Open Grid Services, we need to address three distinct areas of security management:

- **integration**
- **interoperability**
- **trust relationships.**

Integration refers to the way that Grid security services work together with security tools already implemented on participating systems. The paper argues that it will not be practical for Grid security mechanisms to replace those already in use on individual platforms, nor to over-ride existing procedures. So Grid security services will need to be:

- **'implementation-agnostic', to avoid problems of incompatibility**
- **integratable with existing services**
- **adaptable, to embrace any new platform-specific services that may come along.**

Interoperability addresses that way that domains within a Grid exchange security-related information. According to the specification, interoperability needs to be managed at:

- **the protocol level (such as via SOAP/HTTP)**
- **the policy level (so that owners of the systems involved specify ways of engaging in a secure conversation)**
- **the identity level (mapping identity and credential information across multiple domains — where possible, providing a standard means of establishing identity).**

Trust relationships means the mechanisms are in place to verify the credentials of an individual or service seeking access to an organization's IT resources. Such mechanisms can be:

- **either 'presumed', through something like a VPN**
- **or 'explicit', as determined by policies and authentication procedures for each access required.**

The point about establishing trust (which is an essential means of streamlining security management) is that this is far more difficult to achieve in a Grid environment, which is characterized by transient services. Applications and users within a Grid seek to access to each other's processing and data resources on a far less predictable basis that would normally be encountered by conventional security policies or tools. This presents security managers with a formidable administration task.

Grid limitations

The Security Architecture for Open Grid Services goes on to describe in some detail how each area of security might best be tackled and standardized within a Virtual Organisation. But the specification also highlights how much farther we have to go to make Grid Services applicable to the commercial world in general.

The jump — from harnessing unused machine cycles and applying them to a huge mathematical calculation on the one hand to the transient use of systems to handle everyday applications on a utility basis on the other — is relatively small in technical terms. But for organizations to adopt and benefit from this technology on the scale envisaged by vendors, means that significant cultural changes will have to take place.

The whole idea of a Virtual Organization, for example, faces considerable limitations outside academia. Even the 'extended enterprises' that were promised during the e-business boom have failed to materialize in the ways that were originally proposed. Automated supply chains and similar tightly coupled channels of communication between business partners tend to be carefully controlled through rigid security policies and integration of security management tools.

It is a considerable leap of faith from this narrowly defined extended enterprise to the kind of VO proposed by Grid proponents. It will happen. There is little doubt about that. The economic implications of Grid computing are too powerful to ignore. But much has to be done first to obtain commercial confidence in the building blocks that make up Grid applications.

Management conclusion

If Grid applications are to have a real impact in the mainstream commercial world, and fulfil the promise of utility computing, the answer must lie in middleware evolution. Over the next few years, integration mechanisms like SOAP and XML, messaging products such as WebSphere MQ and system management products from the likes of Tivoli, CA and Sun, will begin to expand their horizons. They will need to gain the functionality to manage data, applications, user identities and other resources within a less clearly defined organizational framework than is found today.

These maturing products will borrow a great deal of know-how from the 'new boys' discussed earlier, and from the Grid standards bodies, subtly opening up opportunities for users to address security and transaction management challenges within a transient service environment. At that point, Grid Computing will have come of age.

But, put another way, 'its success will all be in the middleware, albeit not as we have known it in the past'.

Generating middleware with automated tools

Tom Welsh
Consultant

Management introduction

The purpose of middleware is to save time and money by simplifying the incredibly complex business of writing communications software. But middleware itself has grown more and more complex, to the point where it is sometimes just not cost-effective.

In recent years there has been a trend towards automating the creation of middleware, culminating in present-day efforts to streamline the deployment of Web services. This analysis by Tom Welsh explains the underlying issues and sums up the state of the art:

- ***starting with RPCs and DCE***
- ***before moving onto CORBA, .NET and Web Services.***

Background

Software re-use is a constantly contentious subject. Many probably feel that its benefits are unproven. Yet some of the most important advances in software technology have boiled down to new and ingenious ways of re-using pre-written code. For example:

- **graphical user interfaces (GUIs)**
- **database management systems (DBMS)**
- **fourth generation languages (4GLs)**
- **and even operating systems and compilers**

all amplify a programmer's effort by enabling large chunks of tried and tested functionality to be invoked with short, simple commands. In each of these cases, the key to success lies in striking the optimum balance between power and flexibility.

At one extreme, imagine a system that provides only a handful of commands — each of which mobilises an immense amount of underlying code. That approach is too coarse for most purposes, as the programmer does not have enough control over the resulting application's detailed behavior. Those of us who have experienced the joy of writing assembly code already know exactly what the other extreme looks like: the programmer has comprehensive control, but producing any sizeable application takes far too long.

So, as the industry's experience of creating software accumulates, our understanding of the trade-offs involved in re-use is gradually being refined. To design and code the equivalent of today's Windows or Macintosh user interface alone would have been far more than a lifetime's work for a skilled programmer 20 years ago.

Yet we take these facilities for granted. Today we use them as a starting point when designing our own applications. This produces a vast saving of time and effort.

The lessons of RPC and DCE

The classic 'open systems' middleware was — and still is, to some degree — the remote procedure call (RPC). As shown in Figure 4.1, a generic RPC product supplies almost all of the software infrastructure required for an application to call procedures located on remote machines across a network. The shaded boxes (in Figure 4.1) represent functionality that comes with an RPC product, while the white boxes must be coded by the application programmer.

Even at this level, it is clear that the RPC product does most of the heavy lifting. Its runtimes and stubs are libraries that handle all the actual network communication chores, as well as allocating threads and marshalling the data to get it into the right format for the remote computer. Almost all the application programmer has to do is write the business logic for client and server.

On the server side, this comprises a set of 'server operations', each of which fulfils one remote procedure. On the face of it, then, an RPC product makes the task of writing middleware completely transparent, so that the programmer is hardly aware of writing distributed software.

Actually, things are not quite so simple. As I said at the beginning, it takes some time and a few iterations before designers can hope to strike an ideal balance between simplicity and power. On the server side in Figure 4.1, the box marked 'Server Control' represents a number of special calls that the application must make in order to initialize, register and manage its RPC interface. The application is also responsible for top-level error handling as well as cleaning up failed sessions. Another important point is that RPC itself is rather limited, providing only for clients to bind to servers and make remote calls.

The Open Software Foundation (OSF), which has since become part of The Open Group, addressed this concern by creating the Distributed Computing Environment (DCE) standard. This was a much more complex system which supplemented RPC with additional standard services, including:

- **directory**
- **distributed time**
- **threads management**
- **security.**

Unfortunately, DCE's inherent complexity was considered excessive by many organizations and it never achieved great popularity. It was just too challenging, even if it did deliver valuable services. The difficulty of programming 'by hand', combined with a veritable wall of reference manuals, was one of the main reasons for its poor market acceptance.

The RPC development process seems complex at first sight. But it quickly becomes familiar. The first step is to write down the details of each interface, using Interface Definition Language (IDL).

This deliberately simplified language — which has no way of describing how operations are to be carried out — is an important contribution to software engineering. Experience has shown that better results are gained by agreeing

on interfaces first and then writing code to fulfil them, than by the usual practice which reverses the order of these activities. Special tools are provided to translate IDL into C, C++ or another language, and this code is then compiled together with the client and server application code.

New techniques for hiding complexity

In a sense, the RPC and DCE designers were handicapped by aiming at a moving target. Their very success in allowing ordinary programmers to create practical distributed applications greatly increased demand for such applications. This in turn sharply raised expectations.

Now the ideal middleware tool came to be seen as one that hid all the messy details, leaving programmers to concentrate solely on writing business logic. But this was much easier said than done.

While increasing numbers of UNIX and, later, Windows developers began using RPCs, middleware designers branched out in several different directions as they sought to deliver higher productivity and ease of use.

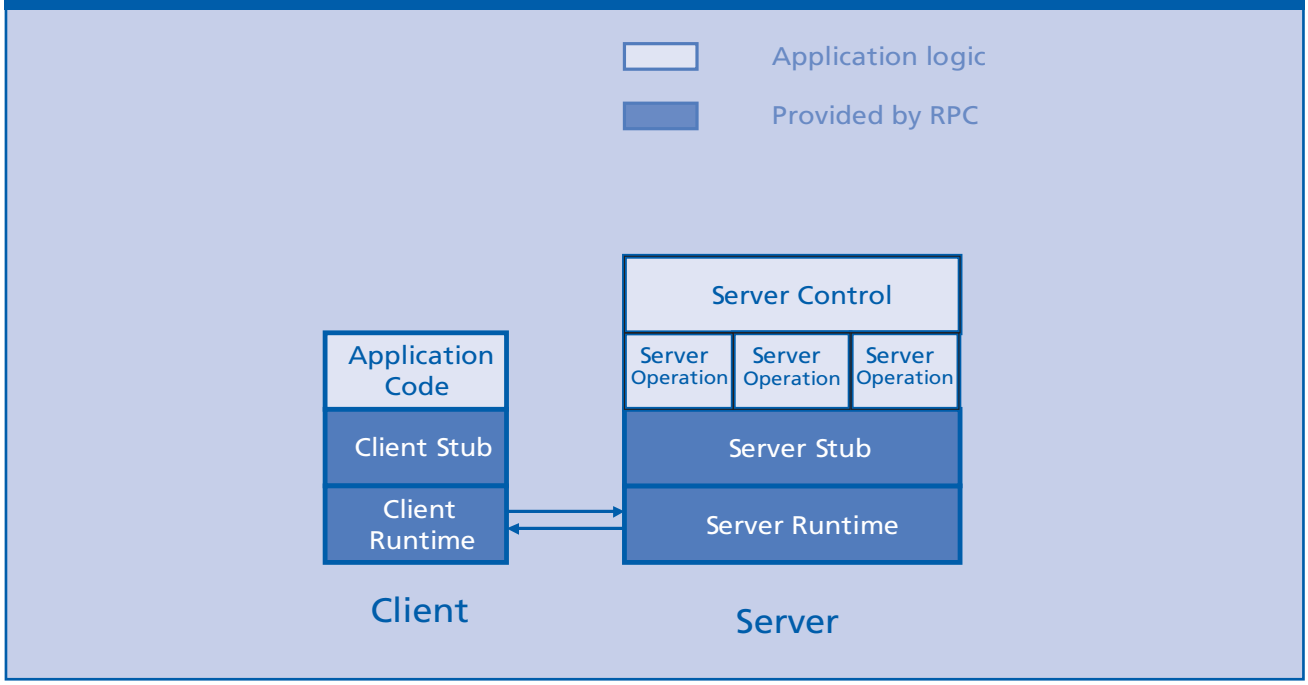
Specialist DBMS and database tools vendors found that they could hide RPC capabilities inside their products, using it as a convenient way of exchanging commands and data.

In a typical two-tier client/server system, Windows or UNIX clients talk to a DBMS server over a LAN. Client-side products like PowerBuilder and VisualBasic — with their built-in SQL-over-RPC links — gave relatively unskilled developers the ability to put together quite powerful applications very quickly indeed.

The secret, as usual, lay in methodical re-use. These products automatically created the GUI front end and the database back end, enabling the developer to accomplish the desired results mainly by customizing a 'default' application. This form of middleware sacrificed a good deal of flexibility — which was felt to be unnecessary for the given task — in order to make the development process quicker, easier and less error-prone.

The 'third generation client/server' vendors took this kind of middleware automation and generalized it. Companies like Dynasty, Forte, Nat Systemes, Seer, Template and Unify launched some extremely sophisticated (and correspondingly expensive) products in the early and mid 1990s. The common distinguishing feature of this third generation of client/server was the addition of the third, or middle tier. This became an architectural component that eventually evolved into the application server of today. In many ways, these products were forerunners of J2EE and .NET — the critical difference being that some of their most attractive differentiating features were proprietary.

Figure 4.1: Compenents of a typical RPC product



Take one product, for example. The Forte Application Development Environment (ADE) completely separated the writing of code from the process of partitioning — determining which procedures should be deployed to the various clients and servers:

- **programmers wrote their applications without worrying about which piece would run where**
- **partitioning was done separately at deployment time.**

As well as Forte's own proprietary middleware, developers could choose between use of DCE or the Common Object Broker Request Architecture (CORBA) or Transarc's Encina transaction processing monitor (TPM). In every case Forte would generate, build and deploy all required middleware code.

In addition, a new partitioning scheme could be imposed at any time — so that, for instance, an application could be rolled out on PCs and VAX servers, then (without a single code change) on UNIX workstations and servers. Unfortunately, not many organizations were willing to pay Forte's prices and the company was eventually acquired by Sun.

It can be argued that products like those produced by Forte were good value, even at the high prices charged, for the degree of automation of middleware generation delivered. But they have been superseded by a new generation of J2EE-based application servers which are more open and less conducive to vendor lock-in.

Nevertheless, even the best J2EE products still do not offer anything like the capabilities which Forte provided in the past. But many of these will no doubt reappear in years to come.

COM and CORBA

While the third generation client/server dinosaurs ruled the earth in the 1990s, two middleware standards were slowly maturing. CORBA and Microsoft's Component Object Model (COM) were both attempts to improve on RPC, although with different design foci.

CORBA generalized the RPC concept by adding object orientation and language transparency to RPC's location and platform transparency. It was intended, above all, to be a universal glue that could bind together the heterogeneous variety of:

- **operating systems**
- **languages**
- **networking protocols**

that large organizations inevitably collect. COM, on the other hand, was Microsoft's attempt to provide a universal glue for components running on Windows.

Figure 4.2: Setting COM+ attributes at the source code level

```
cointerface IBank
{
    double DebitCredit([in]long lAccount,
                [in]double dAmount);
}

coclass Bank : implements IBank
{
attributes:
    transaction = "required";
    data_source = "DSN=BankDatabase";
    state = "stateless";
    threading = "rental";

public:
    DataSource m_dsBank;

    [ source=m_dsBank, column="balance" ]
    double m_dBalance;

    double DebitCredit(long lAccount, double dAmount)
        throws SQLException
    {
        m_dsBank.Source = "select * from Accounts where id="
            + ToString(lAccount);
        m_dBalance = m_dBalance + dAmount;
        return m_dBalance;
    }
};
```

Originally limited to local operation, it became middleware in the generally accepted sense with the launch of Distributed COM (DCOM) in 1995-6 (in the rest of this analysis, 'COM' will be used in the sense of 'COM and DCOM').

COM and CORBA use different forms of IDL, which support their respective features and data types. There is no compelling technical reason why it should be any harder to provide automated tools for CORBA than for COM. But, while every Windows development tool worth the name supports COM, the lack of such support for CORBA is one of the most obvious explanations for its relative lack of success.

Programmers who want to get the most out of COM turned to Microsoft's Active Template Library (ATL), which was specifically designed for writing COM servers and is still supported by most Windows development tools. Visual C++ and Borland's C++ Builder provide wizards for creating COM servers with ATL, and COM clients with the more GUI-oriented Microsoft Foundation Class (MFC) library. The compilation and deployment process is also slick with these top-of-the-line integrated development environments (IDEs).

COM was succeeded by COM+ which, as well as being cleaner and easier to use, also absorbed some other important products — like Microsoft Transaction Server (MTS).

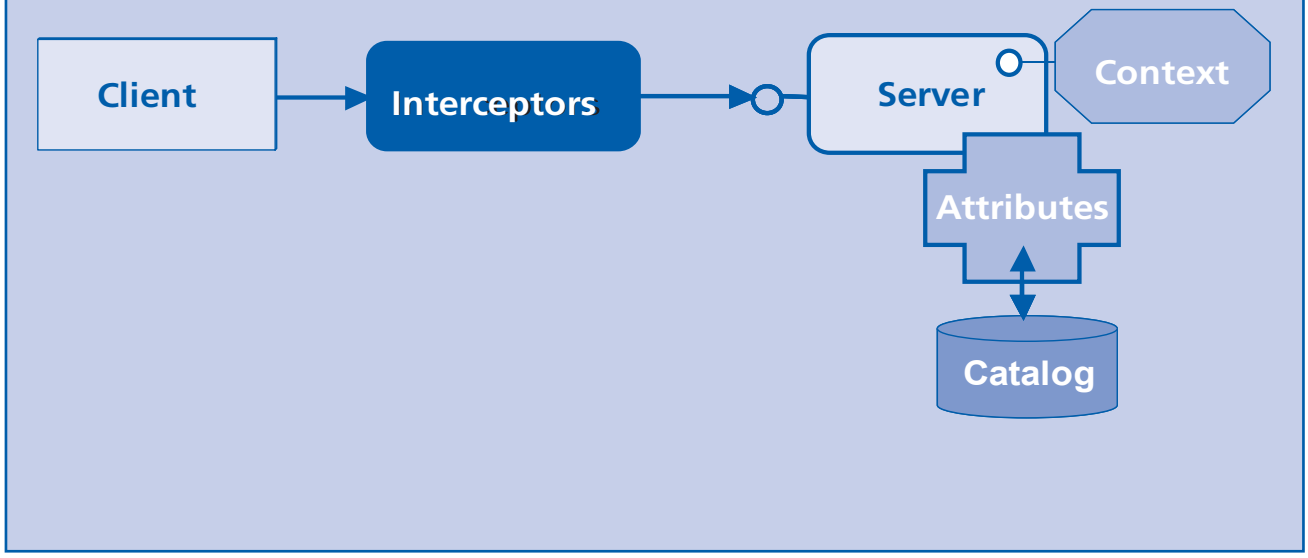
Even in the .NET era, COM+ (now in the guise of Enterprise Services) remains Microsoft's closest answer to Enterprise Java Beans and J2EE. In the words of Microsoft's COM+ Manager, Frank Redmond, 'COM+ is about building scalable component-based applications' — a description that might have come from any CORBA advocate.

One of the most interesting and innovative aspects of COM+ is that a developer can set up attributes — such as transactional behavior, security or threading — with simple declarative statements. Figure 4.2 shows a trivial example (the relevant lines are in bold print, which has no other significance). Choosing the appropriate settings for the 'transaction', 'state' and 'threading' attributes can play an important role in making sure that an application scales well under heavy workloads.

Some COM+ attributes can be modified even more easily, as shown in Figure 4.3, where a mouse-click selects the level of transaction support and timeouts. Some experts feel that this kind of 'declarative' or 'administrative' parameter-setting shows the way for future improvements in the automation of middleware development.

With dozens of CORBA implementations in circulation, there is a distinct limit to what development tools can assume about any given ORB. That is why most of them, including the market leader Rational Rose, stick to generat-

Figure 4.3: COM+ interceptors



ing IDL from designs — usually expressed in Unified Modeling Language (UML).

Much like RPC, the whole philosophy of CORBA requires that:

- **IDL comes first**
- **then server skeletons and client stubs are generated from this**
- **finally programmers fill in the rest of the code.**

But this leaves little room for the more ambitious automation of the kind performed by Visual Studio.NET and Web Services.

Web Services

For the past two years, the idea of quick, easy and flexible middleware development has been associated above all with Web Services. This new approach to middleware owes much of its undoubted popularity to the prospect that any two computers can be linked with a minimum of cost, delay and disruption to existing hardware and software.

The leading Web Services development tools, which embody these qualities to a marked degree, include:

- **Microsoft's Visual Studio.NET**
- **Cape Clear's CapeStudio**
- **The Mind Electric's GLUE**
- **Systinet's WASP.**

Nearly everyone who tries creating a Web Service with Visual Studio.NET for the first time remarks on how easy and painless it is:

- **the IDE dedicates a single pane to designing Windows Forms, editing code and browsing the Web**
- **its unified editor supports all the Visual Studio.NET languages, including all special features.**

Thanks to Microsoft's Common Language Runtime, developers are pleased to find that they can turn any function into a Web Service simply by adding the keyword 'WebMethod', or make it transactional with the 'Transaction' keyword. Data access, too, is highly automated by ADO.NET, leaving the developer free to concentrate on writing business logic.

While freely admitting that Web Services can be written for many platforms — from IBM mainframes to AS/400s, Macintoshes and Linux — Microsoft relentlessly hammers home the message that 'it is far easier to create and assemble Web Services with Visual Studio.NET', on Windows.

Java

J2EE also abstracts many routine middleware development tasks, although not quite as well as .NET. The Remote Method Invocation (RMI) is very similar to RPC and CORBA in this regard, taking several explicit steps to build services and clients. When used in Enterprise JavaBean (EJB) remote interfaces, though, this build process is hidden because communications are handled by the EJB container. The same applies to the Java Message Service (JMS) and Message Beans.

BEA Systems' WebLogic WorkShop is quite similar to VisualBasic, except that it generates Web Services backed by J2EE components. It introduces a special new format known as Java Web Services (JWS), which BEA has submitted to the Java Community Process for standardization as JSR 181, Web Services Metadata for the Java Platform.

The JWS format is plain Java, with annotations added as Javadoc comments. Figure 4.4 shows a small section of a JWS file, with notes explaining what each JWS statement does and how many lines of source code it stands for.

Once a developer has created a JWS file — complete with synchronous or asynchronous Web Services, using either HTTP or JMS — the file is compiled and placed on the server. For each JWS file, the WebLogic container creates four EJBs in addition to whatever business logic the developer placed in the file. These beans handle, and reply to, incoming Web Service messages. Each operation invocation is automatically wrapped in a transaction.

One-stop shopping versus the open market

One theme that emerges from everything we have seen here is that a vendor like Microsoft, which supplies practically every single component of a distributed system, has a great advantage when it comes to automating the creation of middleware. That is because the development tool can collect so much more knowledge about the state of the system. For instance, the Windows Component Services Explorer (a Microsoft Console snap-in) can be used to:

- **create and configure COM+ applications**
- **import and configure COM+ or .NET components**
- **export and deploy COM+ applications**

- **administer local and remote COM+ applications and components.**

This is possible because Microsoft alone controls all the relevant code and interfaces. After all, Microsoft is the only supplier of Windows and COM+.

A CORBA product, such as Iona's Orbix 2000 or Borland's VisiBroker, could attempt to provide similar features. But this would be much harder (or even impossible) because it might have to collect information about CORBA products from several different vendors.

J2EE is much further along in this respect. But even J2EE products interoperate only to a limited extent.

There is, therefore, clearly a fundamental trade-off between:

- **the advantages of buying from a single vendor**
- **those of choosing products from an open, competitive market.**

Slick, fast, efficient development tools are one of the benefits of one-stop shopping.

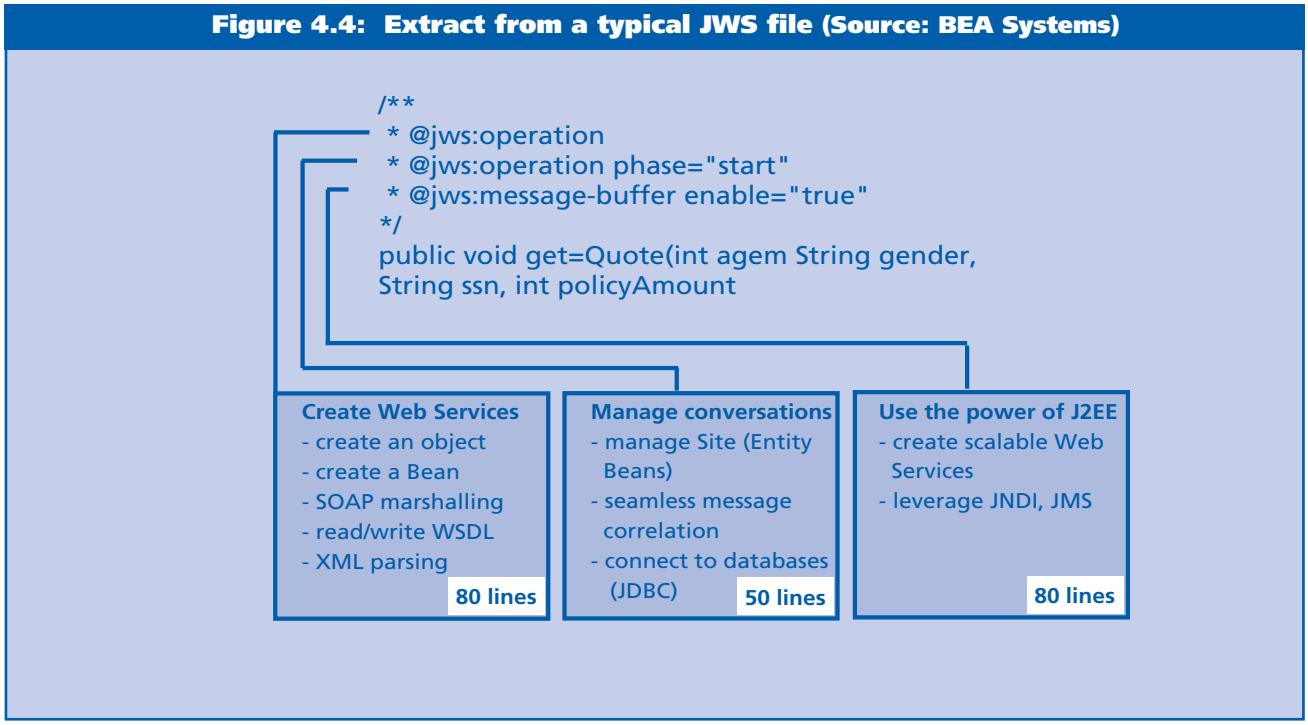
Automated middleware development

Automated middleware development is becoming steadily easier within the two great, integrated environments — Microsoft's .NET and J2EE — largely because they have each attained a 'critical mass' that makes such refinements worthwhile. Thus the .NET developer can set up .NET Remoting or COM+ queued components with a few mouse-clicks, thanks to Visual Studio.NET and its wizards. Similarly, a Java programmer does not have to write much code to link two EJBs through RMI or JMS.

Indeed, it would be fair to say that location transparency has come to both .NET and J2EE. While .NET offers language transparency but not platform transparency, J2EE, for its part, has platform transparency but not language transparency. Building communications between .NET and J2EE is a good deal trickier, however, for a number of good reasons:

- **first of all, nobody is in overall charge of the process — IBM, Microsoft and others have worked out ways of interoperating by way of Web Services: but none has much commercial interest in streamlining the process**
- **secondly, .NET and J2EE use Web Services in different ways: Visual Studio.NET defaults to Document/Literal style, while almost all Java tools default to RPC/Encoded**
- **thirdly, nobody is co-ordinating changes in .NET and J2EE with an eye to maintaining what ever level of compatibility does exist.**

Figure 4.4: Extract from a typical JWS file (Source: BEA Systems)



At the same time, it is interesting to note that there are parts of the IT industry in which the automation of development tools is not seen as an important priority — or even as desirable in principle. Many UNIX and Linux programmers much prefer to work with the simplest tools, so that they can understand exactly what is happening at all stages.

Management conclusion

While software re-use is a powerful technique, which has been progressively applied over the last 50 years — and without which modern software development would be impossible — it must be applied intelligently. All middleware is a form of software re-use, but in order to give programmers adequate control over its working, the middleware development process has sometimes become too complex.

Efforts are now being focused on the important problem of simplifying the middleware development process 'as far as possible, but no further', to paraphrase Einstein. Useful progress was made in the 1990s, but the resulting products were often excessively proprietary, expensive, or a combination of both.

Today the techniques discovered in recent years are finding their way into popular development tools such as Microsoft's Visual Studio.NET, BEA's WebLogic WorkShop and Systinet's WASP. The two most promising techniques are:

- ***declarative programming***
- ***automatic generation of code.***

When used together, the results are good. However programmers must remember to consider the complexity of the software that is eventually generated and deployed — rather than abdicating all responsibility to their tools. If they do this, generating middleware with automated tools is a real prospect.

Achieving application co-ordination with business transaction management software

Alastair Green
CEO and CTO, Choreology Ltd.

Management introduction

In the excitement of Web Services and the Grid, basic fundamentals are often forgotten. In this analysis, Alistair Green — the CEO/CTO of Choreology Ltd and a co-author of the OASIS BTP specification — looks at the importance of delivering application co-ordination when combined with transaction management.

Mr. Green is qualified to write on this topic because Choreology has been providing business transaction management software for application co-ordination. From the work that the Company has performed for customers it has been able to draw key lessons as well as participate in the OASIS Business Transaction Protocol (BTP) process.

Evolving beyond 'build, inspect and fix'

New technology has always been a key enabler for reducing inefficiencies in the way organizations do business. In recent years technology that addresses the management of business processes has been a popular target for investment. But, because these processes are too often managed in isolation, organizations have not achieved the projected benefits. One way to address this problem is to consider technology that co-ordinates processes and the systems they involve, thus ensuring consistent business results.

In many industries, the elements of a business process, or a party and its counterpart, use systems that are designed around the 'hope for the best' principle. Similar to a 1970s auto plant, the approach is 'build, inspect and fix'.

In this 'model', the main flow of work or interaction is followed by a process or procedure that catches inconsistencies after the event. For example, in the context of securities trade processing it is very common to carry out end of day batch reconciliation runs to ensure that trade or trade reference data has reached the designated set of systems. This ensures that the flow towards ultimate clearing and settlement can proceed. When failures or breaks in consistency are discovered, these are:

- **recorded (in a failure queue, for example)**
- **then presented by a secondary repair system on a work-list to a person in the department responsible for fixing the problem.**

Most organizations are aware that the 'inspect and fix' approach is expensive, particularly when processes and interactions involve complex, high-value transactions. Inconsistent business results create massive costs and inconveniences, and can damage customer relations and business reputation. Examples include:

- **the cost of identifying and repairing failures in securities trade processing resulting from inconsistent reference data or partial propagation of trade data**
- **the failure to accurately provision multi-element services in a communications or network grid**
- **the failure to allocate or settle correctly revenues and payments between multiple back-end providers**
- **the inability to create reliable, easily-controlled multi-level supply chain orders — which can lead to manufacturing or delivery delays.**

What is needed is an intra-day, reliable and co-ordinated approach to integration that delivers and ensures consistent results across all systems and processes. Technology based on such an approach should be 'composable' so that different co-ordination rules can be combined to quickly develop new service offerings. This helps organizations build software solutions that implement and facilitate business relationships, inside and across the firewall.

Transaction models for application co-ordination

Application coordination technology based on recent open standards — such as OASIS Business Transaction Protocol or proprietary specifications or such as WS-Transaction/Co-ordination — can be used to:

- **solve the problem of inconsistent business results**
- **achieve well-synchronized business processes.**

Such technology deals with cross-process, cross-service linkage and integration. But crucially this is done at the application level rather than at the data-store level.

This means it is more suitable for loosely coupled environments where systems belong to different processes, departments or organizations. Such environments:

- **often use very disparate technology**
- **frequently use incompatible standards, or fail to implement the relevant ones.**

The main purpose of an application co-ordination product is to ease, cheapen and standardize the elimination of inconsistencies in information held in different systems. At Choreography, we have identified four key patterns or models where application coordination is applicable:

- **co-ordinated dispersal of information**
- **synchronization of contingent actions**
- **co-ordinated aggregation**
- **co-ordinated creation of binding contracts.**

While a solution that provides application co-ordination can be used in any industry, the examples in the following models (for the sake of illustrating the potential in a complex environment) focus on the financial services industry.

Co-ordinated dispersal of information

Under this model, business information is propagated — or dispersed — on an 'all, some, or nothing' basis. The dis-

persing system knows the exact outcome of processing by each recipient. Business rules determine viable outcomes, enabling partial but still acceptable results.

A good example is the propagation of trade reference data such as counterparty information for a structured trade (Figure 5.1). Here, static data can be fed through from front office to settlement systems to avoid downstream fails when a trade is processed. Propagation is co-ordinated and, because the outcome is monitored, there is an opportunity for intra-day repair.

In this example, the downstream systems are participants in the business transaction. If one of these systems cancels a transaction, because it already has the correct counter-party code and has been sent the wrong one, the controller identifies the problem and cancels the whole business transaction. A new business transaction is then originated with the existing counter-party code.

This, simple, case of data propagation illustrates the potential to gain a new level of information about the state of participant systems.

Creating synchronization barriers

In this model, complex, co-ordinated action is contingent on the result of a prior co-ordinated action. Following the same example in the previous model, an organization ideally will not propagate trade data throughout the front office and settlement systems until all of the reference data has been co-ordinated and checked throughout the system (Figure 5.2).

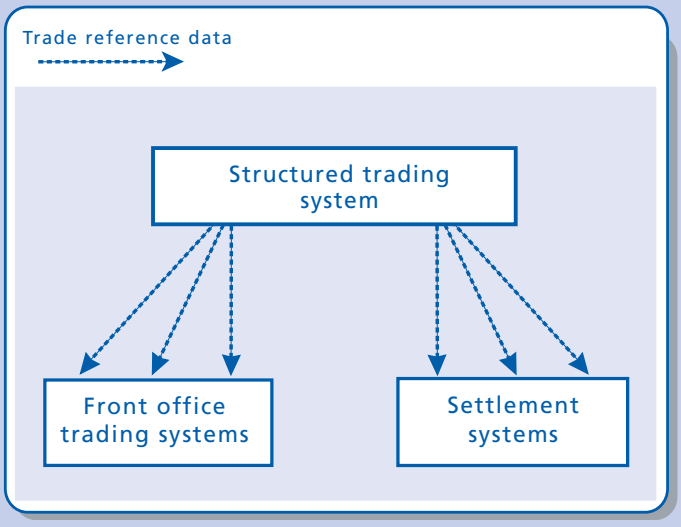
This is particularly important for groups of trades — such as structured products, trades with hedging or back-to-back trades. A synchronization barrier such as these present requires consistency within each system, to remove the risk of partially impacting both payment and risk processes. Addressing the synchronization barrier also reduces the number of trades in the various failure queues.

Co-ordinated aggregation

This is can also be thought of as the escrow model. It appears frequently in credit check or payment-against delivery processes. It involves obtaining information or commitments that build up to a coherent outcome. The receiving system and sending service both know that information has been delivered. The receiving system uses business rules to determine viable aggregations.

A typical example is gathering multiple credit authorizations prior to line draw down (Figure 5.3). Counterparty, country, industry-sector and cross-border limits can be applied before triggering credit. Provisional authorizations

Figure 5.1: Information replication



cause reversible reserves to be allocated against limits, while aggregate authorization triggers firming of reserve allocations.

Co-ordinated creation of binding contracts

This pattern is used for negotiation leading to a deal being struck. Other uses include the sending and acceptance of confirmations. Many trading protocols and information exchanges follow this fundamental pattern.

Acceptance of an offer creates a binding commitment: the parties are bound to deal and acknowledge involvement. Rejection induces counterproposal or re-proposal.

Any outcome is common knowledge, and is mutually assured by comparable records (Figure 5.4). Examples include quote or offer-driven trading in the securities industry or purchasing via online B2B portals.

OASIS Business Transaction Protocol

In June of 2002 the Organization for the Advancement of Structured Information Systems Business Transaction technical committee produced the BTP 1.0 committee specification (www.oasis-open.org). This resulted from fifteen months of collective work in an open process involving significant software vendors such as:

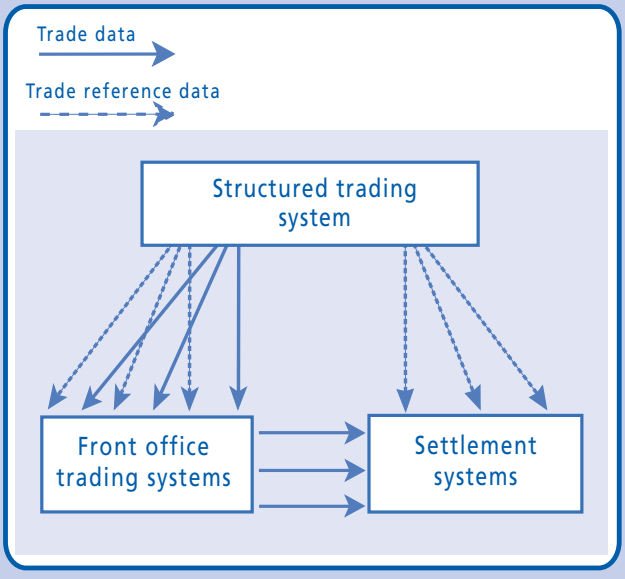
- **BEA Systems**
- **Oracle**
- **HP**
- **Sybase**
- **IONA**
- **SeeBeyond.**

The Business Transaction Protocol (BTP) is the first standard specification for consistency management to target loosely-coupled service-oriented architectures, where minimal contact exists between the systems of departments or organizations. It has been followed by two early draft specifications (Web Service Co-ordination and Web Service Transaction) published by IBM, Microsoft and BEA in August 2002. These specifications taken together are broadly functionally equivalent to BTP, and incorporate its main features. They are currently only available for review and analysis, and have not undergone any standardization process.

Business transaction management is an evolution of distributed transaction management. It uses the well-established two-phase outcome protocol to co-ordinate application or business services, rather than data-level resource managers. In addition, BTP incorporates novel features that enable business rules to intervene heavily in transaction outcomes.

'Cohesions' (cohesive business transactions) enable

Figure 5.2: Synchronization of contingent actions



selections to be made from fluid populations of participant systems. This allows price improvement or alternate path processing to be modeled within a transactional environment. Multiple viable outcomes can then be permitted, so long as critical participants are present.

As such, BTP allows controlled and visible participant withdrawal after declared time intervals, helping to preserve service autonomy and preventing unwanted holds on inventory. It makes explicit provision for 'active phase recovery', thereby enabling very long-running transactions to be implemented while easing the creation of conversations which are both reliable (recoverable) and transactional.

Products based on this specification will help eliminate costly discrepancies and inconsistencies in related pieces of information. In turn, this will enable business services to be created that reliably combine and hide multiple transactional systems.

Organizations will be able to tie the outcome of one event to others and provide for partial or complete rollback. The specification also facilitates provision of immediate information about the failure of an event.

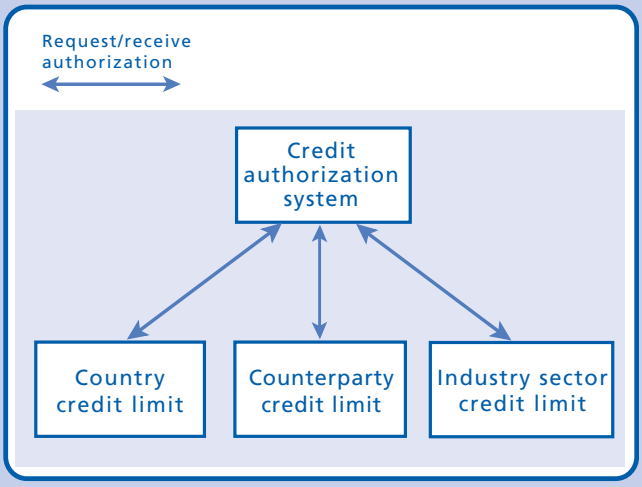
As defined by the BTP 1.0 specification, co-ordinated systems may be owned by the same organization, or split between a party and its counterparties, either point-to-point, or using a hub. Application co-ordination takes place at the level of the application service.

The BTP specification is designed to fit into a heterogeneous environment (which is typically found in large enterprises), where different platforms, languages, containers and distribution mechanisms are used at the same time. Such technology then enables business transaction management in a Web Services environment — but it can also embrace older systems and prior distributed computing standards, such as J2EE or CORBA.

BTP enables application co-ordination by making the transition from unconditional operations to contingent operations. While a business service undertaking an unconditional operation offers but one command — 'do it' — contingent operations have three. The first is a provisional, tentative action that can elicit a response from the other system, preparing it for the next step. This final step will either be a final confirmation or a final cancellation of the provisional action.

A business service implements its own encapsulated concept of the confirmation/cancellation of a provisional operation. A cancellation may invert a tentative operation, or it may utilize a counter-operation which is an alternative to, or a repair of, the provisional work (for example, levy a fee

Figure 5.3: Coherent information aggregation



for a cancelled reservation). The protocol can therefore manage and bridge systems anywhere on the spectrum between full ACIDity and a fully visible transaction/compensation scheme. Partial, uncompleted work may be revealed if useful or necessary.

A complex example

The concept of a contingent operation can be illustrated by the simple example of a buy-side firm sending out a request for quotes on a stock and a related option (Figure 5.5).

At the buy side firm application co-ordination is used by a trading system that acts as a controller. This system determines which quotes should be accepted. This means that the quotes offered by some participating services are to be confirmed, while the remainders are to be cancelled.

The controller (for example, the trading system) first sends out the requests for quote, and the counterparties that are prepared to offer time-lapse guaranteed quotes respond. Their response (including the time-lapse interval) is recorded. This ensures that the participant services will be able to obey the controller's decision to confirm, or to cancel, even in the event of computer or communications failure. The controlling application, in this case the trading system, calls the co-ordination service. The final confirm/cancel decision is recorded and delivered by this co-ordination service. Again, the completion of the business transaction (buy a stock and an option) will be enforced even in the event of network or service failures.

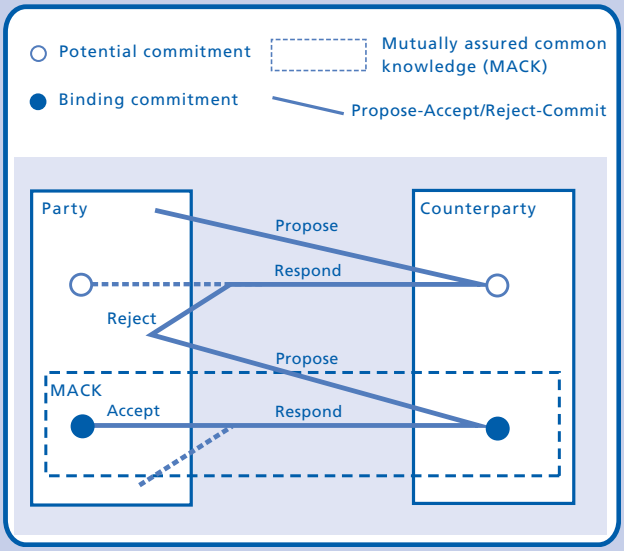
This example illustrates an important new feature introduced in business transaction management software: the ability to apply business rules — in this example, price improvement — to decide which applications should be involved in the final outcome of any co-ordinated action.

Application co-ordination products

From the above analysis, it should be clear that any application co-ordination product will need to provide three main components:

- **a control API, which enables events arising from a business transaction's members or participants to be observed or monitored**
- **a co-ordination service, which is used to create transaction state managers that ensure that business transactions communicate correctly with participants and controllers**
- **a participant framework, which enables some or all of an application service's operations to become capable of participating in coordinated outcomes.**

Figure 5.4: Propose-accept/reject-commit



Within the business transaction environment these components work together to provide the ability to go forward to confirm — or the ability to go backwards to cancel — any particular action, as part of a co-ordinated composite result.

Benefits of application co-ordination

Application coordination can yield a wide range of business and technical benefits. Reducing cost, risk and delays resulting from poor co-ordination and inconsistent processing provides significant benefits for the business. The reliable delivery of new services and products by composition of back-end services is also enabled.

The ability to enforce application-level consistency has another powerful by-product. It becomes possible to expose, in a carefully controlled way, partial or tentative work — intentional data:

- **classic transactions are closed; tentative or provisional actions are invisible until the transaction ends**
- **business service transactions enable MIS, or retry, work to be driven off intended results, providing additional timely data that reflects the actual state of an incomplete business process.**

On a technical level, application or business service level co-ordination based on loose-coupling provides flexible participation and outcome management. By ensuring application consistency, strong business-level links are maintained between two parties.

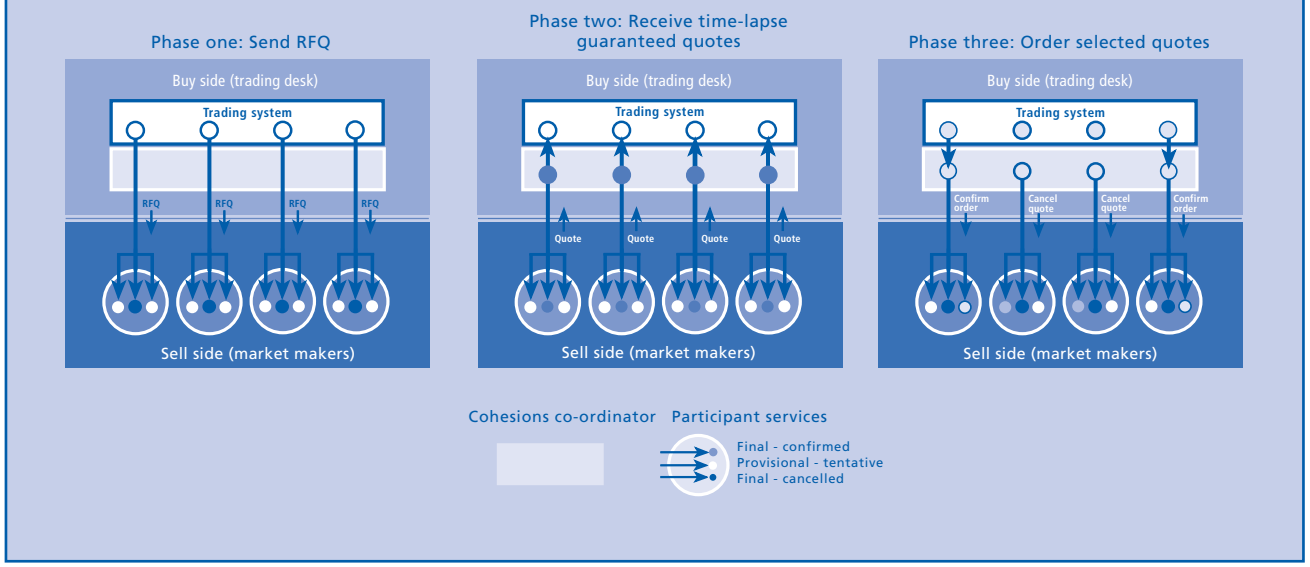
A strong link provides guaranteed delivery and guaranteed processing of a message that causes an operation to be run, and keeps records for both parties.

Management conclusion

Application co-ordination massively simplifies the job of organizing and monitoring co-ordinated activities or business transactions. Self-programming this type of functionality at an application level is hard, and invariably error-prone.

As Mr. Green argues, the use of a carefully written standard protocol (such as BTP), which is the work of a group of industry experts from many vendor companies, enables users to access a systematic, well defined and general purpose solution that is designed for heterogeneous technology environments. Whatever many may wish, transactions are important, a fundamental building block of the business world. Anything that improves their likelihood of success, especially when being delivered between organizations, is going to be commercially attractive.

Figure 5.5: Securities trading example



Middleware and distributed systems: some remarks on future challenges

Peter Bye
Consultant
Unisys Systems & Technology

Management introduction

The past decade or so has seen a rapid — explosive might not be too strong a word — growth in the use of middleware. Middleware products in the form of application servers are now generally used to provide the run-time environment for new applications. Using recognized middleware to integrate different systems, with connectors and wrappers to link applications implemented using different middleware standards, or no standards at all, is also widespread.

This analysis by Peter Bye looks at the questions raised by:

- ***building intra- and inter-enterprise applications by integrating reusable components (as opposed to writing all the code from scratch)***
- ***handling the kind of applications made possible by ubiquitous intelligence in a variety of systems and devices operating in dynamically changing environments, often involving mobile connections***
- ***managing distributed middleware environments***
- ***understanding the performance implications of distributed applications.***

Readers should note that the above are just a selection of the factors likely to prove important in the future; there are many more. Furthermore, several factors may be inter-related. Solutions to one problem area may lead to problems elsewhere. For example, making it easy to build applications from large numbers of dispersed service components may lead to disappointment with performance.

Future challenges and perspectives

We have now reached a level of understanding where we know how to construct certain kinds of distributed systems, for example within enterprises and (to some extent) for inter-enterprise environments. Although the industry's record in software projects is poor, many of the problems are attributable to factors other than middleware choice.

However, the world is not static. Business requirements, the economic climate and technology developments combine to make change inevitable. Responding to these changes will require middleware able to provide capabilities that are not sufficiently developed in current technologies. There will also be implications for the design of distributed applications using these technologies, particularly where performance is concerned.

The middleware bus model

A fairly conventional model, which is referred to here as the middleware bus, is used as a framework for discussing the various questions surrounding distributed systems. In this model, various systems and access channels are thought of as being inter-linked by a software bus — the middleware bus. The various systems 'plug' into it for the purposes of inter-working (Figure 6.1)

In the model, applications are thought of as being constructed from service components offered through an external interface. Each service component comprises a number of software components, which collectively deliver the services. Services are invoked by various access channels and may be delivered by a single service component.

Larger services may also be delivered by combining more than one service component. These service components can reside in a number of different systems, which may be distributed over a number of enterprises.

The middleware bus ties it all together, linking all the various parts. It is a logical construct, implemented within various connected systems, and possibly in a separate tier of integration servers. It may be thought of as a collection of different components providing a variety of functions to its users.

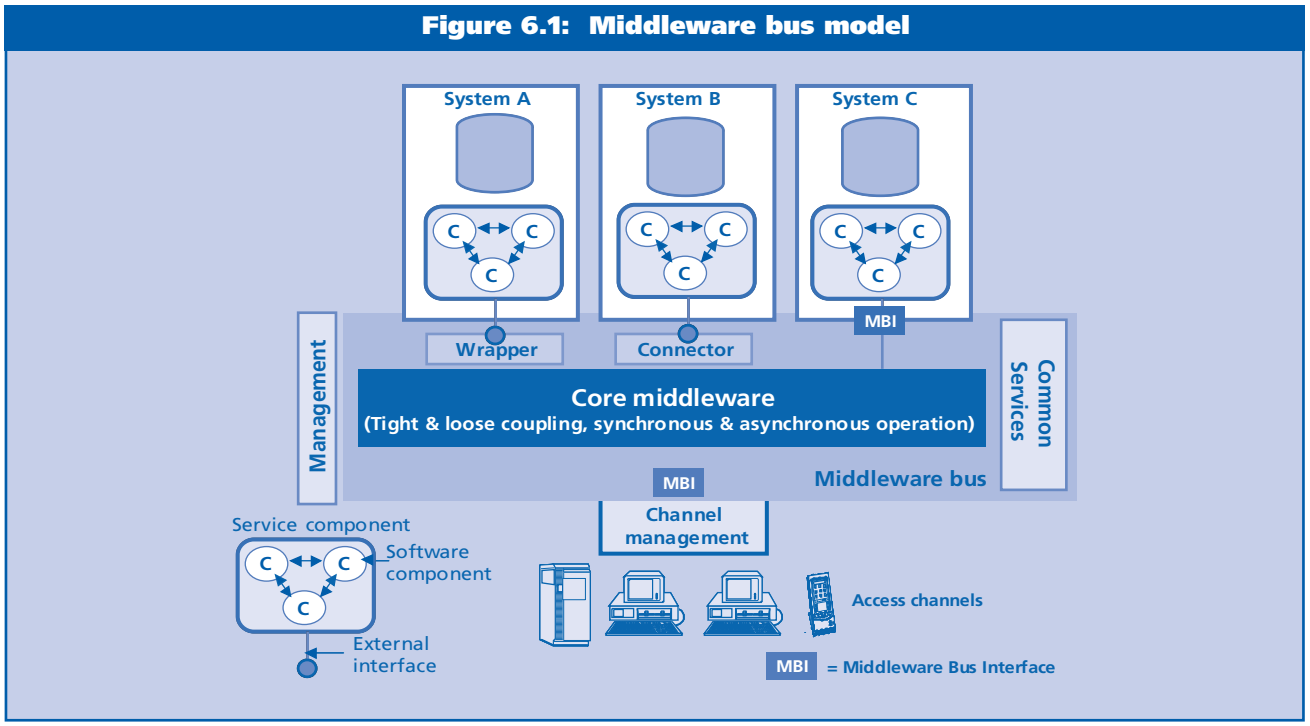
The first component is what is called Core Middleware (in Figure 6.1). This is the basic plumbing which connects the different elements together. It can offer various connection possibilities, as shown, which can include:

- **tight and/or loose coupling**
- **synchronous and/or asynchronous operation.**

Tight or loose coupling

Application servers providing the run-time environment for the implementation of component-based applications

Figure 6.1: Middleware bus model



typically use tight coupling. Components invoke the services of others using a request/response or conversational form of communication. These products generally include the features necessary to support sophisticated functions, such as:

- **distributed transactions**
- **maintaining database integrity across two or more systems.**

They are thus tightly coupled in the sense that the application components must be implemented according the structures dictated by the particular technology. Examples include J2EE, CORBA and, in an earlier technology, the Open Group's DTP.

Unlike tightly coupled technologies, which place specific rules on application structure, loose coupling enables interworking regardless of the way the communicating components are written. It is in this sense that it is a looser form of coupling between components. Examples include:

- **message queuing middleware**
- **SOAP.**

Synchronous or asynchronous

Synchronous versus asynchronous operation is concerned with whether the requestor of a service waits for a reply or whether the operation can be carried out later — using a kind of send and forget model:

- **tight coupling generally supports synchronous operation**
- **loose coupling may support both.**

SOAP, for example, is loose coupling by this definition. But it can use an RPC type of mechanism for synchronous operation. Message queuing with triggers can also produce pseudo-synchronous, real-time interaction.

Figure 6.1 shows three systems — A, B and C — connected to the bus. C is assumed to be able to support interfaces to the core middleware, referred to as the MBI, or Middleware Bus Interface (at least in this diagram). This could be the native API for the middleware, or some higher level of abstraction, allowing the core middleware to be changed. If all the various systems connected to the bus conformed to the same standards, the result would be something like a complete environment offered by one of the middleware technologies, for example J2EE, CORBA and so on.

Not all systems will do this, however, so the bus has to offer other means of access. In Figure 6.1, System A is assumed not to follow any middleware standards; it may be an application implemented using a transaction monitor such as IBM's CICS or Unisys' TIP. In this case, the application can be wrapped to appear to conform. This is shown as the 'Wrapper'.

System B is implemented using a middleware standard but not one conforming to the MBI. In this case, a connector is used, the function of which is to map between different middleware standards. An example could be COM+ to the Open Group's DTP.

The major middleware technologies specify architectures for this common problem and various products provide implementations for mapping between architectures. They generally maintain tight coupling, even though two standards are involved. For example, two-phase commit information can often be passed from one to the other.

Some products also provide mapping between middleware standards and common non-middleware environments. COMTI from Microsoft is an example. It provides interfaces both to IBM's CICS and Unisys' COMS.

The other important element of the bus is a collection of what are called 'Common Services' (in Figure 6.1). These services include:

- **security functions**
- **message syntax**
- **semantics translations**
- **directories.**

Building applications by integrating components

Building systems from prefabricated components, offering specific functions through a well-defined interfaces, just like most other engineering disciplines, has been a goal of the IT industry for most of its existence. Structured and modular programming were the first attempts, followed by notions such as object orientation and supporting middleware infrastructures. The task has proved harder than many expected. Although substantial progress has been made, more remains to be done.

Something like the model described in the previous section forms a foundation for building successful distributed systems — by providing a structure for incorporating the components from which they are made. There are now growing numbers of implementations using this kind of approach. Many are new applications developed to run in

an application server, perhaps with integration of existing systems, wrapped or adapted to satisfy new environments.

In some cases, they are new front-end systems for Web browsers and other access channels, with the bulk of the application and database work being done in existing systems. WebSphere in front of CICS applications is one such example. Similarly, integration brokers are being used in more complex business-to-business applications, in combination with other technologies.

A lot comes down to the question of scale. To realize the vision of implementing large-scale, very widely distributed applications by integrating components, a number of problems have to be solved. These include:

- **developing components which are genuinely reusable in the first place**
- **keeping track of components so that they can be discovered and (re-)used**
- **responding to dynamic changes in the environment**
- **managing the configuration and its operation, including security**
- **ensuring that the result performs effectively and meets Quality of Services requirements.**

In some cases, the difficulties are purely technical, in that the solution lies in software capabilities. In other cases, the difficulties are organizational: the capabilities may exist but persuading people to use them is the problem. In addition, there may be physical constraints in performance, which must be considered in design. The first two of the above points are addressed in the next section; the remainder are covered in subsequent sections.

Re-use

To be fully re-usable, a component must offer a well-defined function, of general value and accessible through a published interface. Such components have been with us for a long time, in the form of basic software:

- **operating systems**
- **database managers**
- **communications systems**
- **storage management systems**
- **etc.**

What seems to be harder is to produce application-level components, which are genuinely and easily re-usable without modification, or configured by using parameters. If extensive changes have to be made before a component can be re-used, it may be quicker and easier to write a new one to do the job.

This problem can only be solved at the time components are designed. Having well-defined functional requirements is clearly essential. If the requirements are not clear, or may be subject to as yet unclear future requirements, the result will be a component tailored to meet a specific need and therefore not re-usable without modification. It is worth noting that some of the general-purpose application suites, for example for ERP, have resulted in much more prolonged implementation times than were originally expected. Much depends on the application domain. The problem is simpler where the domain is well established and therefore has clear rules.

The reality is that there is no substitute for careful analysis and design, based on application modeling and definition of the subsequent software components. The business-level objects forming the application definition must be carefully mapped onto the software-level objects or components implementing them.

To do this effectively is likely to take more time than just developing the application logic for a specific case. The payback will (should) come in subsequent implementations. What has to be avoided is spending the time trying to develop re-usable components but not doing it successfully, thus obtaining the worst of both worlds.

The next point concerns trying to find the components. If it is too hard to find re-usable components, implementers will not bother to look. This problem is made worse if, after an extensive search, the component turns out not to be really re-usable and requires extensive modification. The key to success lies partly in technology and partly in organization and project management. Within a single organization, an effective repository is essential for component re-use, especially if the level of re-use is relatively fine grained.

Management then has to set and enforce standards for its use. Like most management problems, these may be harder to resolve than technical difficulties. Internal politics, for example among autonomous development groups, can make life very difficult.

Another option is to use commercially available components and integrate them into new applications. The suppliers of such components have to design them with a variety of contexts in mind and then publish their availability using a widely accessible repository so that potential customers can find them.

External re-use, as is the goal for Web Services for example, is more complex again. All the questions about developing re-usable components remain. However, the repository is replaced by a directory structure, where services are registered and can be found.

The Web Service standards — such as WSDL and UDDI, from W3C, uddi.org and others — are intended to provide this structure. Interesting questions remain as to how all this will scale up to a size commensurate with the industry hype.

Dynamically changing environments

The next generation of the Web will lead to a massive growth in the numbers and types of device connected in distributed environments. Traditional computers such as PCs and servers, although growing in number, will form a smaller and smaller percentage of attached systems. In contrast, the number and variety of specialised devices and embedded systems will increase.

The result will be a requirement to support the collaboration of large numbers of different devices, many of them mobile and connected through wireless links. Such devices include:

- **PDAs**
- **tablets**
- **telephones**
- **systems embedded in vehicles and aircraft**
- **etc.**

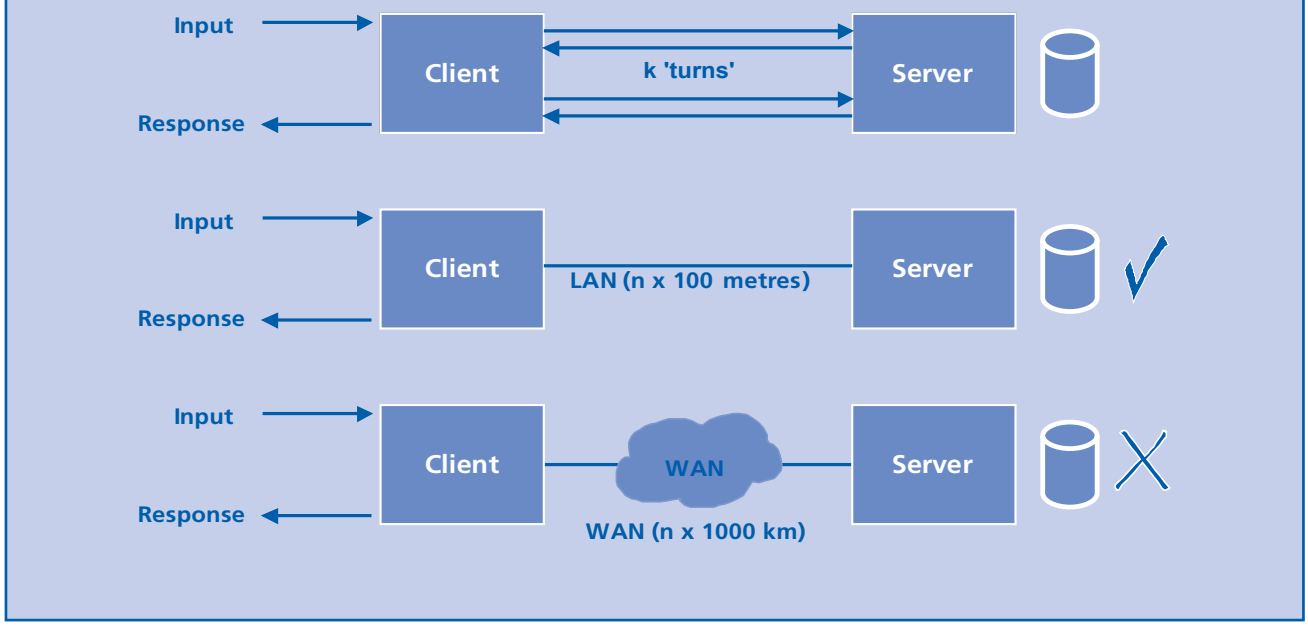
Current middleware does not address all the problems associated with such environments. Traditional middleware will face a number of issues, therefore, including:

- **the need to be adaptable to a wide variety of different systems and devices, some requiring much leaner implementations than others**
- **the capability to change its characteristics for specific operational conditions encountered when it is invoked**
- **the ability to respond to dynamic changes in its operating environment, adapting on the fly to meet evolving conditions while still maintaining an acceptable QoS.**

There has, of course, already been much work done to meet requirements such as these. The explosion in mobile telephone use has driven a lot of the activity — for example, developing standards for software able to operate in these devices and using the relatively limited bandwidth connections available.

Research continues. An interesting set of papers was published in *Comm. ACM Vol. 45, No. 6, June 2002*. An introduction by Gul Agha of the University of Illinois at

Figure 6.2: Latency and response time



Urbana-Champaign prefaced six papers on varying aspects of new middleware.

One general theme in these papers is that middleware must be able to adapt dynamically to changing conditions, either changing itself to meet new circumstances or by enabling applications to inspect and change the underlying middleware.

While one benefit of middleware is that it hides underlying detail (such as networking) from applications, there are cases where it would be of benefit for applications to be able to know about the infrastructure. One example given (in 'The case for reflective middleware', Fabio Kon et al.) concerns a multimedia or videoconferencing application. The QoS can be greatly improved for such an application if it uses a network transfer protocol that suits the underlying network infrastructure. What may work well over a high speed LAN, for example, may not suit a long distance connection.

There are, nevertheless, dangers in allowing applications to inspect and adapt the underlying infrastructure. The design of such adaptable systems must ensure that integrity is maintained.

A further complication is that, in a dispersed environment, there is a danger that different systems could adapt in different ways. A means of avoiding such difficulties has to be built into all the protocols used.

Managing distributed environments

While systems management is required for all systems, it takes on a particular importance in the complex, physically distributed environments which can be created using middleware. If such systems are to be operationally effective, they must be properly managed.

The following are among the leading system management concerns:

- **the ability dynamically to add, delete and change service components, without affecting other services**
- **an environment comprising a number of physical systems, with application components interlinked by middleware over a network, already can be started and stopped carefully**
- **continued monitoring of the distributed environment to ensure that all the elements are functioning correctly, and that the required QoS is being maintained; QoS monitoring requires the gathering and analysis of performance statistics, as well as checking them against specified levels**
- **if an element within a distributed environment fails, it is important to know the wider effects of the failure (what services will be affected? if services are affected, can they be restored?)**
- **managing security, especially in insecure times, is of paramount importance; this must include all the recognized functions such as authentication, authorization and encryption, as well as detecting threats and attempted violations.**

All the above are complex enough in a single enterprise environment. They become more difficult with inter-enterprise environments. They are even more difficult again in the kind of dynamic, adaptive environments of the future.

While the choice and deployment of systems management tools is important, therefore, there are further ramifications to be considered, including:

- **a distributed environment across more than one organization is unlikely to have only one management system; each will be separately managed, requiring some co-operation among the various management systems**
- **the protocols used for inter-organization communication will need to be well suited to the task; the clean separation of the environments, with well-defined rules, is likely to be aided by the use of loosely coupled protocols**
- **middleware must provide suitable agents for reporting management information and should allow user-provided intercepts (or inserts) for specific functions, such as security.**

A final comment on management before leaving the subject: distributed environments, especially involving many independent parties, provide a fertile ground for finger pointing in the event of any problems. It is hard to decide who to blame. Good management tools are therefore essential to minimize the scope for conflict.

Performance

Finally, the performance of any environment is always a concern, as users' expectations become more demanding. Performance was once relatively simple to understand because only a single system would be involved.

Now it is more complicated in distributed environments within a single organization and more complicated again

where more than one enterprise is concerned. The scope for performance problems, for example response times, is greatly increased.

In a recent analysis on Web Services ('*Six things that could spoil the Web Services dream*', **MIDDLEWARESPECTRA** Volume 17 Report 1), Tom Welsh described how performance might yet prove to be one the potential obstacles to realizing the Web Services dream. He cited a number of factors that can lead to long response times, including:

- **processing the invocation**
- **the amount of information in XML messages**
- **the consequences of cascaded services**
- **request latency**
- **and more.**

His comments were, and are, very much to the point. My remarks here are concerned specifically with latency.

Among the factors affecting latency is the bandwidth available. This may be a particular problem with Web Services, as XML messages are long winded. Therefore, what may work well on a high speed LAN could encounter difficulties on a slower WAN connection. Adequate bandwidth may be expensive or even unavailable in some places. An application involving many invocations, or request/ response turns, runs into problems.

Figure 6.2 shows a client invoking a server requiring k turns. As can be seen, it shows that a LAN connection is acceptable while the WAN connection is not, possibly for bandwidth reasons.

But there is an additional factor to consider, which is also shown in Figure 6.2 — the distance involved:

- **the LAN connection may be no more than a few hundred meters**
- **the WAN connection could be any distance.**

While bandwidth improvements over the past few decades have been spectacular, with bandwidths in the terabit range having been demonstrated, one thing has not changed: the laws of physics, specifically the velocity of light. This is just what it always was: 3x108 metres/second in vacuo and 2x108 metres/second in vitrio.

For Europe to Los Angeles, the round trip distance is around 20,000 km. So light in a fibre optic cable gives a round trip time (RTT) of about 100 ms, regardless of the bandwidth. This time is the theoretical minimum possible. In practice, it takes longer, because of factors such as intermediate buffering and occasional loss of packets. Figures from the Internet traffic report show typical RTTs of over 200 ms for North America to Europe and longer for Asia, and so on.

For simple file transfers using FTP, the consequences are that the maximum bandwidth used rises to a peak and then stops, no matter how much bandwidth capacity is available. The principle cause here is that TCP is used as the transport protocol for FTP. The consequences for distributed services over long distances are easier to see. For example, if k = 100 and the connection is Europe to Los Angeles, the complete request will take in excess of 20 seconds to complete, never mind any delays in the systems containing the service.

This means, of course, that great care is needed when designing such distributed applications. If many turns are really needed, either the service should be co-located with the requestor, as Tom Welsh suggests, or the number of turns should be kept to a minimum, preferably 1.

If neither is possible, a long wait should be expected. This also means that, for those concerned, inter-planetary Web Services really are a non-starter.

Management conclusion

In this analysis, Mr. Bye has intentionally raised more questions than he has answered although, with luck, it has provided at least a few pointers to key developments. Mr. Bye believes that we currently have sufficient middleware capability and understanding to enable the construction of many distributed environments, specifically within enterprises and among them, as long as we work carefully and within current constraints. New research and development is likely to solve many of the problems and enable more and more sophisticated environments to be built.

The danger, as it always seems to be in the IT industry, is the belief that there is a magic solution to all problems. Promising technologies are adopted and hyped out of all recognition, with displays of religious fervor and certainty. There is then an inevitable shock when things do not turn out as expected.

This could easily happen to Web Services and some other promising developments. It need not, however, as long as these are approached carefully and with consideration given to addressing questions like those raised above.

Autonomic middleware

Dr Keith Jones
IBM Worldwide Software Solutions

Management introduction

Complexity in enterprise systems is fast becoming a limiting factor to continued growth and exploitation. In this context, IBM has launched its Autonomic Computing initiative that seeks to examine the cause of that complexity and then propose an approach to containing and eventually reducing the mounting cost of ownership for enterprise infrastructures.

Middleware has traditionally absorbed the functional complexity that would otherwise be embedded in business logic. But it too is becoming costly to maintain and operate. The journey toward autonomic computing has begun. The goal is to 'push down' complexity in enterprise systems so that IT professionals can concentrate on creating business value rather than struggling to contain costly infrastructure.

In this analysis, Keith Jones applies the basic characteristics of autonomic computing to middleware components and addresses key questions, including:

- ***can the autonomic computing approach be used to contain and reduce costs?***
- ***how will middleware components change as they become more autonomic?***
- ***how will those changes deliver the promise of self-management?***

Autonomic computing

For decades computing systems have been growing by orders of magnitude in capability, capacity, diversity and complexity. By most yardsticks this growth has delivered a tangible transformation of many aspects of our everyday lives. Although often hidden from view, middleware has played a key role in delivering that transformation.

The cost of technology used in computing systems has fallen steadily as their capability and capacity have grown. Unfortunately the diversity and complexity in these systems has driven up the cost of ownership. The diversity that comes with choice — between technologies, development and run-time platforms and solution vendors — may yet be a limiting factor to future growth.

Most enterprise systems have grown over time through acquisitions and extensions — to become a patchwork of heterogeneous platforms that are fast becoming a nightmare to operate and maintain. While most IT professionals understand the logical layering (Figure 7.1) in those platforms, the reality is often a highly complex matrix of dissimilar components.

IBM has long recognized this reality and is working to produce a long-term solution that will help bring complexity and its associated costs under control and yet continue to promote choice as a fundamental right of ownership. The IBM Autonomic Computing initiative seeks incrementally to deliver a different approach to building infrastructure for business applications.

In an ideal world, (see Figure 7.2) business applications would manage themselves and rely upon an infrastructure that also manages itself. The attraction, if it existed, is that such an 'autonomic' scenario would allow business managers and IT professionals the freedom to spend time and other resources on the 'what' of running an enterprise rather than the 'how'. Today as much as 80% of IT expenditure goes to integration projects, routine maintenance and operations. That proportion continues to rise over time.

Autonomic components

For applications and IT infrastructure to behave autonomically, a number of key characteristics must be built in. The good news is that some of these are already in place at some level in selected hardware platforms, a number of operating systems and various (and increasing numbers of) middleware components.

Reaching the ideal will, however, require invention on many fronts. Nevertheless a series of incremental steps can take us in the right direction. Each will deliver value.

In an autonomic future, infrastructure will be self-managing. Such infrastructure will comprise autonomic components that exhibit several common behaviors. They will be:

- **self-identifying**
- **self-configuring**
- **self-optimizing**
- **self-repairing**
- **self-protecting**
- **self-adapting.**

Research has already shown that software components can behave autonomically using a relatively simple logical design pattern (Figure 7.3). In this pattern the behavior of functional components is monitored and managed logically by a controller component.

In essence this is a feedback control loop. The controller receives data and events from a set of managed resources and, after appropriate analysis, takes action by provisioning new resources or sending commands to existing managed resources. Autonomic behavior can be observed at the boundary of this pattern — at the edge of a component.

Where would we expect to encounter autonomic components? The good news is that the autonomic design pattern can be applied recursively.

At the nanoscopic level, basic hardware components can be managed using this pattern to define higher-level autonomic hardware components. Autonomic hardware resources (for example, CPU, memory, channels) can be managed together with software resources (for example, queues) using this pattern to define autonomic OS platform components (for example, virtual storage).

At higher levels in the functional stack, autonomic middleware components (for example, Web servers, transaction servers and databases) could be defined using this pattern comprising a number of lower level software components. Complete autonomic business processes could similarly be defined using this pattern comprising large numbers of managed resources — hardware, software, data, process and communications. This is the goal for autonomic enterprise computing.

Of course there are limitations. The boundary of an autonomic component is effectively defined by:

- **its configuration of managed resources**

- **the capability of the controller when it comes to recognizing situations that require management action and its ability to take appropriate action.**

How will middleware components become autonomic? What technologies are perhaps already available. What will be required to build such middleware?

Autonomic middleware

It is often said that middleware is the layer in enterprise systems that absorbs complexity — the complexity that would otherwise make business application logic much more costly to deploy. The middleware layer typically comprises large-scale components such as:

- **Web servers**
- **application servers**
- **work flow servers**
- **transaction servers**
- **communications servers (email, messaging, networking)**
- **database servers**
- **etc.**

In turn these large-scale components use algorithms, parsers, virtual machines, queues, caches, connection pools, adaptors and many smaller-scale components to deliver their functionality. The majority of these components require configuration data, performance tuning parameters, startup procedures, backup and recovery procedures, logging parameters, shutdown procedures, version control procedures and many other artifacts for successful operational management. All of these can add up to vast complexity that must be managed in enterprise scenarios.

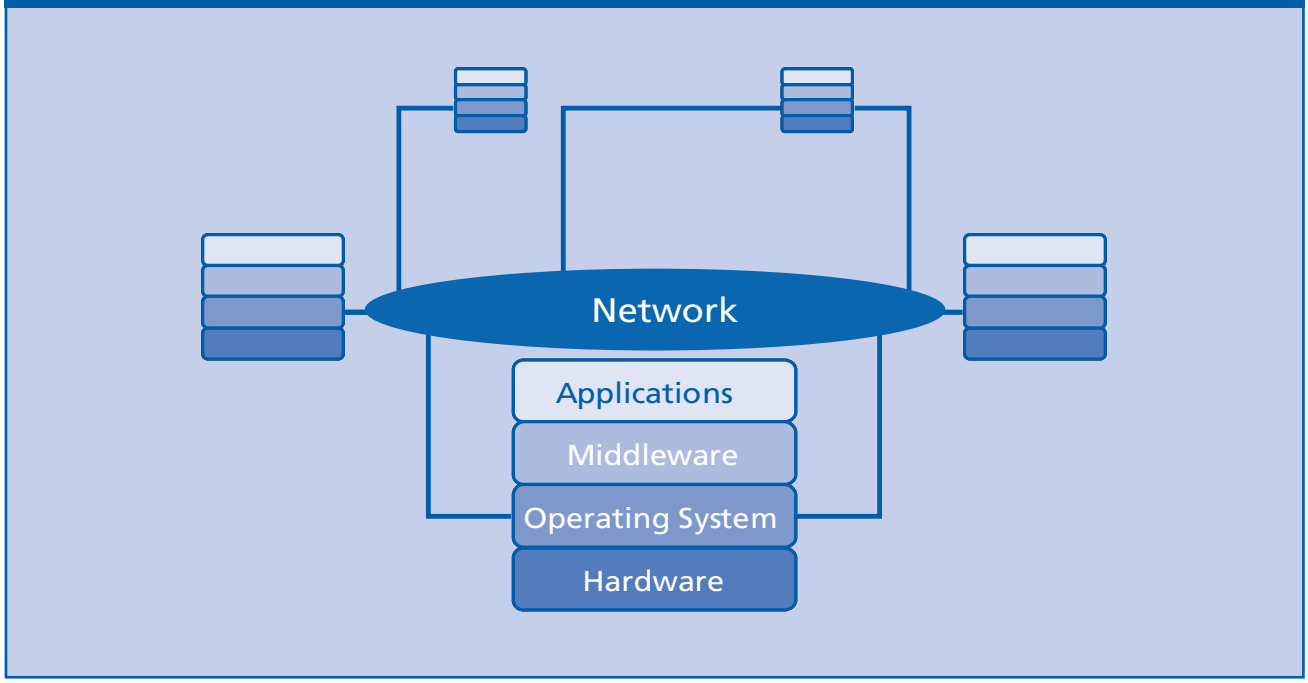
The introduction of autonomy at many different levels within the middleware layer, as well as other layers, could have a dramatic effect on the costs of ownership and deployment. For example, let us look at middleware — in the context of the six 'selfs'.

Self-identifying

All autonomic middleware components must be aware of themselves and their execution context. Each must have a functional type with an identity and an understanding of the dependencies and relationships with other autonomic components in an infrastructure.

This characteristic is fundamental to achieving autonomic goals. Many middleware components in existing systems do not yet have an explicit identity or an explicit definition of relationships.

Figure 7.1: Logical layering



Self-configuring

Some existing middleware systems have hundreds of configuration options and permutations. Autonomic components must configure themselves when installed and be capable of reconfiguring their resources to meet changing needs.

For example, Web servers must automatically configure functional support for cell phone users as they login to enterprise resources. The type of workload being handled (for example, from a cell phone) must be recognized and automatically result in the configuration changes needed. There must no longer be the need for systems administrators statically to pre-configure and later reconfigure such functionality.

Self-optimizing

As configurations morph in response to changing needs, autonomic middleware (Figure 7.4) must constantly:

- **monitor internal efficiency**
- **then optimize resources to meet business goals.**

For example, this might result in dynamically adding and deleting resources, such as cloned server images or networking bandwidth, to process changing workloads whilst at the same time maintaining transaction response times.

WebSphere workload balancing, for example, already has some of this capability built into its network deployment configurations. A fully self-optimizing configuration would recognize comparable (for example, substitutable) resource types with respect to handling specific workload streams and combine this with tactical configuration changes — such as interposition of caches or alternative algorithms — to achieve business goals.

Self-repairing

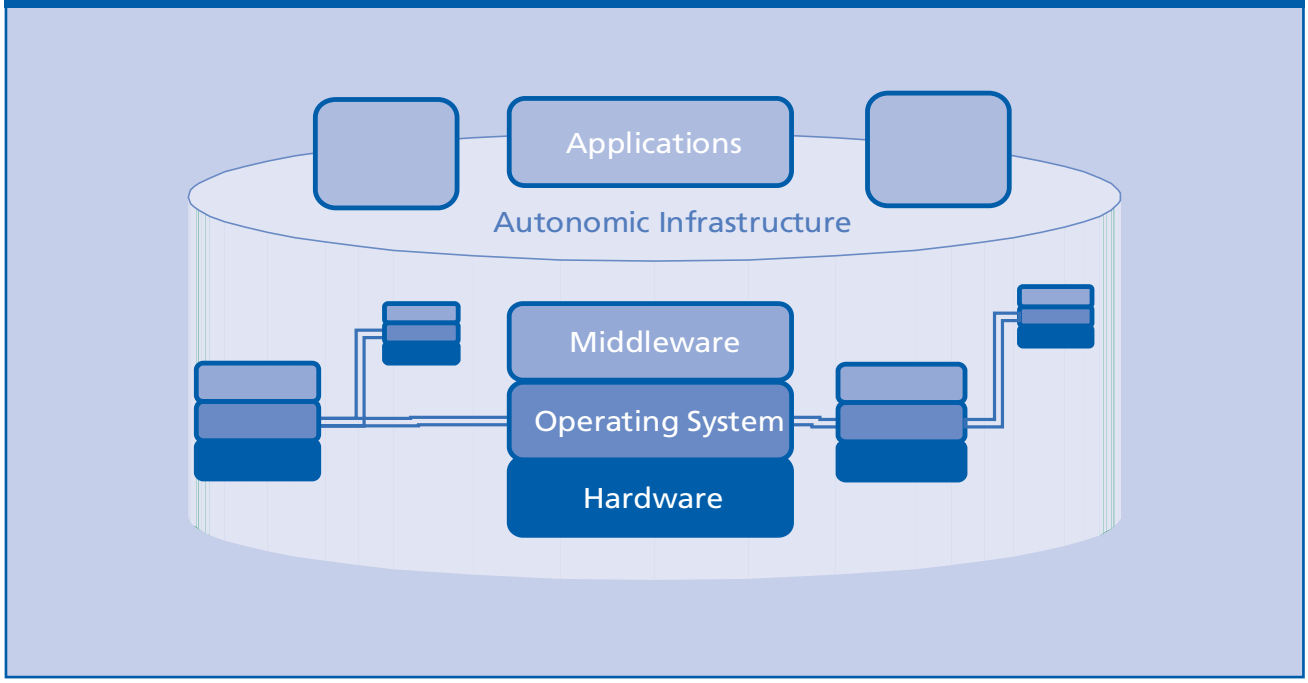
Autonomic middleware components must be able to recognize failure and react to it so that processing continues within tolerable limits. Existing middleware systems — such as WebSphereMQ and DB2 — already have some of this capability, including:

- **re-routing of messages**
- **recovery of database integrity.**

This is achieved by reloading data written to logs or opening alternative channels.

In future, autonomic middleware will not only take the actions already built into existing systems but add sophisticated problem diagnosis. Predictive problem circumvention will be built into middleware components. With this level of autonomy, system administrators would no longer be required to make adjustments. Only in cases of cata-

Figure 7.2: Ideal enterprise systems layering



strophic component failure would human intervention be required.

Self-protecting

As enterprise systems are made available to expanding numbers of end users — such as suppliers, business partners, consultants, government agencies, employees and, of course, customers — there is a greater risk of attacks on system security and integrity. Internet access has already brought with it attack by viruses, hackers and malicious denial-of-service overloading.

Existing middleware systems detect some of these attacks but others are still vulnerable. In future, autonomic detection of malicious attacks by analysis of patterns of access and prediction of unhealthy status will be matched by automatic reconfiguration and policy-based security measures for self-protection of the system. This will depend upon a strong definition (model) of what is normal and therefore safe and what is exceptional and therefore potentially dangerous in terms the autonomic systems can understand.

Self-adapting

In addition to knowledge of itself and the ability to protect, optimize and repair its internal configuration in the face of changing circumstances, autonomic middleware must recognize and respond to external factors. For example, new types of device may be connected to an infrastructure or new types of component added to extend an infrastructure over time.

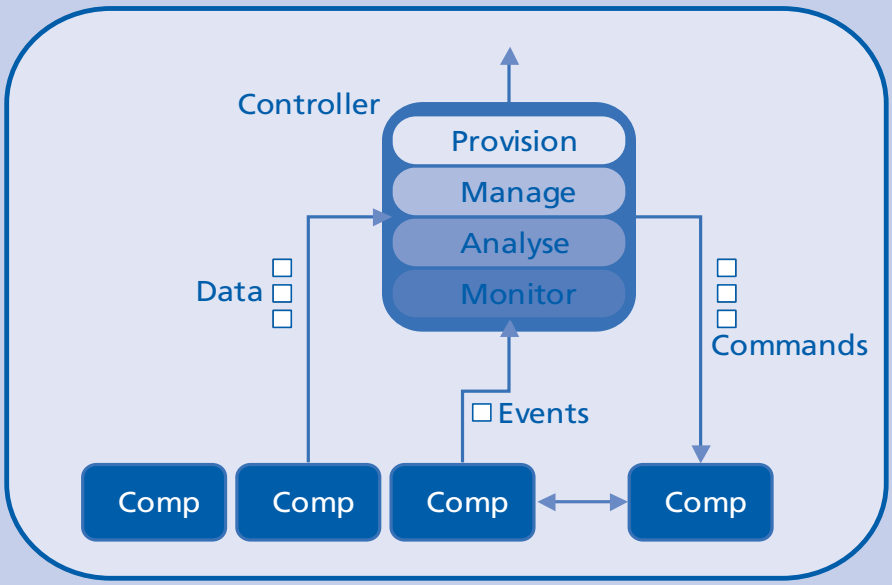
Self-adapting components will know how to change by acquiring new resources from external sources and how to change their external appearance to other components. Such behavior can only be achieved through open standards for external interfaces and data exchange between autonomic components. Autonomic computing cannot be, by definition, a proprietary solution.

The paradox

But how can all this be achieved? What magic is needed to create autonomic middleware components with these characteristics? The answers lie in the design pattern and more specifically in the controller function (Figure 7.5). The controller comprises several functions. These mirror to some degree the autonomic response mechanisms that are believed to operate in the human body:

- **the monitor is low-level and completely automatic**
- **the analyzer is constantly involved in handling routine events**

Figure 7.3: Autonomic management



- **the manager is providing contextual awareness, external representation and supervision for handling exceptional events.**

The 'monitor' is aware of the configuration of resources available to an autonomic component and is constantly collecting data and events from them. Which data and what events are critical questions at this level as they characterize component behavior. For example, self-optimizing components must tune their operations to certain cost-performance goals. Such goals dictate which data must be monitored and which thresholds may cause exceptions.

Monitoring can be costly and so the amount of data routinely collected must be fine-tuned down to the minimum necessary for meeting goals set. As workloads change or resources become stressed more data must be collected and thresholds adjusted. This adaptation to changing circumstances is achieved by analysis of monitored data.

The 'analyzer' is constantly looking for patterns in data being monitored to recognize circumstances under which action is required. Autonomic components come preprogrammed with knowledge of certain patterns and dynamically acquire knowledge of new patterns over time. As patterns are recognized, predictions of future behavior become possible and these in turn become part of the ongoing analysis.

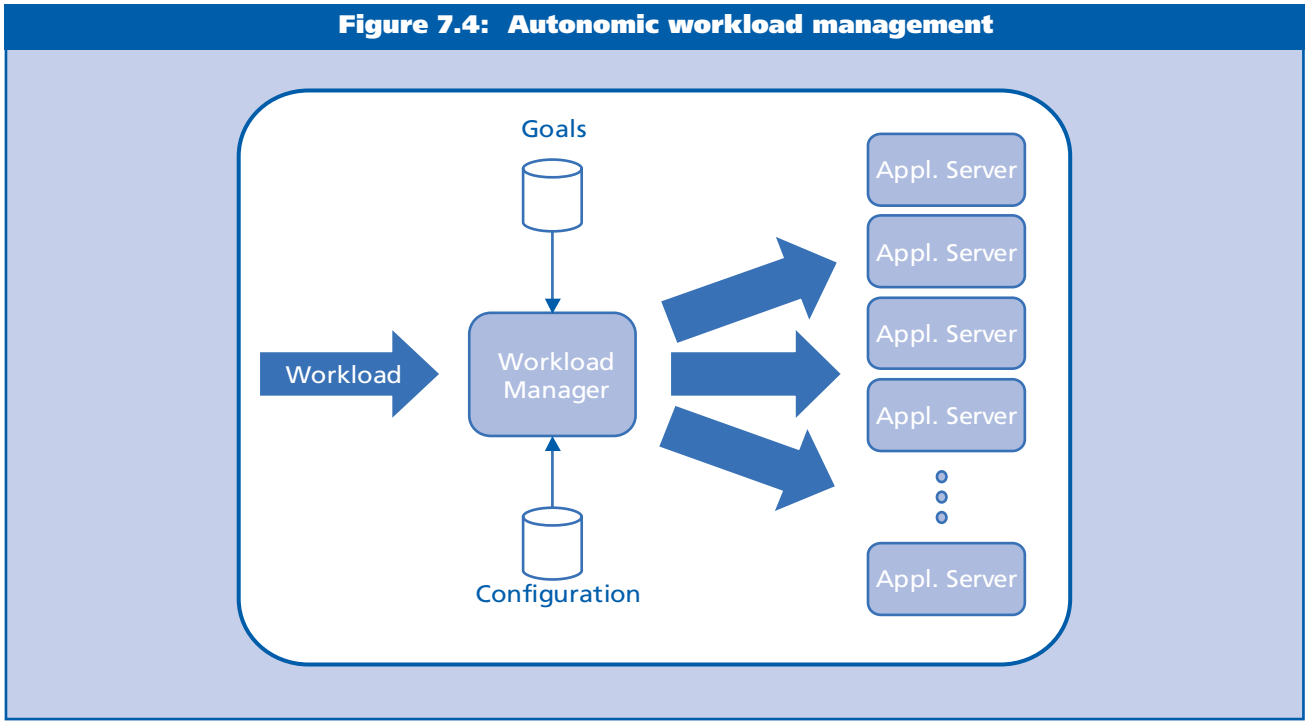
Predictions of behavior are a key input to the 'manager' in an autonomic component. This function is aware of goals that have been set for the component, the availability of resources that may be provisioned and the actions that can be taken. If predictions from the analyzer indicate that goals will not be met then action is taken to make configuration changes that will bring component behavior back into line.

At this level, the 'manager' must maintain an internal model of component behavior under a wide range of probable circumstances. This model will encapsulate a set of rules that dictate what action must be taken in response to changes observed. For example, if a virtual machine is running out of available memory then the 'manager' may provision additional memory from the operating system or trigger the creation of an additional virtual machine to handle increasing workload demands.

This hierarchy of functions — 'monitor', 'analyzer' and 'manager' — is key to providing middleware components with autonomic capability. For small-scale components these functions may be relatively simple. The goals set are straightforward, the patterns of behavior are well understood and the monitoring needed is uncomplicated. Examples of such components might include:

- **queues**

Figure 7.4: Autonomic workload management



- **connection pools**
- **router algorithms.**

However, for large-scale components such as business processes, the goals set will probably be numerous, the patterns of behavior expected will be complicated and many different kinds of data will need to be monitored to ensure autonomic performance in the business context. At this level, human interaction will be required to set realistic goals for resources availability. This will require modeling tools that can understand the functional capabilities of available resources and how these can be composed into higher level, large-scale business processes.

Yet it remains paradoxical that, in order to deliver the benefits of autonomic enterprise computing at the highest level (namely dramatic reduction of complexity costs), the internal sophistication that each component must encapsulate is significantly higher than most components have today.

Is it happening?

The journey toward autonomic computing has already begun. IBM has, for example, already identified five major milestones that most enterprises will pass as they undertake this journey. Starting at the basic (non-autonomic or manual) level, systems will steadily acquire autonomic capability as they progress through:

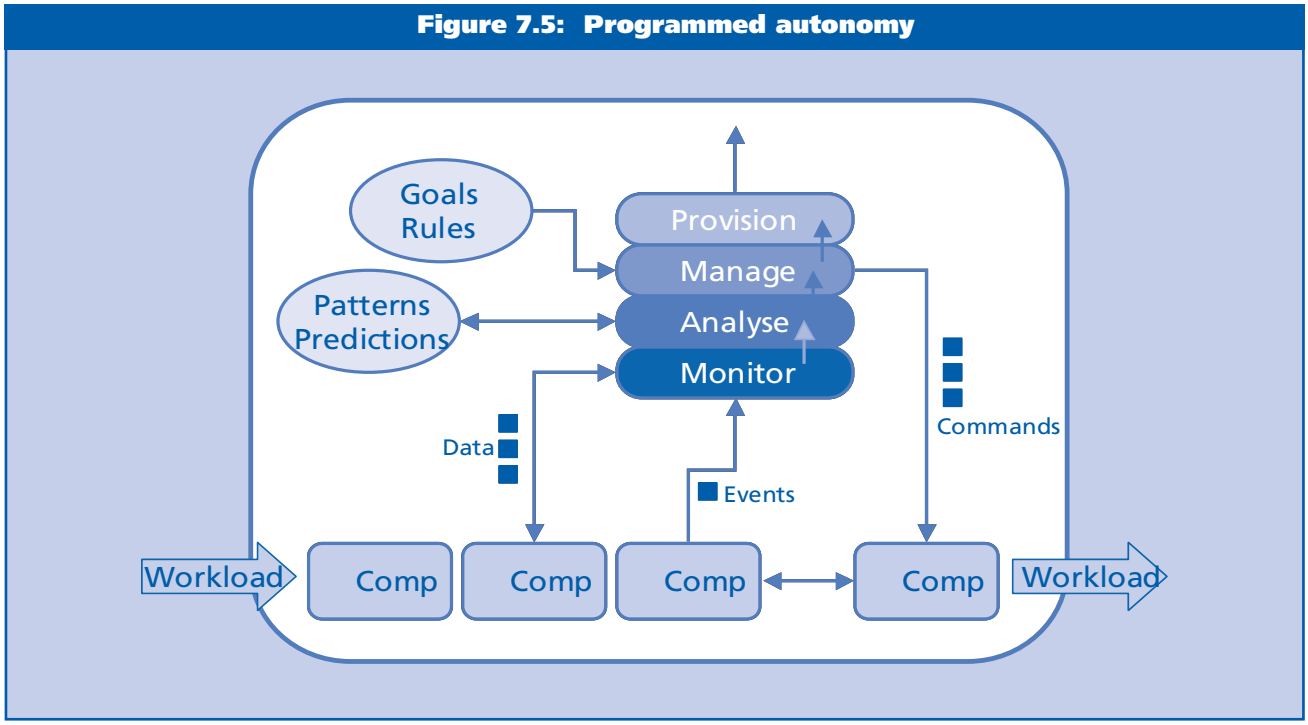
- **the managed phase**
- **the predictive phase**
- **the adaptive phase**
- **the autonomic phase.**

IBM e-Server hardware — such as the p-Series and z-Series machines — now has autonomic components built-in. Already the benefits are being realized. IBM's operating systems, such as AIX with its dynamic partitioning and z/OS with its resource management algorithms, are also becoming autonomic over time.

At the middleware layer, WebSphere, DB2, Tivoli and Lotus products are being restructured into re-usable components, with autonomic features being added. Many of these products already have some of the characteristics needed and others will be added over time. For example, embedded Java Virtual Machine components now have code fragment optimization, memory management and failure/repair mechanisms included.

At the application layer, there is research being carried out that may result in new autonomic programming models for business logic in the future. Microsoft has recently announced its Dynamic Systems Initiative, which focuses on this topic within the Windows platform. Other vendors and academic research institutions have also announced their programs of work in support of the 'Autonomic Computing' initiative. The point is that autonomic computing

Figure 7.5: Programmed autonomy



has started, albeit remaining at an early point in its life-cycle.

Management conclusion

Autonomic computing sounds smart. It has the potential to be this, if its delivery occurs within a fast enough timeframe to avoid being superseded by newer technologies.

What will really make the difference, however, is if broad industry support emerges, with:

- ***extensive research programs***
- ***new open standards***
- ***vendor products that support these***
- ***a significant amount of innovation.***

As Dr Jones argues, this is a challenge. But the potential reward is nothing short of incalculable.

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

USA and Canada:
Spectrum Reports, Inc.

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

UK and Rest of the World:
Spectrum Reports Limited

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

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

Email and Internet

Email:
spectrum@middlewarespectra.com

World Wide Web:
www.middlewarespectra.com

ISSN 1356-9570

[incorporating FINANCIAL MIDDLEWARESPECTRA ISSN 1460-7220]