MIDDLEWARESPECTRA

incorporating FINANCIAL MIDDLEWARESPECTRA

Contents

August 2005

Volume 18 Report 3

Enterprise IT Management — a necessity both for now and the future

John Swainson
President and Chief Executive Officer
Computer Associates

Management introduction

Enterprise IT Management (EITM) is becoming an all embracing topic as well as an increasingly well defined requirement for all organizations — large, medium and small. While EITM is viewed by some as merely being a narrow extension of conventional systems management (which has always tended to be focused on mainframe-type environments), it needs to be much more — taking in all systems, networks, remote devices and software (including middleware) in use within an organization. In this discussion, John Swainson reviews why EITM is needed and how it should be approached.

Mr. Swainson is President and Chief Executive Officer of Computer Associates (CA). Before joining CA in late 2004 he spent more than 20 years at IBM, most recently running all of the WebSphere part of IBM's software business (including the WebSphere family, the MQSeries family and CICS). As such he possesses a background and perspective which understands what EITM will need to deliver if organizations are to manage their business and IT assets into the future.

What is EITM?

Enterprise IT Management — formerly known as Enterprise Infrastructure Management — is about realizing a vision for the future of securely managing information technology environments. At CA we started to understand what was needed to achieve EITM more than two years ago. But, even then, our initial focus envisioned it as 'only' being a logical extension of traditional systems management.

As we researched what was involved we discovered that attempting to extend the traditional approach was not sufficient, that our objective was about much more than systems management or even infrastructure management (which was what we originally thought we were addressing). The notion of managing infrastructure was insufficient because EITM is really the joining of infrastructure management with business process management.

This forced us to expand our thinking. Now we see EITM in different terms. We think of it as embracing a more proactive way for organizations to manage their entire IT environment to support the needs of the business. In this vision the focus will be on managing to the business process with an emphasis on automation, rules and a policy-driven approach. Security will be a fundamental consideration of this approach. This will represent a major change from the way we manage our systems today, where the concentration is largely upon what are 'IT processes', not business ones.

As important as good IT processes are, it is the business processes — order entry, supply chain management, bill to cash, etc. – that are really important to the successful operation and continuation of any business. In this context, service level metrics — and all the associated information about whether or not you are meeting those service level commitments — are now focused on the delivery of key business processes.

Today, many IT organizations are taking a one dimensional approach by managing from the bottom up only — that is to say that the focus is primarily on the operations of IT, including asset and network management. What is lacking in this approach is context — the ability to understand the interdependencies of how the IT operations continually relate to and affect the business process — the top down approach.

What we have found is that you need to be able to 'visit' both the business process level as well as the system level in order to be able to remedy problems in ways that are:

- **clear in their resolution**
- **able automatically to adapt the system infrastructure to changing business conditions.**

EITM envisions an organization being able to manage its own business, including its IT shop, according to that business's particular policies and business practices — instead of according to traditional IT policy specifications, in effect managing top down rather than managing from the technology up.

An architecture

The CA concept for EITM embraces an architectural way of connecting top to bottom (Figure 1.1). To achieve this we desire three stages or layers (from top down):

- **the business process layer (Layer 3 — the Business Services layer)**
- **the service or abstraction layer (Layer 2 — the IT Services layer)**
- **the systems layer (Layer 1 — the IT Operation layer).**

I should also say at this point that we envision orchestrating business processes and business policies in order to manage them: we do not see ourselves as being a creator of business processes in themselves. This role is for:

- **each organization itself**
- **business process specialists — like SAP, Siebel and other application providers**
- **systems integrators, using either packaged or custom written applications using technologies like WebSphere or .NET or any other environment (including legacy ones).**

Individual business processes will then be assembled into composite business processes, via some sort of work flow or extended process management. In our vision, we (in CA) wish to tap into the management points in these processes. This will enable us to map those business processes to the service layer.

This secondary layer is central; it is where one places:

- **information about service level agreements**
- **an abstracted view of the IT infrastructure**
- **the catalog of policies and practices**
- **the description of rules and all related data needed for managing the bottom layer**

- **the associated roles and permissions required for access by various people and applications to the underlying services.**

This bottom layer is, then, the IT environment itself. It is an environment which still has its own systems management and tools associated with it. But we will increasingly characterize these tools, for purposes of differentiation, as being more than just administration.

The key point here is that the point of emphasis will change. Business management will wish to think and act at the top level — with the consequences being the execution or administration (of the results of management decisions) delivered through to the systems level.

Where are we now?

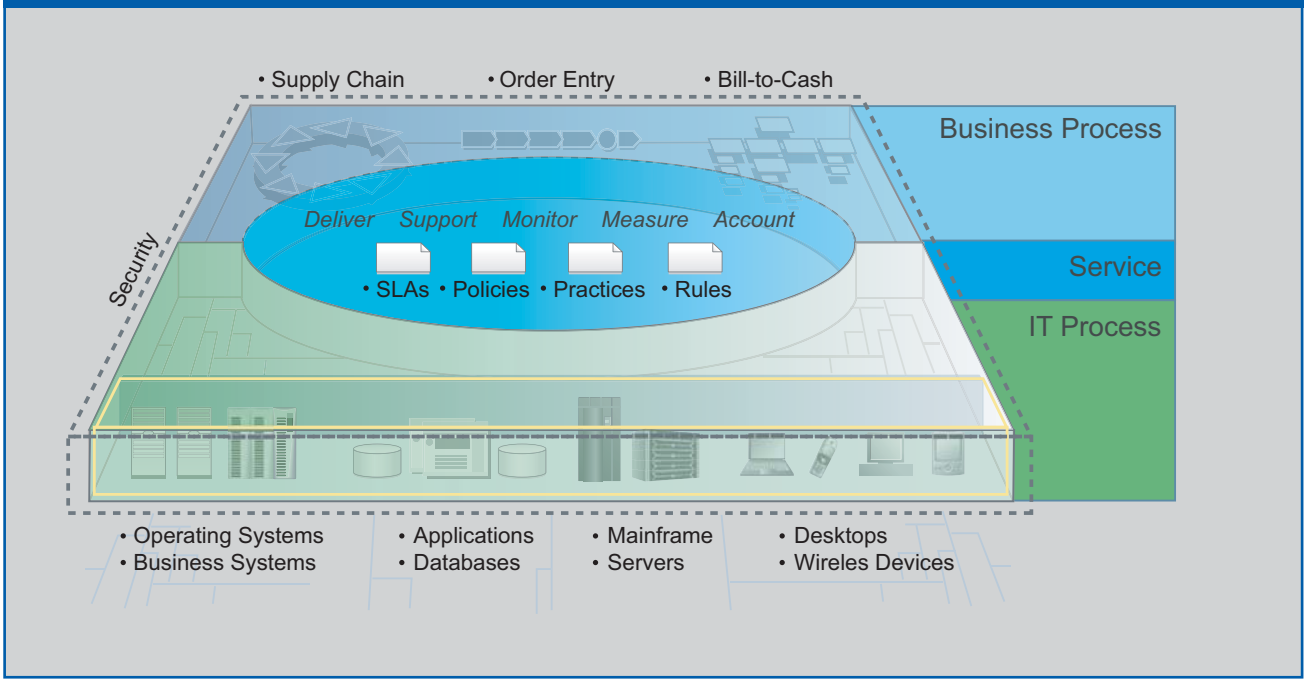
From my perspective, layer one is where systems management vendors have operated for the past 30 years. This has been reasonably well done by the likes of ourselves, H-P, IBM (Tivoli) and a few others. That said, I would argue that the systems level requirement is only partially satisfied today, in part because of the continuously evolving nature of the systems platform environment (blades and wireless being the most recent new hardware technologies, SOA and Web Services being recent software innovations) but not least because there is a long list of supplementary (and often specialty) systems management vendors offering highly specific administrative tools. In effect, most user organizations have a fragmented systems and management environment — which often times results in cost and complexity.

Part of the reason we believe that organizations need to introduce the second layer — the service or abstraction layer — is to start to provide some means of unifying this fragmentation that exists in the systems layer. As such our second layer is likely to revolve around an IT Services foundation (comprehensive management database) which possesses all the information that permits the systems layer to be managed. In addition, the IT Services foundation will serve as the core information model around which EITM is delivered.

I am the first to admit that this concept, in all its breadth, is not new. Even if you add the notion of being able to map the business processes to the service or abstraction layer — this is not conceptually new. That said, the practicality is that today the top two layers are more visionary than reality while the bottom layer is the 'state of the practice'. And, as mentioned, at best it is highly fragmented. What is worse, much of the effort at the systems level has been concentrated on 'large systems' rather than on 'all systems' that exist within an organization.

In some senses this is not surprising. There have not been a good set of standards in place around systems and security

Figure 1.1: Layers of infrastructure management



management— although some are at last emerging. Historically it was the mainframe that established the strongest management environment. UNIX followed, albeit in a somewhat haphazard way.

When one comes to Intel environments, progress has been much less dramatic — even though the numbers of Intel systems (and their cumulative power) dwarf the processing capability of mainframes and UNIX combined. While this situation is improving, it remains true that it is the mainframe environment which still has most of the really robust systems and administrative tools. This must change. As distributed computing continues to expand, and the proliferation of wireless devices and voice over IP continues, the need and criticality for end to end security and management in a heterogeneous environment grows exponentially. IT cannot be aligned with business processes until the entire IT infrastructure is addressed effectively.

Developing the upper layers

Delivering the second and third layers is not going to be simple, or happen overnight. The relevant initiatives will take time to mature. They will not just jump straight out of any magic box. There is much to understand while these are turned into deliverable products.

That said, progress can be made today, which is what we are driving forwards. For example, we (CA) have an initiative related to layers 2 and 3 which we call 'Business Service Optimization'. This involves the extraction of the management layer into sets of services which are backed by the relevant information about service levels and policies. In addition, this work can take account of (say) many of the ITIL processes which are, if you think about them, a set of best practices for managing systems.

What we want to do is:

- **take all these methodologies and codify them, whether creating a set of coded frameworks for best practices that are launching points that enable companies to speed up time to value or simplifying the process of filling out the layers or linking IT services to business processes (which explains our recent acquisition of Niku)**
- **then create linkages into languages like BPEL and use this to open doors for us to connect up to the business processes at the top level.**

Put another way, this is building the middle layer first — in order to be able to accept the input from the top layer in the middle layer which can ultimately pass on the appropriate directions to the lower system layer.

As I said earlier, conceptually this is not original. The notion of being able to connect the top to the bottom has been tried before. But it has always failed because the bottom part is characterized by extreme change:

- **users are added or removed**
- **servers are moved or repositioned or changed**
- **networks are constantly undergoing reconfiguration.**

By comparison, business processes are relatively static.

In prior attempts to connect the top and bottom, what organizations discovered was that they were having to change the relationship between the top and the bottom all the time and it just became impossible to keep relationships straight and consistent. In our model for the future, what we are saying is that we (CA) know that the systems layer will continue to change a lot but that reality does not need to impact the business process layer if we can isolate the constant changes via an intermediary middle layer which simultaneously:

- **acts to support the business processes**
- **understands what is happening (and changing) in the systems layer across the entire enterprise.**

There is an additional benefit. In doing this we will be giving businesses — and their people — a better understanding of what is actually occurring. The whole point is to:

- **return to basic business principles**
- **give people — whether executives, managers or partners or whomever — the capability to understand what is going on in their business**
- **provide insights into what IT is doing and how this impacts the business.**

Example scenarios

There are two scenarios I would like to use to illustrate what I am talking about.

The first is the classic failure scenario. This is where something breaks. The immediate questions that arise are about:

- **what is affected?**
- **how many users are impacted?**
- **is this a mission critical business process or not?**
- **what is the cost of responding?**

These are essentially reactions to an event or events. The actions are caused by a failure which triggers the subsequent activities. That is the way most companies think of their systems management infrastructure today.

My second scenario is the opposite. It is the non-failure scenario where activities are pro-active.

For example, consider what should happen if one sees a rise in demand for orders arriving via a Web site. If an increase in orders is being placed, then knowledge of the business process should tell you that inventory will be needed, additional processing will be required, etc, etc. The issue is how to exploit the relationship between, in this case, the business demand uptick and its impact on the entire business environment.

If one understands what is happening at the business level one can put in place actions like automatically provisioning more servers to maintain Web server performance. One can go further, if you understand how the business process works. You use this information to anticipate downstream activities to ensure that, when the results of these increased orders show up in subsequent processes, the required resources are readily available.

If you understand the relationships, you can anticipate the necessary steps to provision capacity ahead of time and be able to do that according to a set of pre-defined business rules that will enable services level agreements to be maintained. Thus business processing becomes key to success.

Another aspect which I regard as being very important about this middle layer is that this is where the business rules exist which enable the systems layer to react to changes in the overall environment. A number of people have said to me "why does CA still put so much emphasis on its Aion rules engine?" It is because the IT environment of the future will have to be based on automated rules-based processing — if only because this will be the only way you can assure an auditor that an organization's systems are, and will behave, according to approved and formalized practices and policies.

What is driving change?

One of the interesting questions here is to consider what is driving change at this point in time. To me there are a number of factors.

I think the complexity of IT environments has reached a tipping point where organizations are really struggling to manage an endless stream of 'next thing, next thing, next thing'. So much of IT has had to become reactive rather be pro-active. Even with server consolidation and all of the parallel initiatives that IT people are putting in place to try to simplify their environment, the result is still complexity and it requires people to do the managing. With the addition of technologies such as wireless devices, RFID and Voice Over IP, the enterprise continues to get more complex by the day.

On top of this you have to lay increased regulatory demands — such as the implications of the Basel II initiative for financial services, Sarbanes-Oxley and HIPAA (to mention just three). Furthermore, many of these initiatives have significant penalties for non-conformance — which is why organizations are, correctly, taking them so seriously.

The result, that I see, is that organizations are having to go through the process of examining their IT (and other) processes and policies. In so doing they are discovering that they do not already have well defined repeatable processes in place. What they do have is more akin, at best, to a collection of business process band-aids. What they need is to put in place something that is more like a formalized and automated solution.

I find positive reactions to this analysis when I talk to customers. Indeed, as you might imagine, the larger and more complex the environment the more this analysis (and our solution) resonates with them.

I had a discussion with the CIO of one of the Canadian Provinces a few months ago. He described his situation to me. Working for a Provincial Government he has responsibility for everything from hospitals to road cleaning to educational class scheduling and content delivery. Some of these activities, like at a hospital, are mission critical — with highly sensitive transaction processing where any failure to manage the transaction successfully could potentially endanger someone's life. In contrast, the failure to deliver a video stream for a school lesson may be disruptive to that class, but it is not life threatening.

The point he made was that his environment has to cater for the extremes — and almost anything in between. He and his people have an acute need to be able to understand the impact of, say, a network failure. Only when they do understand can they take actions appropriate to the

particular circumstance, which might be as extreme as switching off the school feed in order to ensure that the emergency facilities at a hospital can keep operating:

- **if he harms a school schedule, it is unwelcome but it is not the end of the world**
- **if he impacts a health care system, not only will there be complaints but there maybe lives at risk.**

For a person running a complex service center environment with lots of different clusters of users, the concept that improved knowledge leads to superior and relevant decision making resonates. I think that initial adoption will start with those customers and organizations that possess a complex process environment. Their need is greatest, at present.

Ultimately, however, common sense tells me we need a better way to run IT. The notion of running IT based on servers and storage devices and network devices with individuals (not formalized processes or procedures) being the repository of experience and of business process knowledge is an antiquated notion. It is too vulnerable. That is why, in time, I think almost all businesses will move eventually to adopt the notions that:

- **it is the business process that fundamentally matters**
- **IT is simply there to help efficiently enable business processes.**

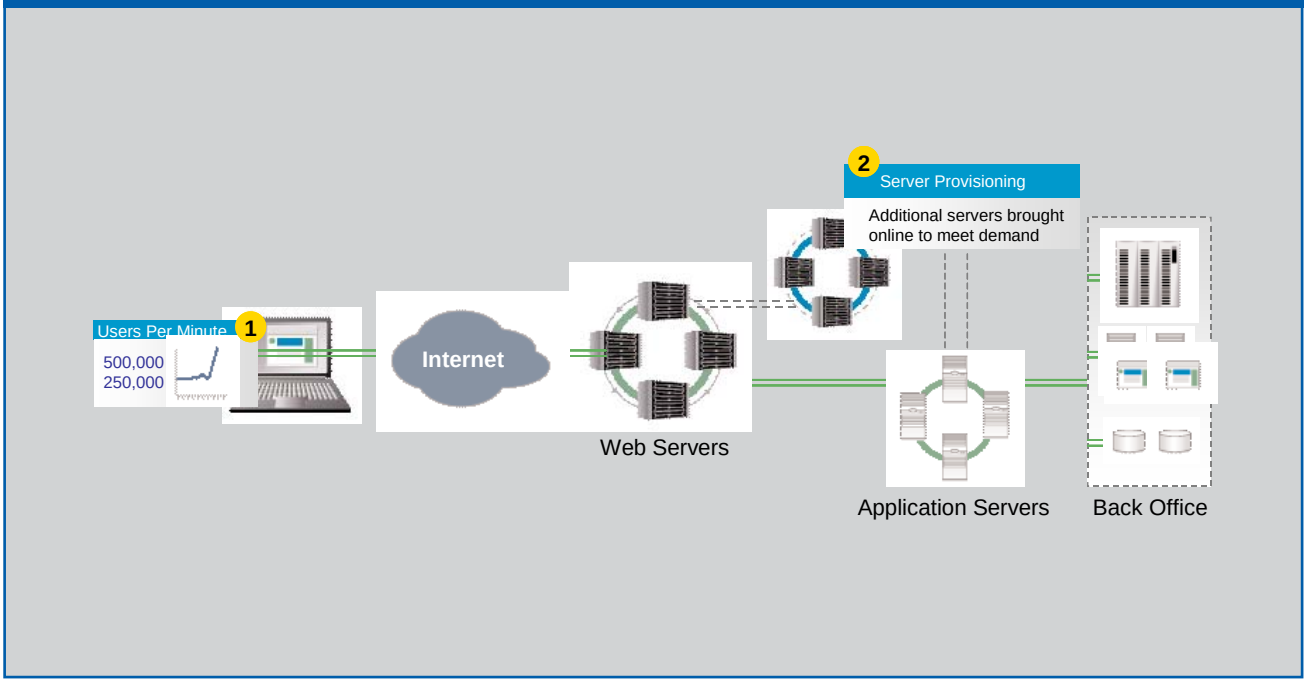
Furthermore, this substantiates a trend that sits behind the way that the IT industry has changed in the last five years. People and organizations are focusing on what they need in a quite different way to that of the past. The desire for technology has gone out the window. The need for results is determining what happens next, not the need for technology.

That said, I suspect there are other changes that are happening less overtly. To me it is gradual transformations which tend to have the greater impact, not least because most of us tend not to notice the gradual transformations as much. I think the move to EITM will be one of these gradual transformations.

How will organizations implement these changes?

From my perspective, I expect most organizations to start to approach these changes through the lens of IT service management — trying to put in place better policies and practices for the IT environment. Specifically, I expect many of them to use approaches like the Information Technology Infrastructure Library (ITIL) — which is a set of best practices used to deliver high quality IT services (and which were derived from 10+ years of experience gained from

Figure 1.2: Automation of provisioning



thousands of IT and data processing professionals worldwide). The attraction is that ITIL has become the de facto world standard for many IT best practices.

In practical terms I see people trying to take and apply best practices initially to cleaning up the mess in the bottom (systems) layer. In so doing they are providing a foundation for what will follow.

That said, what I have been describing for the second and top layers, is — realistically — a decade away from full and general implementation. It is not something that you hire a single consultant to come and do for you. Indeed, it is not something that CA itself will do for you. What we will offer is increasingly more sophisticated and relevant tools to enable others — whether internally within an organization or externally sourced from a third party — to do what is needed. It will be gradual, and it will need to be because we are talking about how to change the way you map actual business processes to IT (and vice versa) at the same time as improving how IT works and functions.

This will be a long and complex process with organizations typically making a start using best practices like those associated with ITIL. As for CA, we will continually update our road map to delivery of Enterprise IT Management.

In terms of the sequence, especially where to start, I think the 15 chapters of the main ITIL book will continue to apply. These vary from the basic — change management and configuration management — through to the more complex, like service level management, security policies, etc. That said, I do think that different organizations will start at different places — depending on how they perceive their own particular problems and opportunities.

So, I do not think there is any one answer. There will be multiple paths forward. What we are trying to do at CA is make sure there is a way for all of these initiatives eventually to converge and the way we will achieve this is to exploit sets of different products.

We will map our products to ISO processes and the methodologies associated with those. We will try to ensure that all of the products underneath have a common schema and use a common repository, while the administrative consoles possess a common look and feel as well as a common user interface. This will be complemented by a common rules engine — so that the parts and products can easily snap together as organizations evolve up the layers towards business process management which works with IT.

Our intention is to manage this in an orderly way. Indeed one of the reasons for buying Niku is to give people the project management capability that can help them define the information as well as manage the project which delivers this. This matters.

The connection point (between business process and IT) lies in the middle layer which is where organizations have to fill in the data about what is where and why, and how it is and should be used. If you do not have this and you do not know what you possess in the systems layer below, you cannot even contemplate sensibly addressing the business process layer above.

How far in five years?

I think in five years we will have:

- **the overall road map**
- **many of the enabling products in place**
- **these products having been built on a common technology base**
- **a relatively small number of large customers well into deployment**
- **still be working with the majority of customers to help them through the first stages of their EITM transformation.**

Many customers are well on their way to implementing this vision; for others I see EITM as being at least a ten-year process. It looks and feels harder to me — and is more transformational — than something like WebSphere, which I was close to for more than five years when I was at IBM.

My reasoning is this. With EITM we will be impacting so much. In contrast, WebSphere was a new standalone environment that had to connect to other parts of IT. But it was effectively new.

EITM by its very nature has to tie together the existing pieces of both business processes and an IT infrastructure. Historically that has been very hard. Is not just a change of systems and management; it is a change of business philosophy into the bargain.

Lessons learned

The one lesson which has made the greatest impression on me is the importance of providing highly specific road maps of where we are going. When we first came out with EITM (even before I joined CA), a lot of customers were confused. They sort of seemed to understand the conceptual

idea that you might manage at a higher level of abstraction — but it all seemed too theoretical and not terribly practical.

What I have learned is the importance of providing practical, tangible examples of what organizations — and people — can do now, and later. The connection with ISO is critical here. It provides a recognized reference point, based on quality.

We have also learned the importance of providing the tools to help people get going. Again, the acquisition of Niku is relevant.

Similarly, standards matter more and more. Frankly you cannot expect to reach any full implementation of this vision if you cannot describe the bottom layer, the systems management layer. Even if what is running there is not CA's, you still have to be able to work with what the customer possesses. Standards are going to assist enormously and those who do not conform will gradually be dropped by the wayside.

Having said all the above, it will probably take longer than any of us would like. I suspect that if you had asked my colleagues here a year or two ago they would have told you that EITM was relatively easy, especially given CA's acknowledged technical strengths in systems management. But creating a unifying view of the information model associated with IT is a complex task. People have tried before and failed. The performance dimension is tricky. The environment keeps changing …

Two years ago you probably would not have predicted the amount of server consolidation that has been going on, the acceptance of blades and the emergence of wireless connectivity becoming real. I recently came back from China. There the Hong Kong Jockey Club is thinking about how to provide the 100,000 people who already have their wireless handheld terminals with the capability to place bets, whether on horses or soccer games or whatever takes their fancy. Necessarily this will need to be offered in a managed and secure environment. EITM seems to be a solution for this, and many other problems.

The lesson that we will have to learn and relearn is that the nature of the infrastructure layer itself is ever changing — even as we try to create an abstraction of it in the middle layer by which to make the connections between IT and business processes. May I put this in a slightly different way?

I have spent many years delivering at the bottom layer. This has enabled me to see the problems that the layers above present, and which the middle layer will be instrumental in solving. I believe we in CA understand the base level environment. I believe we already understand more about managing business processes than most. What we now need to do is marry how businesses work with how systems work. That is the challenge — and not only for CA.

Management conclusion

Mr. Swainson presents a persuasive argument for change and for greater automation, consistency and success. Can CA deliver? If it can it will be because this is becoming CA's primary focus. While there are many capable alternative vendors for specialty parts, most seem to be trying to fight many technological battles rather than to look at the big picture, as CA is attempting.

Systems management was — and still is — CA's strength. If it can adjust itself to address what Mr. Swainson sees as being necessary, it likely possesses the engineering skill and architectural understanding to deliver over the short, then medium and as well as the long term.

SOA: one more step on the road to 'User Specified Software'?

Amy Wohl
Principal
Wohl Associates

Management introduction

Service (or Services) Oriented Architecture(s) (SOAs) are a fashionable subject these days. There is much talk, but not necessarily as much action as many would have you believe.

In this analysis, Amy Wohl examines where SOAs are currently placed, and what may happen next. A key question is whether they really are a step on the long road to a 'User Specified Software' environment.

Setting the scene

Sometimes it feels as if the history of the information processing industry could be divided into periods demarcated by which scheme was currently being promoted for developing software. Always we seem to be an industry in search of ways to make software easier to create, and much faster to develop.

That is because we are seeking to achieve three important goals:

- **being able to react quickly and flexibly to changes within a business organization, with supporting changes in an IT organization**
- **enabling business organizations to drive both what software does and also how much of an organization's IT resources will be allocated to solving each business problem, preferably in an automated way**
- **permitting user participation in the software specification process (perhaps even in the software creation) by providing tools that exploit business knowledge yet do not require programming skills.**

Think of those goals as a Holy Grail that the IT industry has been in search of for years. By definition, that means we have not found it yet.

But we have been looking. Sometimes the emphasis is on assisting professional programmers, with easier to use programming tools and schemes to architect software in a formal way and re-use code. You will remember Object Oriented (OO) Programming. I remember not only the rise of OO (the coding concepts and methods survive, of course), but also any number of failed schemes to manage and re-use a diverse selection of objects — and even elaborate schemes to create markets on which to sell those objects.

Integrating business knowledge with programming skills

At other times, the emphasis is on trying to figure out how to integrate the skills and knowledge of:

- **business professionals (who know everything about the business but nothing about programming)**
- **with the skills and knowledge of programmers (who know quite a bit about how to write good software, but nearly nothing about the businesses they work in).**

This problem has bred an amazing variety of solutions. Some emphasize tools for capturing business knowledge and passing it on to professional programmers, who will then write the software. Many business process modeling schemes fall into this category, together with methods for providing definitions for the tasks being modeled.

Others try to provide tools which enable the business professional to have one view of the problem (a non-programming view, to be sure) and then send that view, perhaps augmented or filtered in some suitable fashion, to a programmer who can then use high level tools (lots of dialogue interfaces and pick lists) to create the software. New versions of tools from most sophisticated development tools providers, such as IBM's Rational, seek to provide the means for every member of a team so that, with suitable integration among the inputs and outflows, productivity can be maximized.

Still others focus on the business professional as the user and attempt to provide tools which will allow him or her to specify a solution and pull previously written software elements together without writing code. Such 'programmerless programming' is a worthy goal, but it has rarely succeeded. The resultant tools are:

- **either limited in scope (they can do some things, but not all the things the business professional might need)**
- **or they require that the non-programmer acquire substantial understanding of programming-like skills (if you do not believe me, I refer you to 3rd and 4th GLs and Natural Language Programming — see also Figure 2.1).**

Web Services are a more recent candidate for creating more granular pieces of code ('services'), together with schemes for storage, retrieval and re-use. These have been interesting mainly, so far, for their abilities to wrap legacy applications in XML wrappers, thereby allowing them to be more easily integrated with other software. In vertical industries, where services have been created that are specific to tasks in that industry (insurance, for example), user organizations have also been able to speed development time by adopting the Web Services model.

And now there are SOAs

In the meantime, along comes SOA, allegedly an improved approach to Web Services. I know that it is often poor form to quote from your own writings, but I am unable to resist putting in a short bit from my IBM DeveloperWorks blog that seems relevant here.

"At any moment in time we have tools that reflect the problems we're trying to solve and the technology that's currently available to solve them. It's important not to think of this as a permanent situation. Both the problems and the technology are changing all the time so, of course, the tools are going to evolve, too. We have to keep up with all this and not get frozen in some prior solution state." [Amy Wohl's Blog on IBM DeveloperWorks — http://www-128.ibm.com/developerworks/blogs/dw_blog.jspa?roll=-2&blog=455 May 26, 2005.]

You will want some context. I wrote this during the Rational Developer Conference, which I was attending and blogging. My careful statement above is really a distillation of what Danny Sabbah, Rational's new General Manager, said at a press conference after his keynote presentation. He pointed out that nothing, not even something as smart and appealing as SOA, would last forever but rather that it was the solution for now — to be succeeded five or more years into the future by something better.

The trick here is to understand:

- **what any 'new, new thing' offers**
- **whether this is likely to be of real value to your organization**
- **where it (the 'new, new thing') is in its cycle of development and adoption.**

It is the third point that is the killer. If it is too early, the technology is likely to be less than fully developed. You can try it out, but many of the pieces will be missing, or at least will not yet be well refined. More to the point, the ecosystem of supporting tools, training and skilled professionals you may require will not yet have appeared in the marketplace.

Conversely, if it is too late, and you are unlikely to obtain sufficient from your investment before something else comes along and looks better.

A study of IT perspectives on SOA

The British research firm Quocirca Ltd. recently spoke with 1,356 IT professionals — worldwide — and asked them about their knowledge of, and plans and experiences with, SOA in a study called 'SOA: Substance or Hype?' About 35% of those who responded had looked at SOA in a detailed way and nearly all of them (99%) believed there were significant benefits to be gained using an SOA. Some of the benefits they cited (hoped for) included:

- **faster development**
- **improved deployment and maintenance**
- **reduced integration time and costs**
- **enabling IT to be more responsive to the business.**

Figure 2.1: Natural language programming evolution

Name	Concept	Fate
3rd and 4th GLs	Query Languages for Business Professionals	Strict Syntax limited their use to small numbers of frequent users.
Natural Language Programming	Users could write programs in ordinary English (or French or German . . .)	Reality could never meet user expectations. There are always rules and limits, depending on how much AI is built into the software and how much processing and memory is made available. Of course, we are getting better at it over time.
Object Oriented Programming (OO)	Separate Data and Program; Reuse Code	OO Lives on. Code re-use has always been sparse.
Business Process Modeling	Tools for modeling business processes and/or work flows.	An important part of Web Services and SOA
Web Services	Creating applications as more granular services (or componentizing existing applications into services)	In Progress -- Slowly
Service Oriented Architectures (SOA)	Combining componentization of software with Web Services interfaces	In Progress

Across all of the 1,356 study participants, nearly 45% are active in SOA now (18%) or plan to be active within the next year (25%).

Study participants also believe that:

- **the implementation of SOA for key commercial software applications benefits interfacing and integration**
- **Web Services facilitate the connection of user organizations' systems with partners, customers, and/or suppliers**
- **Web Services and SOA together make it easier to integrate in-house and hosted applications.**

Nevertheless, adoption rates are still relatively modest, especially in Asia Pacific. Quocirca believes that most users are still only at the beginning of the SOA process. (You may get your own copy of the Quocirca study by going to Quocirca at: http://www.quocirca.com/ reports_soa.htm)

Why should every firm (in a regulated industry) create their own reports?

I spoke with David Kelble, Vice President of Corporate Systems at American Business and Financial Systems of Philadelphia, PA — a mortgage company. Kelble noted that his personal knowledge of SOA was at the 'reading' level these days, but he could see the use of Web Services and componentized applications growing. He felt that a company like his would be interested in SOA over time.

For example, he pointed out how — in his industry — everyone makes extensive use of government forms and reports to the government using these. There is no reason why each firm should do their own programming to create these reports. As examples he identified the IRS 1098 and 1099 forms for reporting payments to contractors and the HMDA reporting to the federal government (this provides then demographic statistics for mortgages, to ensure compliance with non-discrimination regulations).

How open is open?

One myth often associated with SOAs are that they are entirely based on 'open standards'. While it is true that underneath an SOA are Web Services based on Open Standards, building composite applications out of the (hopefully) re-usable services calls for someone's specific development environment. The reality is that each major offering for this has its own eccentricities. It is possible to do integration across services developed in different development environments. But it is definitely easier to work with services that were all developed within (or for) your specific development environment.

That means you may want — or have — to choose a single development vendor for your organization, and trying to take everyone's requirements into account. But it is impossible to dictate what everyone in an extended environment (partners, suppliers, customers) uses. If this is your reality, you will need to be prepared to spend much more time and effort in the integration process and on tools that are designed to support multiple platforms.

Management conclusions

That said, SOAs are catching on. They are not the cure-all for everything — any more than any other technology can be — but they are appealing. It is wise, however, to remember that SOAs are really only the latest offering in a long line of technologies, each of which hoped to make it easier for business users to communicate with application developers and, therefore, easier to develop good software.

Similarly, it remains true that, in IT, major investments stay around a long time, often much too long. Such investments haunt those who made the decisions — and those who have to live with those decisions — along with elusive thoughts of better decisions that might have been made in the past.

This, of course, can produce a fool's game. Technology is not something with a beginning and an ending, but is rather a continuum. There are better times and worse ones in which to make investment decisions. But all of us must make decisions nearly every day. What is important is to learn how to make the best decision for right now — and how subsequently to change your mind gracefully and transition to a better alternative when it inevitably comes along.

The Enterprise Service Bus: a re-evaluation

Tom Welsh
Consultant

Management introduction

The Enterprise Service Bus (ESB) is being talked up in some quarters as an essential prerequisite to deploying robust, reliable SOAs. Yet most ESBs are proprietary, and differ widely from one another; they make use of standards like XML, SOAP and Java, but do not themselves conform to any industry standard.

ESBs are paradoxical in other ways, too. For instance, most of them employ the Java Message Service (JMS) as a secure, reliable messaging backbone. Yet this choice cuts across the open, vendor-neutral nature of Web Services and SOA. For, as we shall see, JMS only imposes a standard API for Java programmers; it does not guarantee interoperability between different vendors' messaging products.

Are ESBs the only safe way to adopt SOA? Or are they just a transitory stage on the way to open, fully standards-compliant SOA? And must all ESBs use JMS for their messaging infrastructure? These questions are of pressing practical concern to everyone planning to deploy SOA.

What is an ESB?

The term 'ESB' first came to public attention in 2002 when Gartner Group analyst Roy Schulte defined it as "...[A] new architecture that exploits Web Services, messaging middleware, intelligent routing, and transformation. ESBs act as a lightweight, ubiquitous integration backbone through which software services and application components flow". At that time Gartner forecast that most large enterprises would have deployed an ESB by 2005.

Dennis Byron of IDC took a much more conservative view. "We don't think [the shift from Message Oriented Middleware (MOM) to ESB] really reflects anything new, as [ESB] is a form of MOM from a computer science point-of-view — once you get beyond the marketing hype."

These statements are not quite as contradictory as they might sound. Byron's comment that ESB adds little by way of technical innovation is compatible with Schulte's belief that it is an important new architecture. Indeed, it is preferable — although not always observed — to base any important new architecture on tried and tested technology.

Nevertheless, the ESB continues the software industry tendency to use vague and confusing buzzwords, following such illustrious predecessors as EAI, Web Services, SOA and BPM. None of these seems able to be explained in clear, simple, unambiguous terms; and yet most leading vendors persist in insisting that they are indispensable.

ZapThink analyst Ronald Schmelzer points out that whenever a new buzzword appears, everyone tries to jump on the bandwagon. "Companies that have traditionally been in the Enterprise Application Integration (EAI), Message Oriented Middleware (MOM), and object broker markets are increasingly calling upgraded versions of their existing products ESBs" he warns. "Is it really the case that ESBs don't represent a new technology approach or market, but rather the natural evolution of existing middleware infrastructure as standards-based, loosely coupled computing approaches take hold?"

Peter Linkin, senior director of product marketing for BEA's WebLogic, explains the logic behind ESB as follows. "What was needed was a system that just told us what the message was, what the contents were, where it should go, and what the quality of service for it should be. Then, that was handed off to the next level which is something like a central post office. The postmaster says 'send me all your messages from all these outpoints, and I will intermediate. I'll make sure they get sent individually, and reliably to all the end points.' It's a message broker that is driven by business process. The end points don't have to know about each other so there's ignorance at each end and the logic of the business process is in the plumbing."

ESB judgement criteria

Some experts have gone so far as to publish criteria for judging what is, and is not, a true ESB. (Needless to say, their own companies' products usually qualify comfortably, while their competitors' appear marginal, at best.) The outstanding example is David Chappell's article 'ESB Myth Busters', in which he puts to the sword no fewer than 10 'myths' that misrepresent the true doctrine. Chappell is a VP and chief technology evangelist at Sonic Software, which claims to have shipped the first production-quality ESB in March 2002.

Cape Clear offers its Web-based 'ESB Truth Test'. This consists of a short questionnaire, which highlights the shortcomings of ESB products based on answers to a dozen or so searching questions. As might be expected, some of these appear to be self-serving; for example, "Can you download and independently evaluate the ESB software to take this test?" Whatever the policy of rival vendors, this assures us we can download and evaluate Cape Clear 6.

In the absence of anything resembling an ESB standard, it is difficult to reach any conclusions as to what the 'true' ESB must include. As an uncontroversial starting point, let us call it 'a form of middleware that can be deployed throughout (and between) organizations, and that provides queue management, guaranteed delivery and controllable quality of service (QoS)'. Figure 3.1 shows a schematic overview of such a typical ESB.

It is no coincidence that this simple definition consists mainly of characteristics that a typical SOA lacks. An SOA is certainly middleware, and can be deployed within (or between) organizations. But it usually does not have queue management, guaranteed delivery or other desirable QoS features. Even if it does, they are likely to be implemented in non-standard ways, making it difficult to impose any degree of uniformity across the whole SOA.

Mainframe-centric computing — for decades the dominant IT paradigm — was highly centralized. It had — and still has — significant merits, such as simplicity, uniformity and efficiency. In the 1980s and 1990s, however, the newer 'distributed client/server' paradigm became more popular, due to its superior openness, flexibility and apparent cost advantages. But the pendulum swung back when users discovered the full costs — including opportunity costs — when confronted with a plethora of distributed open systems.

SOA takes decentralization further than ever before. That is fine in theory, but poses many tricky problems in practice. Hence the latent demand for a form of 'packaged SOA', with:

- **a single supplier**
- **a single predetermined architecture**
- **a single point for management**
- **a single focus of responsibility.**

This is where most ESB vendors see their opportunities.

Why do we need ESBs?

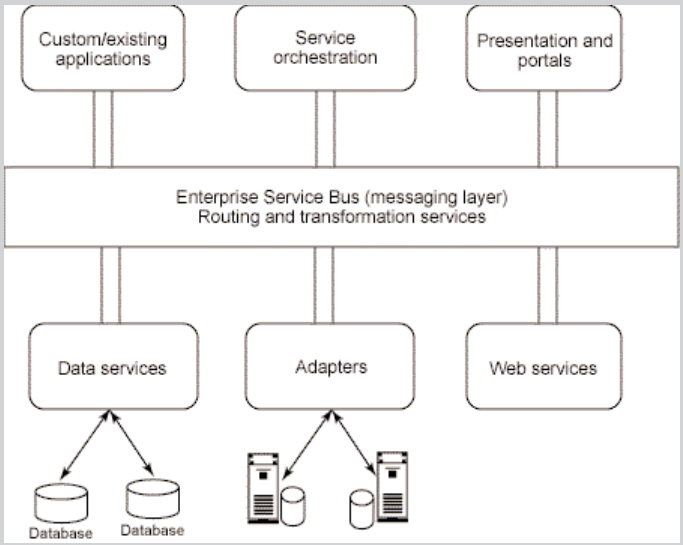
One school of thought holds that ESBs are little more than a marketing slogan got up by Sonic, which thereby created a new market segment (in true Ries & Trout style) and partially broke IBM's stranglehold on the MOM segment. IBM's WebSphere MQ (formerly MQSeries) certainly is dominant, with 12,000 customers (according to IBM) and 75% market share in 2004 (according to IDC). Similarly, upstarts like Cape Clear can challenge big, well-established vendors like SeeBeyond, TIBCO, Vitria, and webMethods by changing the rules and undercutting prices.

As Andrew Binstock, principal analyst at Pacific Data Works, wrote at the end of 2003: "The principal companies pushing ESB are JMS vendors trying to break out of the commoditization of the JMS market. Companies with excellent solutions in enterprise infrastructure — IBM, TIBCO and the rest — are ignoring ESBs".

Software vendors with middleware portfolios can be divided into three main groups, according to how they view ESBs:

- **'ESB boosters' — those vendors that that have made a deliberate choice to pin their marketing colors to the ESB mast and which energetically position their flagship products as ESBs; obvious examples are Cape Clear, Fiorano, Iona, PolarLake and Sonic**
- **'ESB conformists' — those vendors that initially ignored or belittled the ESB concept, but have now re-evaluated the situation and decided to make some concessions to the ESB movement; suppliers like BEA, IBM and webMethods fall into this category**
- **'ESB laggards' — those vendors still holding out against the ESB tide, and either ignoring or criticizing the concept; the big EAI specialists (such as SeeBeyond, TIBCO, Vitria, and webMethods) used to belong to this group but recently some of them have shown signs of wavering while Microsoft is a special case (both BizTalk Server and the upcoming Indigo have been described as ESBs, but the Company itself has had little to say on the subject).**

Figure 3.1: Top-level overview of a typical ESB (source: IBM)



Pattern or not?

One of David Chappell's 'ten myths' is the notion that an ESB is merely an abstract design pattern, rather than a product that can be purchased off the shelf. He joins Gartner's Schulte in vehemently rejecting this view.

On the other hand, IBM — which does not have an off the shelf ESB yet — argues energetically for the 'ESB-as-pattern' viewpoint. Indeed, IBM goes further. Steve Mills, senior VP and group executive at IBM Software, maintains that he has been delivering ESB functionality for many years. The core product in IBM's 'existing' ESB package is WebSphere Business Integrator (Figure 3.2).

In this, WebSphere MQ provides the transport capabilities, while other pieces like WebSphere MQ Workflow and WebSphere Portal can be plugged in as required.

Sonic and some of the other pure-play ESB vendors condemn IBM's approach, on the grounds that it is makeshift and may not deliver all the features and benefits of a true ESB. This argument rings hollow, though, as no two of them are agreed on exactly what those features and benefits are.

If an ESB is responsible for providing most of the core infrastructure needed to implement SOA, then surely it should share the most prominent attribute of SOA — namely, openness? Yet nothing is more antithetical to openness than a single, proprietary package that dictates how everything must be done.

A new twist — an open source ESB

The latest development in the ESB saga gives a new twist to the tale. Iona Technologies has announced plans to sponsor an open source ESB called Celtix which will be hosted by the not for profit ObjectWeb Consortium, based in France. Iona is positioning Celtix as a lightweight, entry-level ESB — no doubt hoping that a proportion of users will eventually upgrade to its own Artix.

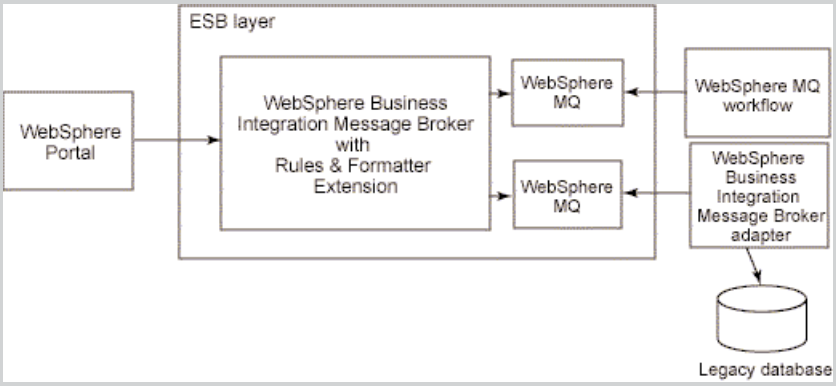
Two other open source ESBs, Mule and ServiceMix, already exist. The Mule project team has even expressed willingness to co-operate with the Celtix initiative. Meanwhile, Sun has also declared its intention of creating an open source ESB, which will implement the new Java Business Integration (JBI) standard.

Must an ESB depend on Java?

Another interesting question to ask is whether an ESB must be based on Java. Furthermore, to what extent can application servers fill the role of an ESB?

Some experts pour scorn on this latter notion, claiming that application servers are too big, ponderous and expensive. On the other hand, if JMS is a necessary ingredient of pre-

Figure 3.2: Outline of IBM's ESB offering (Source: IBM)



sent-day ESBs — which seems to be one dimension that so far is generally agreed — then computers participating in an ESB must support a Java infrastructure that complements JMS. Looked at in that light, J2EE does seem a coherent starting point.

JMS and ESB

The ESB concept represents a compromise between the almost unlimited freedom of SOA and the practical needs of IT users. It is all very well to talk up the theoretical advantages of SOA, such as freedom from vendor lock-in, less dependence on monolithic legacy applications, and the flexibility needed to support new business directions quickly and inexpensively. But it is not so easy to implement SOA without making certain concessions.

Thus, if an SOA is to be absolutely open, it must rely on a standard type of middleware which can connect to any applications regardless of platform, programming language, network, etc. At present, that means Web Services (usually based on SOAP and WSDL). It then becomes possible to add services written in C++, Java, Python, Visual Basic or other languages and running in the .NET, J2EE, Linux or other environments.

Unfortunately, using Web Services also entails some sacrifices. Typically, these include performance, scalability, security, manageability and the many features lumped together as Quality of Service (QoS). The logic is simple: Web Services go everywhere, and therefore provide only a limited common subset of the features offered by specialized middleware. They are like a walker, who can move about with great freedom — but without the speed, comfort and safety of cars, buses, trains or aircraft.

ESB vendors typically support several network protocols, including HTTP, SMTP, FTP, and JMS. Among those four, it is easy to spot the 'odd one out'.

Generally speaking, JMS offers higher performance, better scalability and more security, reliability, and manageability. In short, it registers high on the QoS scale. The other protocols were designed for specific purposes, none of which are very demanding in terms of QoS delivery.

As such JMS looks like a natural choice for an ESB's messaging backbone. Unfortunately, it has another characteristic that militates against the fundamental nature of an SOA.

JMS is an API for accessing enterprise messaging systems — in other words, it is a wrapper for MOM products. The underlying MOM product that does the actual work of exchanging messages is known as a 'JMS provider' (you cannot have a JMS product, only a JMS provider product that delivers the JMS API).

Unlike SOAP, and the many other specifications that have been layered on top of it, JMS says nothing about how messages get from one end of a link to the other. This means that, if the application at one end of a connection uses WebSphere MQ as its provider, the application at the far end must also use WebSphere MQ. JMS ensures that different Java application servers can communicate through a standard API, but it does not allow different vendors' products to interoperate.

For this and other reasons, Sun notes that "...the JMS API is intended for use within an enterprise, while XML messaging between independent enterprises over the Internet is being supported by the ebXML initiative and the new Java APIs... that support various ebXML specifications". It follows then that:

- **if an ESB is implemented with JMS, it is only suitable for use within a single enterprise**
- **if that ESB is then used as the backbone of a SOA, the SOA too will be restricted to the boundaries of a single enterprise**
- **or, if that ESB spills across enterprises, it will need a complex (and costly) apparatus of gateways and translation mechanisms to mediate between their different JMS providers.**

In the last case, all participants must agree to lock themselves into a single vendor's JMS provider or ESB. Once established, such a lock-in could probably never be broken, as all the organizations involved would have to make any change together.

The trade-off between SOAP over HTTP and JMS lies at the heart of SOA:

- **HTTP can go anywhere, and talk to anything; but it is relatively slow, insecure, and lacking in QoS**
- **JMS, on the other hand, encapsulates powerful MOM products like WebSphere MQ, Sonic MQ and Fiorano MQ; but it does not allow interoperation between them (except through gateways) and, with JMS, the same product must be deployed at both ends of any given link.**

There are some contradictions inherent in the whole idea of basing enterprise integration on Web Services. SOAP is

cheap and ubiquitous if layered on top of HTTP, the Web's application protocol. But then it lacks reliability, security, transactions and QoS in general. If, on the other hand, SOAP is layered on JMS, it:

- **acquires these desirable qualities**
- **loses the ability to go anywhere (especially through firewalls) and to interoperate with any type of application.**

Likely future developments

In his blog, Richard Turner (a program manager in Microsoft's distributed systems group) criticizes the ESB concept in these words:

"To my mind, an ESB is smart-plumbing to which to attach dumb nodes. To my mind, the ESB approach is less about providing customers with value and more about extending attempts to lock customers into a given platform. How, for example, would I add mainframe nodes, COM+ nodes, Web Service nodes, etc. running on different platforms and technologies on top of JBI/Sonic/[enter your ESB of choice here] without the use of adaptors and translators? And more importantly, if I chose to change vendors, how would I replace my ESB with that from someone else? Isn't it just crazy to even begin to countenance such an approach?" Even though Microsoft has some commercial interest in the matter, this is a logical analysis that highlights some of the weakest points in the Gartner/Sonic case for ESBs.

Responding to Turner's remarks, Anne Thomas Manes (research director of the Burton Group) goes further still. "I expect the pure-play ESB market to go away within the next 3-5 years. Today there are about a dozen vendors promoting their wares as ESB. I just don't see the market supporting this many vendors for very much longer". Manes also predicts that the ESB market segment will merge with those for Web Service management and Web Service registries.

Another possibility is that, as open source ESBs become mature, more and more SOA architects will turn to them, in preference to expensive proprietary solutions. Thus open source ESBs like Celtix, Mule and Sun's forthcoming package may bridge the gap between today's ESBs and a future generation based exclusively on the WS-* specifications.

Management conclusion

At present, ESBs look more like a wish-list than the specification of an actual software product. If there is a product that embodies it, that product is probably Sonic's ESB. It is certainly quite an achievement to snatch perceived leadership of a whole market segment away from no less a player than IBM. But that is what Sonic — with a little help from Gartner — seems to have achieved, moving the spotlight onto ESB (and its own implementation of it) and leaving MOM (and its best-known exemplar, WebSphere MQ) in lengthening shadows of neglect.

The exact definition of ESB depends on whom you ask:

- ***to a JMS vendor like Fiorano or Sonic, it is JMS with a set of optional add-ons***
- ***to a Web services specialist like Cape Clear, it revolves around XML and Web Services***
- ***for a full-spectrum IT supplier like IBM, with its multifarious products and its massive Global Services division, ESB is a pattern — a complex collection of products and services, whose precise specification depends on each individual customer's requirements.***

Even if ESBs major in standards compliance, they themselves are anything but standardized. Thus ESBs can be summed up as an attempt to preserve the advantages of SOA, while stiffening its QoS backbone. That involves upgrading the middleware infrastructure from Web Services to something faster, more secure and with better integrity.

The obvious candidate is enterprise messaging middleware such as IBM's WebSphere MQ and its competitors. But these are unashamedly proprietary, and do not even attempt to conform to any industry standard. So the best solution does seem to be to settle for JMS, which throws a veil of standardized respectability over whichever proprietary product lies beneath.

Most vendors and analysts depict ESB as an indispensable core that permits the construction of viable SOAs — rather like the steel skeleton of a building. The question is whether an SOA that is absolutely dependent on a proprietary suite of products can be said to retain the most vital characteristic of an SOA — openness. True, it may be possible to connect all sorts of services and applications to it. But the SOA itself is locked in to a single vendor.

Surely, as soon as the necessary Web Service standards are thrashed out and debugged, organizations will hasten to base their SOAs on those open, vendor-neutral standards. Only that way can the promise of open, vendor-neutral distributed computing be realized.

Enterprise Service Bus — optimizing the relationship between legacy applications and Web Services

Mark Lillycrop
Principal Analyst
Arcati

Management introduction

Application integration middleware has evolved to meet an increasingly complex set of cost, performance and data integrity criteria within the large organization. The Enterprise Service Bus (ESB) takes the integration issue to a new level of sophistication, creating in the process some interesting challenges for today's IT users.

In this analysis, Mark Lillycrop examines the linkages between legacy applications and Web Services.

Complex goals, complex tasks

There is nothing like the issue of application integration to remind us of just how complex the IT business really is. Commercial business systems have existed for little more than 40 years. But, in that short time, we have generated wave upon wave of — more often than not incompatible — technologies.

Despite an awareness of this, large enterprises remain typically built around islands of business function and information (silos, as they have become known). These are not only technically dissimilar but cannot even share the data and business logic they contain — unless a considerable effort to 'integrate' is made.

It is not difficult to see how this complexity has occurred. Constant pressure to deliver improved business benefit has encouraged companies to explore the latest development tools and techniques. New hardware platforms have brought with them an ever broader selection of operating systems, databases, and software packages. On top of this, the continuous process of company mergers and acquisitions has obliged IT departments to cope with an increasingly heterogeneous infrastructure.

What has become clear in recent years is that integrating dissimilar technologies is invariably more cost-efficient than the 'rip and replace' approach. Large companies have invested hundreds or thousands of man-years in developing core business applications and tuning them to achieve optimal performance in mission critical environments. Mainframe applications in particular (which in many cases were developed in-house and still offer unparalleled performance and data management characteristics) need to interact effectively with newer Web-facing applications without losing any of their inherent value to the business.

In today's rapidly changing extended enterprise, it is no longer sufficient to provide fixed points of integration with high-overhead translation methods. There is immense pressure on commercial organizations to:

- **be more responsive to customer needs**
- **combine disparate sources of information in order to create new business opportunities or to identify trends and changes.**

This means that today's integration tools must be flexible enough to:

- **handle both data and process integration 'on the fly'**
- **provide this around the clock**
- **exploit lightweight or heavyweight solutions depending on the characteristics of the application or task in hand**
- **ensure that all data integration must be conducted in a totally secure and manageable way (not least because security and regulatory compliance rank high on the corporate agenda).**

Before considering products and concepts that are evolving to cope with the new requirement for 'secure flexibility', let us review how application integration has evolved.

Point to point integration

Application integration really became an issue in the early 1990s, when large companies first recognized the need to share resources between core centralized systems and the newer breed of client/server technologies. Early point to point integration solutions were often designed in-house and were intended to cope with a relatively limited number of data format translations and network protocol conversions.

What characterized these early integration projects was that applications were tightly coupled. Once joined together, they were not easily put asunder. Communication between the two applications involved was generally synchronous in nature and the interfaces at either end needed specific knowledge about the data routing and translation processes involved. Any changes in this process required considerable programming effort and technical expertise.

Fixed point to point integration was quite sufficient for simple interoperability — internal applications exchanging data at predictable levels of performance and needing to generate minimal reporting information. Indeed, this approach to integration was often extremely reliable and secure. But it was notoriously difficult to change.

As corporate architectures and data exchange requirements became more complex, users developed equally sophisticated requirements and began to look for more generic, platform-independent solutions for their integration needs. Support also became an issue: what started as a way of integrating legacy data with new logic became a legacy platform in its own right. In addition, the overhead involved in maintaining fixed integration technologies discouraged all but the most essential modifications.

EAI and messaging middleware

The solution to the problem came in two forms. With the

arrival of products such as IBM's MQSeries, TIBCO's (then Teknekron's) TIB and other message oriented middleware in the early 1990s, users had at their disposal a backbone for application integration — a simple bus structure that offered asynchronous communication. This meant that applications could be loosely coupled: each application needed little or no knowledge about the data formats or protocols supported by the other.

It also meant that the delivery of dispatched messages could be guaranteed by the backbone — once and once only. Now larger numbers of applications could interact and exchange data — a process that often involved complex sequences of actions — and the messaging infrastructure would manage the data flow as well as ensure the integrity of the corporate information crossing the network.

A parallel development throughout the mid- to late-1990s was the emergence of new Enterprise Application Integration (EAI) technologies that reduced the overall integration effort. Companies such as BEA, IBM, Microsoft, NEON and Constellar developed a whole range of message brokers, hubs, adaptors and connectors which converted data, interpreted calls, and performed routing and other functions. Application logic and data were 'wrappered' to a greater or lesser degree to separate them from the integration process and maintain their integrity.

The important thing about these products was that they were non-intrusive. They did not affect the original application code itself. They provided a more generic mechanism for exchanging data and events between networked systems.

In terms of flexibility and support, these EAI and early messaging products made life much easier for the corporate integration specialist. They externalized integration logic away from the applications being connected. They enabled changes to be made relatively easily.

But, although many EAI functions were absorbed into the application servers that came along afterwards, they did not offer the real automated 'on the fly' capabilities demanded by today's on-demand service-oriented e-business environment. Moreover, in many cases they actually added to the complexity of the infrastructure: companies often found that different EAI tools were deployed in different parts of an organization which could not then be successfully integrated with one another.

In effect, it can be argued, the integration tools could not themselves integrate with each other. This was farcical.

Application integration in today's Web-centric environment

With the development of Internet-facing applications, many of the rules and objectives of enterprise integration changed again. One of the great positive developments of recent years has been the evolution of Service Oriented Architectures (SOAs) and Web Services, which are built around technologies such as .NET, SOAP, XML and J2EE.

Although still in their infancy, Web Services look as if they will overcome many of the fundamental problems that were encountered with EAI. For example, they offer to replace heavyweight interfaces with simple services that exchange information in a more transient, unstructured manner than with traditional applications.

In some ways SOAs are a natural progression of more mature service oriented technologies — such as CORBA and DCOM. But they may well be a technology, or group of technologies, whose time has come and which will ultimately relieve the enterprise of many of the integration headaches that it faces today.

Not all Internet-led changes have been beneficial for the large enterprise, however. One of the main problems has been that many of the tools and technologies underpinning the brave new world of Web Services are considerably less mature and functionally capable than the enterprise products they are intended to replace.

Furthermore, Web applications often proliferate in the furthest corners of the extended enterprise, well away from the disciplined data center environment. The management tools used to control and integrate them are far from ideal:

- **Simple Network Management Protocol, for example — the TCP/IP-based toolset for managing Web-based traffic — has taken many years even to approach the level of sophistication of enterprise management tools such as NetView**
- **similarly FTP (File Transfer Protocol), which was designed for nothing more than simple file transfer in the non-commercial world, is increasingly being used as an integration tool between secure, business-critical applications**
- **XML, which has revolutionized the way that data is deployed within Web Services, is very resource-intensive and can seriously affect enterprise network performance.**

IT departments attempting to bridge the gulf between mature, high-performance enterprise applications and the

new generation of flexible — but relatively unmanageable — Web Services need a consistent approach to integration. This must be one which will force them to focus on the relative performance, security, and integrity characteristics of each business process and transaction under their control.

To be clear, this is not just a performance issue or even a service management consideration. In many cases the main focus of application integration is regulatory compliance. With legislation such as Sarbanes-Oxley, Basel II, HIPAA, ISRS and the COBIT Act imposing a high level of data and application transparency on businesses across the world, technologies such as FTP — which contain little or no function for ensuring the integrity of data — need tight control.

These regulations apply particularly to organizations that exchange data with partners and customers, and where different policies may apply as information moves outside the corporate boundary. But even within the enterprise itself, there are now formal requirements to demonstrate and document the integrity of data exchanged between applications.

The Enterprise Service Bus (ESB)

This complex mix of technical challenges and governance requirements has led to the development of the Enterprise Service Bus (ESB).

One way to regard an ESB is to see it as an extension of the messaging backbone, but in a virtualized form. As such an ESB is not necessarily or specifically a product. It is a set of services and specifications that enables data and messages to pass across an internal or external network while retaining the performance, security and functional characteristics of the applications at either end.

One of the main differences between the ESB and the earlier implementations of messaging backbones is the breadth of application programming styles that an ESB supports. The ESB typically has many virtual entrances and exits in order to carry and manage a wide variety of traffic types. It allows data and messages to be both 'pushed' and 'pulled' between applications — supporting mechanisms such as publish and subscribe as well as Web Services.

At the same time, it should offer a range of add-on services like:

- **data transformation**
- **content- and subject-based routing**
- **database joins**
- **load balancing**
- **persistence**
- **reliability services.**

These can be automated to permit the ESB to determine the right level of business criticality and network resource to apply to any given process or transaction.

This level of flexibility — in the services and resources applied to each instance of application integration — is essential in today's IT environment. Nimble, lightweight low-cost Web Services need to be routinely combined with business-critical, highly available processes and transactions, built on other standards or even fully customized and proprietary. With multi-hop considerations and bandwidth constraints across the Internet, it is essential to keep traffic overheads as low as possible. Conversely, with business critical data on the internal network, bandwidth is far more manageable and the 'value' of the data in transit demands that a more heavyweight delivery mechanism be used.

The need to support these two extremes and everything in between, while giving the business the flexibility needed to change service management priorities quickly and easily, is at the heart of any decent ESB philosophy. In addition there are now a number of ESB implementations on the market, at various stages of maturity. IBM and BEA, for example, have relatively mature specifications available, built around their WebSphere and WebLogic application server platforms respectively (Figure 4.1).

There are also a growing number of offerings from integration specialists that apply ESB principles to specific market sectors and technical niches. The more sophisticated ESBs can contain features such as distributed query engines as well as so-called 'service orchestration' engines for sequencing the execution of services and supporting long-running processes.

ESBs pose new challenges

Although ESBs incorporate many of the features that are familiar to enterprise application integration specialists, they do pose their own new challenges. For instance, presentation services are needed that enable users to build portals which draw services from multiple sources. Similarly, development tools are required to allow programmers to take full advantage of the various quality of service functions embedded in the ESB concept. At the same time, system management tooling needs to evolve to provide a consistent management interface to the broad range of message and process types handled by the Bus.

All these aspects of the ESB are currently being addressed

by vendors but this is too often from too much of a technical perspective. In consequence it will be some time before the true value of ESB solutions is realized in mature data centers.

Making full use of an ESB

Having the technology in place to support a broad range of connectivity standards, protocols and styles is only half the battle. For most companies that support heterogeneous applications, the idea of an Enterprise Service Bus is attractive.

Arriving at a point where an ESB can be used to its full potential, however, is another issue entirely. For many, it might be unclear how technology of this kind can be used specifically to address an organization's strategic business objectives, or to assist an enterprise to move forward to adopt a Service Oriented Architecture while continuing to maintain and enhance the existing base of legacy technologies.

For users, SOAs often require a period of considerable self-analysis — at the business level as well as at the IT level — when the organization has to examine its current portfolio and the requirements of the business, and sets out a roadmap that will allow the IT function to evolve into a service-driven infrastructure which is closely aligned with the business. Full-function ESBs can involve a substantial amount of re-engineering. Organizations need to be completely confident that their existing requirements will be adequately met by the new integration backbone.

As might be expected, the larger ESB vendors are selling their products more as services than as free-standing technology. In many cases, they are encouraging customers to adopt an SOA and ESBs in an incremental fashion:

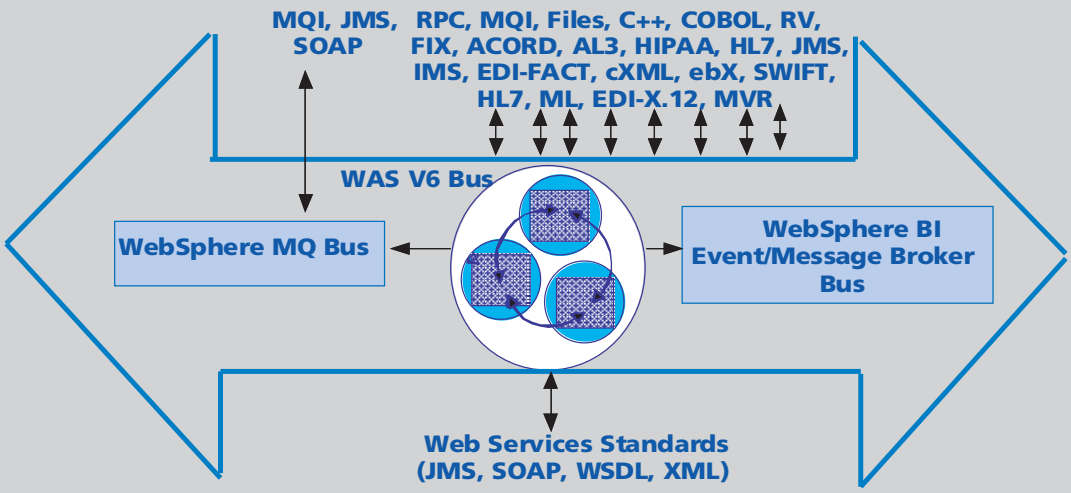
- **deploying relatively simple message management services across a backbone initially**
- **then taking advantage of the more advanced QoS features as the technology becomes established.**

As such ESBs are rapidly becoming an area of premium-level expertise, since implementation requires both:

- **a detailed understanding of the relative cost/performance needs of the different types of traffic crossing the network**
- **a full appreciation of the way that this new capability can be used to meet new and existing business objectives more effectively.**

As with many major changes in IT infrastructure, user organizations will turn to service providers to ease them over

Figure 4.1: IBM view of ESBs



the initial stages of transition. They will gradually form their own teams of integration professionals with a specific understanding of their individual business strategy.

Management conclusion

Today's heterogeneous IT environment is a very different beast from the secure internal data center of a decade ago. Integration is as much about demonstrating data security and integrity (to satisfy regulators and stakeholders) and about responding rapidly to changing business requirements, as it is about handling multiple data structures and routing methods.

SOAs and the Enterprise Service Bus are a direct response to the integration challenges facing the enterprise. While the ESB concept is arguably still in the early stages of development, some of the products now available offer a very rich implementation of the concept as it stands today. Enterprises which need to improve their level of IT responsiveness by embracing Web-based technologies (built around XML, SOAP, J2EE and .NET) while maintaining and enhancing the value of their existing business applications should consider the ESB route seriously.

Information flow in middleware infrastructures

Dr Keith Jones
IBM Software Solutions Worldwide

Management introduction

We live in the 'Age of Information' — and information is an important currency that drives modern business development and performance. We also recognize that knowledge accumulated within an enterprise is one of the most valuable assets when used creatively to solve business problems. We are increasingly knowledge workers now, and highly dependent upon information flows.

Yet the IT systems we construct to support enterprises, both large and small, are built from components that have been designed for speed and accuracy in the handling of data — not information or, more ambitiously, knowledge. What can middleware designers do to raise the level of processing above that of data to provide enterprise systems that understand and process information?

In this analysis, Keith Jones reflects upon current middleware support for data processing and asks what can be learned from disciplines outside the IT industry that might be applied to middleware architectures so that real information in corporate data flows is recognized dynamically, extracted and processed for greater value. The answer seems promising and applicable in the relatively near future.

What is information?

We are constantly reminded by the media that we now live in the age of information. This comes from an historical analysis of civilized society and its development through eras that were, at one time, preoccupied with agriculture, then manufacturing and now services. During those earlier eras the main currencies, and therefore sources of power, were labor and then energy. Today it is information.

Sir Francis Bacon is often quoted as the first to coin the phrase "for knowledge is itself power" some 400 years ago. Much has changed since then in the commercial world but Bacon was right — many of us are now considered to be knowledge workers who are empowered by the flow of information. In fact the essential differences between data, information and knowledge are coming to the fore as our focus is turned toward middleware support for communication flows in commercial systems.

What is information and how does it flow in modern middleware?

The study of signs, linguistics, philosophy and computer science are converging on a greater understanding of information and our creative use of it as a valuable resource. The philosopher, Charles Morris, was the first to identify the dimensions of language usage as:

- **syntactic (related to symbols)**
- **semantic (related to meaning)**
- **pragmatic (related to intention).**

Language and information are very closely related in the modern world.

Data flow

In 1948 Claude Shannon published his landmark Mathematical Theory of Communication. He observed that, in communication systems, information is only present when signals received in a stream are unpredictable and that many communication streams are highly redundant (Figure 5.1). That English language text is so redundant is often illustrated by the fragment "only infrmatn esentil to understandn mst b tranmitd". The information theory that Shannon founded underpins the enormous capacity, now often measured in giga-bits per second, of modern commercial communication links.

The truly incredible volume of bits that flow between nodes in modern enterprise networks is a fact of life for middleware infrastructure engineers. The data that is contained within those bit-streams is processed by many thousands of middleware components. But how much information is flowing at the middleware layer and how much knowledge is accumulating in a form accessible for creative re-use? These are the interesting questions now being posed.

The data flow within enterprise systems takes many forms — from networking links to operating system memory to middleware application servers to business logic and file and database servers to display and other forms of user interface. It is no coincidence that in the early days of middleware systems this flow of enterprise data was called 'data processing' and not 'information processing'. The hardware and software components involved were aware of bits and bytes but not the information content that was flowing.

Even today the hardware, firmware and operating system software provide the basic handling of bits and bytes with relatively little recognition of data types and structure (I deliberately exclude compilers in this discussion). As the 'push down' principle applies over time (Figure 5.2) these components are becoming functionally richer and some amount of data content is now recognized at even the lowest layer in infrastructure platforms.

At the middleware layer a growing number of components provide data recognition capabilities. Network listeners, security policy mechanisms, Java Virtual Machines, messaging channels, XML parsers, relational database access components — and many more from a variety of vendors — all recognize a wide variety of standard data types and structures for their internal use and support for business logic.

At this layer, middleware facilitates the rapid and efficient flow of data. However, the typed, un-typed, structured and un-structured data is only a representation (or mixture of representations) of the information that is flowing. Encoding and decoding of information to and from appropriate representations is presently a function of the business logic layer. Transformation of data from one representation into another may be provided by the middleware layer as required by the endpoints for a data flow (Figure 5.3).

Whereas human observers might be able to determine information content in a data flow from knowledge previously acquired about data types (for example, voice, video or text) and from the context of a flow, it is presently unlikely that a middleware infrastructure will do so. This is because information flow requires not only data (representation) but also flow context and appropriate recognition of types within the flow. If the token 'PA0574638' flows through middleware from business logic to a user interface there is usually no recognition by middleware components

that this is customer number information rather than bank account number or airline ticket identifier information.

Information flow

Some middleware components — for example Java Virtual Machines, application servers and relational databases — provide capabilities that go beyond recognition of simple data types such as Unicode characters and signed binary numbers in data flows. Definition and recognition of complex 'application' data types by these components allows for more compact business logic and for automatic encoding and decoding into convenient representations — for manipulation in memory, transport or storage.

Even with these components in place the enterprise system does not yet recognize the flow of customer number information without the presence of appropriate business logic. It is this logic that not only recognizes customer numbers but also acts upon the relationships and properties of such information. For example, the relationship between customer number and customer address may be coded within business logic in terms of the set of operations that can be applied — add address, change address, remove address, etc.

The business logic layer in a typical enterprise system might also encode and enforce rules that guarantee the integrity of critical business information. Such rules might ensure for example that customer information is:

- **whole (all customers are associated with at least one address)**
- **secure (all customer information is password protected).**

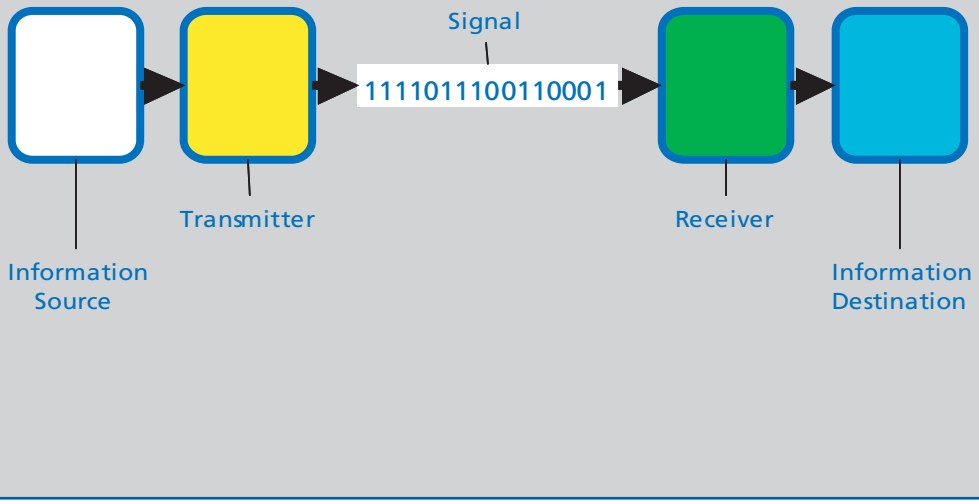
In this kind of scenario middleware business rules engines can provide support for configuring business logic with easily modified rules. There is no sense in which the middleware knows how to apply these rules 'automatically' without the business logic being present.

The context that distinguishes an information flow from a data flow can also be supported by middleware. In Java application servers, for example, context data can be carried by the middleware to qualify the transport of messages between components. WebSphere J2EE message handlers can create context that contains:

- **identity tokens**
- **security policy statements**
- **quality of service parameters**
- **a variety of other data as required to transport messages between communicating endpoints.**

This kind of context data may serve to distinguish data flow

Figure 5.1: Communication system



contents and cause middleware to take specific action — such as enforce enterprise policy or choose between available data transport protocols for reliability or availability. However, even when context data is available with message content and when business rules are present at the message processing endpoint (Figure 5.4), there is yet no sense in which middleware can readily re-use the types, relationships and properties of the information inherently represented by a data flow. This is because there is no coherent organization of meta-data to support that re-use.

Information flow within middleware would require this metadata, together with facilities for type recognition, context switching and representational transformation. Without 'push-down' of the semantic capabilities presently built into the business logic, middleware cannot automatically recognize customer numbers and handle them according to their meaning within the context of a business process.

Knowledgeable middleware

If you accept that middleware might at some time — and this could even be in the reasonably near future — be capable of recognizing and correctly handling information in enterprise data flows, the Holy Grail will be when middleware can somehow become 'knowledgeable', albeit in a restricted sense.

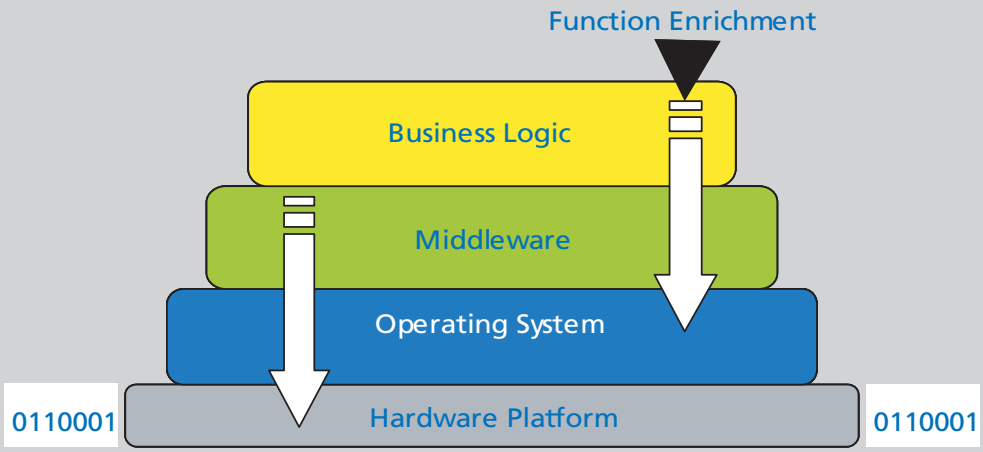
Linguists and philosophers tell us that to be knowledgeable is to be able to possess information and possess the capacity to use that information in creative and constructive ways. If middleware components are able to recognize elements of information and store them in accessible forms, the first step toward building truly intelligent enterprise infrastructure will have been taken.

The current generation of middleware supports collaborative application software — for example Lotus Sametime — that enables human users to exchange information and exercise their knowledge to solve business problems. Instant messaging, whiteboards, voice, data and video all flow through the middleware infrastructure. However, these flows are data flows and middleware is not aware of what information is actually present.

Middleware that is capable of recognizing information flow can become knowledgeable at the lowest level of 'expert' proficiency by adding algorithmic rules to determine processing of that information and take appropriate action. Autonomic middleware infrastructures are now emerging with this level of capability; for example look at WebSphere. By recognizing information flows from internal sensors and applying rules, this middleware is able to take action to correct faults or optimize processing in some way.

To become more expert, as humans can, middleware

Figure 5.2: The 'push-down' principle



components must be given the capability to:

- **recognize information flows**
- **apply rules when normal circumstances are encountered in context**
- **dynamically construct alternative strategies when needed.**

This quantum step may prove to be more difficult to achieve as it will involve reasoning about information received and understanding the bounds of possibility within context when it comes to taking action. At this level middleware could be said to 'know what to do' when circumstances demand action.

At the extreme, on a scale of expert proficiency this corresponds with the capabilities of a human expert, middleware would be able to make decisions and take action based on available information flows without recourse to rules or algorithms tuned to specific circumstances. This would require large amounts of stored knowledge about the domain of expertise and considerable processing to determine most appropriate actions.

Very complex business logic in some enterprise systems exhibits some of the capabilities of an expert system. But no commercial application system has yet achieved the extreme that might one day be possible. The results of significant research in this field and further technology push-down will be required before middleware can take on the role of the expert at this level.

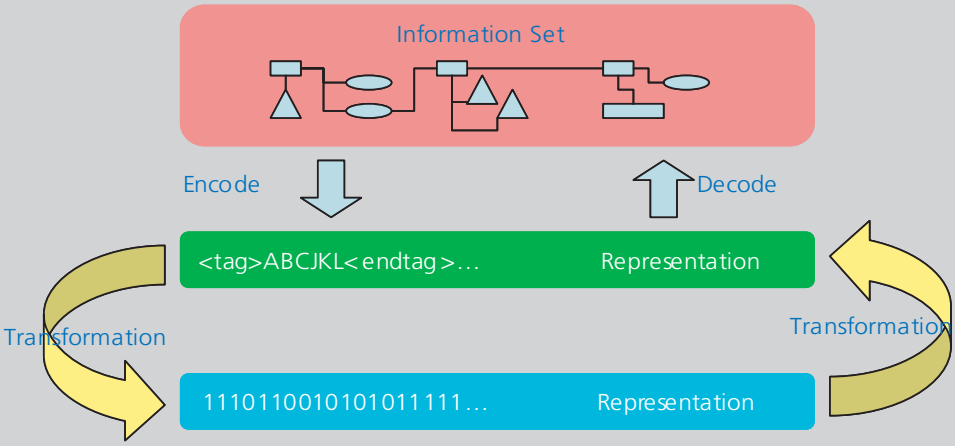
Service information flows

It is surely more than just a happy coincidence that as society becomes more service-oriented, and information intensive, so too are the enterprise systems that support our commerce. Services within enterprise IT systems are discrete units of processing tuned to business process goals that handle data flows at their interfaces. Middleware components are responsible for transporting and routing those data flows according to service contracts in service-oriented infrastructures (Figure 5.5).

Each service description declares the types and structure of the data flowing between service endpoints subject to service invocation policies. This gives middleware infrastructure an opportunity to understand the information content within those data flows. By adding business process context and semantics (meaning) to the data flowing, middleware components could become aware of service information flows.

For example, an ESB infrastructure could not only route and transform service requests and responses at the data level but also intelligently handle specific information types by

Figure 5.3: Information representation



providing semantic enrichment and transformations at the information level.

The effect of middleware taking on the responsibility for recognizing and automatically processing information within service data flows would be further to reduce the burden application programmers for producing business logic. It would also provide an opportunity for enterprise-wide consistency in processing of certain critical types of information.

Middleware which recognizes information by type and provides support for action inferred from that type or combination of types in service flows (Figure 5.5) must be configured with semantic metadata that is specific to particular business processes, cross-enterprise usage, particular industry or cross-industry usage. Since information flows between service endpoints must share semantic metadata in order to communicate effectively, the role of middleware that is aware of such meta-data could be to dramatically increase the value of reuse at both the semantic and service levels.

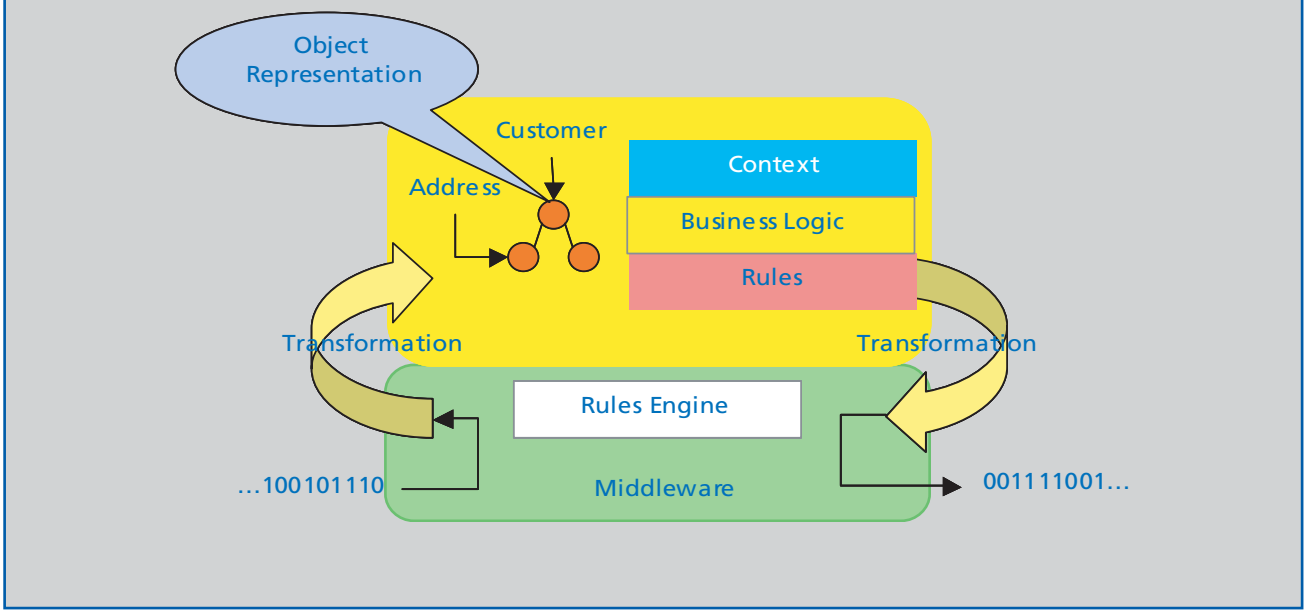
Management conclusion

Information is one of the most valuable resources in the commercial world and yet there is limited scientific understanding answer for what constitutes good information and how to process it. Most are agreed that, whatever it is, information powers the age we live in and that we are rapidly becoming a knowledge-driven society.

The IT systems that provide the engines for commercial enterprises in that society are extremely fast and accurate as they process huge volumes of data flowing through their middleware infrastructures. Yet there is an important difference between processing data, however quickly, and processing information. Linguists and philosophers equate data to tokens or symbols that represent real-world objects. Information on the other hand is representation (data) with associated meaning. Furthermore that meaning is often highly dependent upon the context in which information is being presented.

Middleware is already acquiring many of the capabilities needed for enterprise infrastructures to start recognizing meaningful types, their relationships and properties in data that flows from component to component. At the same time it is recognized that business logic presently contains much of the context needed for information to be extracted from data as it is presented by middleware. If that context and the metadata for meaningful types (relationships and properties) could be isolated and made available to middleware components it would be possible for those components to become aware of the information content in enterprise data flows.

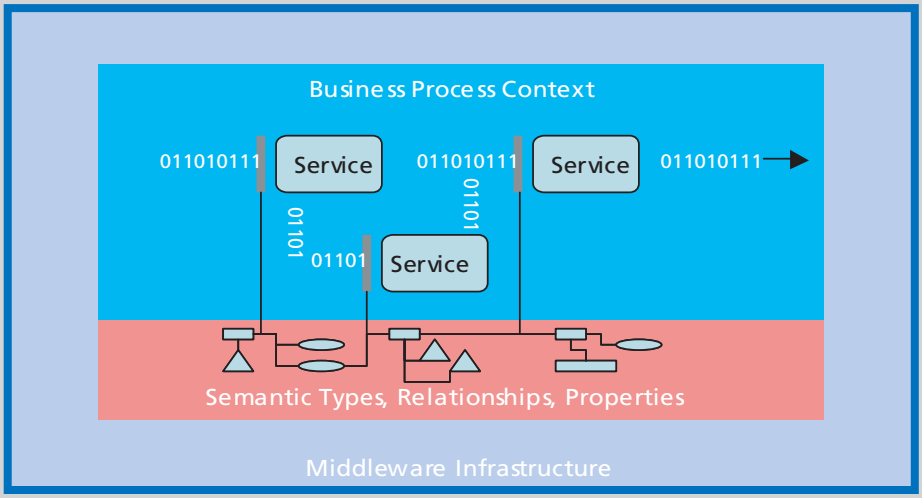
Figure 5.4: Information flow in middleware infrastructure



The current focus on service-oriented middleware is timely as it introduces service interfaces into enterprise data flows at strategic points that are closely aligned with business process boundaries. These interfaces might be the optimal place to insert contextual and semantic middleware function to raise the level of processing and the value of middleware infrastructures.

A longer term goal for middleware is that it should not only recognize and process information but also become more knowledgeable. Some 'expert' behavior can already be seen in some emerging autonomic middleware infrastructures where the information flows are internal and rule-based behavior is automated. In future, behavior of this kind will undoubtedly become more widespread and more intelligent.

Figure 5.5: Service information flows



Thoughts on approaches for SOA/ESB applications: part I

Nick Denning
Chief Technology Officer
Strategic Thought

Management introduction

A Service Oriented Architecture (SOA) should not be a framework just for deploying re-usable objects. If the goal of an organization is to improve the efficiency of the development process then it should focus on improving its use of Integrated Development Environments (IDEs) — such as Visual Studio .NET or Rational Application Designer (RAD) and their associated repositories — so that developers can consult a dictionary of objects which can then be re-used or extended. SOAs have a greater purpose.

In this analysis, Nick Denning uses the experience of Strategic Thought to:

- ***describe the historical problems that organizations have encountered when developing applications***
- ***examine where and where not to use an SOA***
- ***reflect on three case studies that illustrate the challenges***
- ***consider whether everything is a Web Service***
- ***show how problems are likely to be encountered in an SOA architecture, and explain the importance of addressing potential problems in advance.***

As will become clear, Mr. Denning anticipates a need for high grade architecture, excellent staff and first class quality procedures — and these will significantly increase costs in the short term although, in the long term, major benefits will flow through increased business agility and reductions in overall costs. In a subsequent MIDDLEWARESPECTRA, he will outline the patterns and approaches — again using examples — that Strategic Thought believes must be followed to design and deliver applications that can be deployed into an SOA.

The case for SOA — moving away from stove pipe processing

We believe that there is a fundamental requirement for organizations to adopt an SOA. Currently applications are written as 'vertical' stovepipes focused on delivering business functions rather than on 'horizontal' cross business processes. EAI technologies have been used to facilitate cross business processes to provide electronic processing within the business and between business partners.

For this analysis we accept as given that these links are essential to deliver internal business efficiencies and to enable trade with partners and customers. Undertaking these projects has identified challenges. The EAI technology is relatively straightforward but the design challenges are significant. For it is difficult to:

- **expose existing application logic as services**
- **provide a 'tag' to existing applications that enable external processes to track work within the system executed by a process**
- **identify changes within a database which will cause output messages to be generated**
- **manage transformation data which will enable message content and format to be converted as messages flow between systems**
- **deliver an appropriate high level architecture and phased implementation that maintains the confidence of a customer by delivering value from each phase.**

An SOA approach should address these challenges and:

- **reduce infrastructure costs through the use of common technology**
- **reduce develop costs through the use of standard frameworks**
- **facilitate business change through a more flexible process definition capability**
- **enhance standards that improve connectivity between organizations.**

However to benefit from an SOA we need fundamentally to change our approach to the design and implementation of all existing and new applications. Current applications must be decomposed into components that can be invoked by a processing layer implemented within SOA technology. Then, new applications must be designed as discrete components.

It is necessary, therefore, to reverse today's most common design approaches. We need to think first about the architecture and the interfaces and to consider how we communicate with our partners and customers. This should be done before we think about the internal business logic in each component.

We need, therefore, to work towards a model where end users are able to choreograph new processes using a code free work flow approach that invokes existing services. This will enable every business to focus on delivering end to end processes across its enterprise.

An architecture (SOA) is essential to developing these processes, interfaces and components. Such an SOA requires:

- **state management of those business processes**
- **the invocation of services to deliver those business processes**
- **automated management of exceptions (where possible) to act if/when any of these processes stall or fail.**

To implement an SOA means that there is a need to architect a layered approach with each business process implemented in its own well defined layer. This means that these business processes may be extended by using these well defined and robust services which:

- **can be executed as standalone, standards-based services**
- **possess well-defined characteristics for input, output, compensation and exception handling.**

Organizations should, therefore, only consider implementing an SOA if they have already introduced the appropriate and necessary internal changes and adaptations, namely that each business must be:

- **process driven**
- **focused on delivering a measurable return on investment**
- **committed to regular incremental spending to support the progressive evolution of the infrastructure and business capability**
- **capable of, and possess, the necessary high grade expertise to architect, design and implement business solutions which deliver competitive advantage.**

An organization that is unable to integrate their processes electronically with those of their customers (and suppliers)

will not be able to trade and will be squeezed out. 'Do nothing' is not an option.

The rate at which this architecture must be implemented is going to be shaped by 'co-opetition' — found industry by industry — which may mean that it may be several years before all have been forced to adopt some aspects of such an infrastructure. However once major organizations demand full integration as a pre-requisite to doing business, others must follow quickly.

Organizations must, in consequence, be acutely sensitive to avoiding unnecessary trail blazing in any industry that is not yet ready or where there is not yet an accepted business case, yet make sufficient investment in systems and capability readiness so that they are able to move quickly when required.

There is, however, a parallel risk that many organizations will try to implement their own SOAs without:

- **having made the necessary investment in high grade staff to architect, design and deliver such an SOA solution**
- **researching the business case against which the investment can be justified.**

The result will be failures to achieve the desired business advantage.

The alternative

The alternative to internal development is to procure an SOA as part of an overall enterprise solution. If an organization realizes that it does not have the necessary capabilities or drivers, then it must consider buying in an infrastructure from a competent vendor (such as SAP or Oracle or others) and supplement this by building added value components that provide their own industry-specific capability — rather than trying to grow their own SOA from scratch by purchasing technology.

We believe that this will be the most sensible choice for organizations that currently support their business by purchasing Commercial Off The Shelf (COTS) packages. A benefit (or challenge) will arise depending whether or not organizations and their application providers in each industry are (or are not) able to select a common technology and interface definitions on which an industry standard SOA can be based.

Web Services go someway to facilitating interoperability. However there will not be sufficient room for an excessive number of infrastructures, each requiring its own separate skills to manage.

The full benefits of an SOA will only be achieved if an organization is able to adopt two separate technology platforms enabling it to minimize the cost of technology while retaining supplier competition. This will support both process and application integration via the SOA and the application infrastructure itself.

The dangers

Purchasing packaged applications, however, is not a silver bullet. We are aware of too many failed projects — and even failed companies — that resulted from purchasing a system and then seeking to tailor that system rather than supplement it. ERP procurements appear to offer particular challenges.

To sweat our assets we need to exploit the SOA to add value to the standard packaged solutions by:

- **understanding how to integrate existing applications into the new SOA to deliver quickly**
- **being able to identify when to write new business logic.**

A further real danger exists: most developers are not used to writing applications that handle high transaction rates and many legacy applications will not perform under load. Many applications have been built to respond to user input from a screen-based interface which inherently introduces intervals between transaction that avoid a range of problems such as lost updates or deadlocks. They do not scale and often suffer 'application lock up' under heavy load.

An SOA application almost necessarily is message based and can be expected to deliver work unevenly and in bursts of excessive traffic. The whole point of an SOA is to deliver an electronic infrastructure that will minimize or remove the human interaction to support Straight Through Processing (STP), and must be able to support higher transaction levels as business increases.

The danger is that if an existing single threaded application service is wrapped into the SOA via the Web Service, that capability will not scale and transactions can get stuck in the infrastructure. On the other hand if the component is implemented in a multi-threaded model it may suffer 'application lockup' if too many threads are running concurrently.

The purpose of the case studies below is to identify some

characteristic scenarios of poor application design which may impact the ability to wrap and introduce existing logic into an SOA.

The yet more serious issue is that developers, writing new services, must ensure that their designs address these issues.

Case study — the implications of polling

A sales trading application (Figure 6.1) for a large investment bank needed to refresh its screen whenever any changes were made to the underlying database. When a sales trader received an order from a customer and entered it into the screen, the system needed to alert the trader as quickly as possible so that the trader — possibly operating in another geography or time zone or market or combination — could:

- **see the order**
- **execute trades in the market at the best price to fill that trade**
- **record the executions on the database**
- **receive change notifications to the order from the client and incorporate those into the order or reject them.**

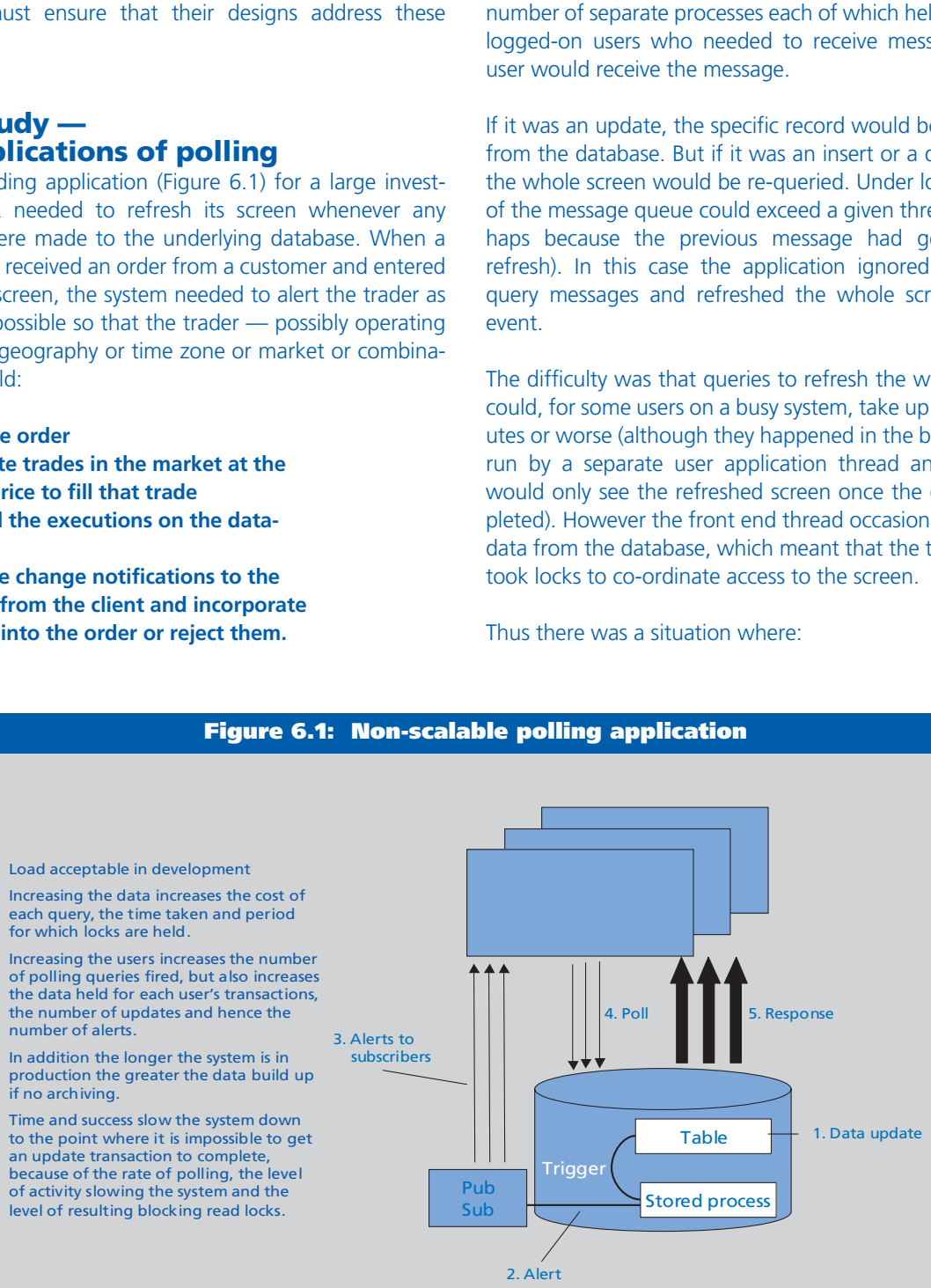
The existing application held details of the order. A Sybase extended procedure, written in C, was fired from a database trigger which generated a message using the MSG facility in the UNIX kernel. These messages were read by a separate application which then broadcast messages to a number of separate processes each of which held a table of logged-on users who needed to receive messages. Each user would receive the message.

If it was an update, the specific record would be recovered from the database. But if it was an insert or a delete, then the whole screen would be re-queried. Under load the size of the message queue could exceed a given threshold (perhaps because the previous message had generated a refresh). In this case the application ignored the single query messages and refreshed the whole screen in any event.

The difficulty was that queries to refresh the whole screen could, for some users on a busy system, take up to 20 minutes or worse (although they happened in the background, run by a separate user application thread and the user would only see the refreshed screen once the query completed). However the front end thread occasionally fetched data from the database, which meant that the two threads took locks to co-ordinate access to the screen.

Thus there was a situation where:

Figure 6.1: Non-scalable polling application



- **if the system was busy the users could take up to 30 minutes to log on in the morning before they saw any data; this was due to the highly complex dynamic SQL queries that the users were able to build up to select specific records from the database**
- **whenever a long query ran it was possible for updates to block, timeout or deadlock; thus the business of the system could not be executed**
- **in normal trading the system just about worked; however, if the system became busy with many new orders, this triggered full screen refreshes for the affected users (although most of the rest of the system could keep up)**
- **but, if it became busier still, the messages broadcast back to the screens (even the simple and efficient update messages) could exceed the size of the queue — causing a full query to be initiated which in turn caused all the other application queries to take longer; this meant that the number of other users, whose message queues exceeded the limit, increased further, thereby triggering additional full queries**
- **the system went into meltdown whenever the load exceeded a particular threshold.**

Over-complexity had produced a situation where an increasing transactional load could cause a sudden increase in system load which in turn could cause the application to lock up. A design focus on functionality had resulted in a failure to consider:

- **what happened under load**
- **what happened when the number of users increased**
- **the implications of locking and transactions.**

Performance testing should have identified these weaknesses but there was no framework to stress-test the system. Our (Strategic Thought) involvement started when we were called into to resolve the situation.

We were able to do this by implementing a 'publish and subscribe' model:

- **we removed and replaced the UNIX home grown publishing model (that had no concept of subscriptions, so all users received all messages) with a publish and subscribe model based on WMQ's publish and subscribe**
- **one saving grace of the existing system was that the records displayed on the screen were standard; this meant that whenever a record change occurred we could publish the whole image so that the user would not have to go to the database in order to recover the record again**
- **we introduced a copy of the current state of the screen in an XML document which sat behind the screen that maintained a screen image of data**
- **we examined the query and were able to identify markets that were of interest to the user; we then required the user to subscribe only to those messages relating to that market, providing at least a first order filter (and thereby reducing the message traffic)**
- **once a record was received by the front end application, we applied a different filter — based on the mechanism by which the original screen was built up; this enabled us to check whether a specific record was of interest to this specific user.**

The effect of this approach was:

- **once a user had started up and generated his or her initial database view there was no requirement to re-fresh the screen (though there was a manual function for providing confidence testing in the early days); at a stroke we removed completely all of the previous polling load**
- **the number of records published was dramatically reduced by providing a 'coarse' pub/sub filter — leaving the front end to perform an additional fine filter; the loads on 'normal' client PCs were reduced — where previously a dual Pentium with 512MB of RAM had been the minimum configuration, now a regular PC could be used**
- **if a database record changed, WMQ's pub/sub mechanism passed the message to the front end almost instantaneously, and substantially quicker than the previous mechanism; because the pub/sub engine broadcast all messages within a database transaction, and because the front end applications had a thread waiting on an MQGET, the messages were delivered at the same time (at least as far as most users could tell)**

- **when trades were applied to the system the huge reduction in system load — and the almost total absence of read locks from re-reads — meant that the transactions were processed almost instantly and were then propagated quickly**
- **if the system became busy there was only a linear impact on performance, rather than an exponential growth in traffic; a system that had previously been based on a 4 CPU Solaris machine with 4GB of memory and which ran flat out now operated as being in a mode that was practically idle.**

Case study — building a work flow engine

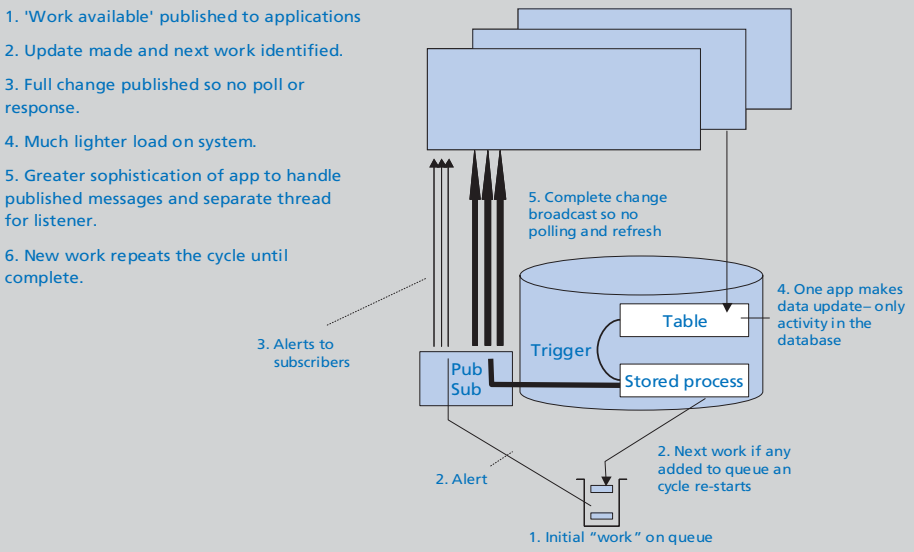
The improvements described above proved to be a great success. This success then meant that a new capability was required. It was necessary to build a FIX 4.2 link between fund managers and the sales traders (Figure 6.2) so that:

- **fund managers could transmit their orders to the sales traders electronically**
- **once the order was fulfilled, the sales trader could release details of the executions used to fill that order to the fund manager.**

Basically the requirement was to replicate the functions performed by hand on the screen by the sales trader to enter orders, and then to capture the details of the executions returned so that the FIX reply messages could be generated and transmitted. However the sales traders needed a significant degree of control over the system so that:

- **they could see all orders passed into the system, and decide whether or not they accepted the order from the client (subsequently they required a business rules engine to be able to allow orders from particular clients, under a definable order size, to be entered automatically onto the system)**
- **a dialog of messages based on the FIX 4.2 standard needed to be implemented — with some messages released automatically and some released optionally under user control (which is stateful because, for instance, it needs to know where in the overall sequence of messages the particular message under consideration is and the state of some of the other messages in generating a reply)**
- **specifically the sales trader needed to be able to control details of order fills released — so that the average price for the stock could be transmitted (which might hide a significant variation in individual trade prices).**

Figure 6.2: Scalable work flow solution



The solution implemented was based on the following mechanism:

- **inbound messages were received by an enrichment agent which read the message off the inbound queue and converted the FIX message to a raw XML structure which had a one-to-one match with the FIX tag delimited structure**
- **the second stage then performed transformation and enrichment by taking the raw data fields and creating the internal codes and structures that were understood by the system**
- **the third stage of processing created records in the database tables for the transactions**
- **a message was placed on a queue identifying the transaction, explicitly setting the MSGID to the key of the record in the database (which was, therefore, by definition unique)**
- **a message was published via the pub/sub engine.**

Users authorized to receive inbound trade requests from clients immediately saw the record on their screens:

- **to perform an operation the application created a command message which it transmitted to a server; the server received the message, recovered the underlying operational message from the queue identified in the command message and invoked the required database functions to perform the same operation that might be performed by the sales trader, and then enqueued a message to the pub/sub operation (to remove the operation from the command screens)**
- **if by any chance two users invoked an operation on the trade at the same time, only one user could dequeue the underlying operational message (the other user received an error message); thus there was no possibility of encountering the 'lost update' problem described previously**
- **when the executions had been filled, stored procedures fired on updates to the relevant tables; in turn these checked whether the orders being filled related to orders input via the electronic feed — which required FIX messages to be generated and transmitted back to the commissioning fund manager.**

A point to make here is that, regardless of who actually performed the operation to accept or reject a trade or to release executions back to the fund manager, the mechanism for refreshing the screen was identical for all users, including the user actually performing the operation. There was no synchronous update or special treatment for that user.

The essential elements of this solution, which was adopted, were:

- **the use of a pub/sub mechanism for operating identically for all users (and for maintaining the screen displays); there was no retrieval of data synchronously to render the screen, once the first query to build the screen had been completed**
- **the identification, within a queue, of all work requiring to be performed, so that only one user could lock the operation — by dequeueing the message on the queue, performing the business transaction and updating the database in a single transaction**
- **the maintenance of all data in the database so that the business transaction (to change the state) could take command, retrieve the data from the database, apply the business rules and then complete the operation within a single transaction.**

Case study — an overloaded application

For a quite different organization we worked on an application that was running on relatively underpowered equipment, although performance was generally good for most of the time. The data structures (on the system) modeled, however, the equipment located at the customer's site — comprising:

- **client details**
- **allocated controllers**
- **devices attached to the controllers(s)**
- **the elements maintained in the device.**

When any item of this equipment was changed, a reprocessing took place of all the equipment on the customer site. This involved a database transaction that updated all the information provider's records (Figure 6.3).

Most of the customer clients had one, or possibly two, controllers with up to 24 devices on each controller and up to 24 elements on the device. Updating the details of these

clients usually took about 5 seconds, due to the low power of the computer (which could support only about 80 users).

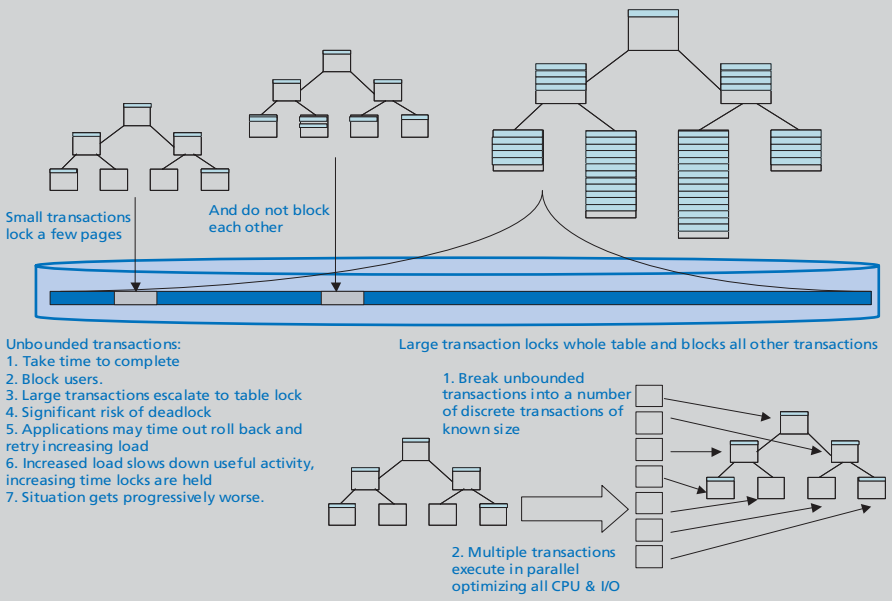
If, however, one of the clients that possessed a large number of controllers was updated (one client had 256 controllers, while others had as many as 40 or 50 controllers), then it could take 20 minutes in a test environment, with only one user running, to update the database. In production, with other users working in parallel, the system would take longer and could even grind to a halt for a whole afternoon — typically for the following reasons:

- **there was a timeout set — if any user query was blocked by other users locking pages in the database a lock timeout could occur; after a lock timeout, the affected transaction would perform a database rollback to undo any changes already in the system and would then re-try the transaction for up to 3 attempts**
- **this meant that if the system was already loaded, as soon as timeouts occurred the load on the system increased dramatically; this caused every other process to slow down and increased the risk of a transaction being blocked and then timing out — which then slowed the system down further resulting in more yet more users timing out and starting to re-try, and ...**
- **the system would grind to a halt until all sessions had timed out three times and returned control to the user; however the user usually performed a retry and started the processing off again, even though it then timed out yet again**
- **the database system used did not have record level locking (only page and table level locking); once the system started to become busy — with lots of separate users, each requesting their own page level locks — then the risk of blocking increased which in turn increased the chance of more threads blocking (due to the inherent coarse granularity of the page lock design).**

The coup de grace arrived if a user tried to modify a client with a large number of devices. First the actual processing time was a linear function of the number of elements being processed. Thus the time taken for a large client update increased.

However the database records were quite large, and if the layout of those records on disk meant that the number of page level locks exceeded the number of locks allowed on a table, the process would escalate the level of locking and request a table level lock. This meant that an exclusive lock was held on the table preventing all users getting any locks

Figure 6.3: Characteristics of an overloaded application



on any page in the table until this one session had completed the transaction.

This guaranteed that every other user would be unable to complete any work. In effect 80 users would be constantly trying, timing out, retrying, aborting and restarting. Thus a process that might take 20 minutes in test, could take over 3 hours in practice.

Unsurprisingly, this system was perceived to be unusable. Add to this problem the fact that there was a significant amount of cursor processing in the system. As a cursor was processed, data was being written onto the network to control elements of equipment — but that transaction might be abandoned and changes cancelled: but the changes were already in the network and were never corrected until a user finally ran a successful process for that specific client.

In short, though not a 'big application', this was a disaster. Something needed to change.

The adopted solution that we recommended to the client was to start by adopting some rules:

- **no database transaction was to be capable of being 'unbounded', and must always execute within a defined time**
- **preserving the integrity meant that we could not break a database transaction up into segments if there was any risk that not all the segments could complete**
- **a mechanism to prioritize work was introduced so that the most expensive resource — the user — was never kept waiting by a screen which was 'processing...'.**

The approach implemented was to:

- **take all the previous database transactions and identify them as a business transaction — writing a single record into the database business transaction table which held the data for the business transaction**
- **the business transaction was represented by a series of one or more 'design transactions', each represented by a message in a design transactions table that held the parameters for each design transaction procedure**
- **a new daemon process was introduced which read each individual design transaction message and processed the transaction in the database; each design transaction took at most 5 seconds, and the results which needed to be sent onto the network were stored in a results table**
- **the daemon worked by reading the design transaction message, moving it to a processed transactions table and then processing the transaction itself (in the event that a transaction failed, the transaction aborted, the design transaction record returned to the table and was subsequently re-tried)**
- **if the daemon failed, there was a 'timeout' before the daemon re-started; when the daemon completed a business transaction the results were then transmitted onto the network.**

The result was:

- **users could enter their 'work', their work was then validated to check that it could complete, and was queued to be executed**
- **as there was only a single daemon process running there was an implicit throttle on background work, enabling most of the resources to be free and available for front end activity**
- **no transaction was permitted to escalate an exclusive page lock to an exclusive table lock**
- **any failure was automatically re-tried without impacting users**
- **all work identified in any one day was entered on that day — and all work queued was completed, sometimes by as late as 22.00 (previously some work took several days to complete); in addition all users completed their work by 17.30 each day, rather than by excessive overtime being consumed by regular late working**
- **the configuration of all elements on the network was correct at all times.**

Addressing these design issues

Is there a silver bullet to address these issues? Well, ... partly.

Even good design is not perfect and testing is essential. The implementation of a regression testing harness that can do both functional and performance testing and which is run regularly as components are delivered (well before system testing) will identify problems and enable them to be resolved in good time.

I wouldn't start from here

Historically, in large organizations, separate project teams have implemented systems on a range of technologies. More recently there has been some increased commonality, particularly following the database wars when organizations finally settled on a common SQL-based database product like Oracle, DB2, SQL Server (or a few others, like Sybase in the finance sector).

But, even as databases were becoming more common, few projects shared:

- **programming standards or environments**
- **development tools**
- **data models**
- **coding tools.**

The consequence was the development of multiple separate stove pipe applications. Thus, when it became necessary to provide integration between these disparate systems, there were multiple challenges:

- **it was difficult to identify changes reliably in the system, particularly if the underlying RDBMS did not support database triggers to detect changes to database records**
- **injecting data directly into a database could be problematic if one did not access the database through the application code; this became worse if the application code was mixed with screen logic within a GUI-based application and could not be extracted and invoked from a message driven system without a major re-write.**

The result has been that too many organizations have ended up writing adapters to integrate with the existing system using adapter code which was slightly different from the core business logic of the main system. In turn this left still more scope to introduce bugs through slightly different behaviors on implementation, or by failing to modify components of code when making subsequent bug fixes.

If this were not bad enough, many stovepipe applications were fundamentally function-based rather than process-based. Even if one was able to build an adapter, there was no 'tag' to the process carried through the system to correlate processes in with processes out — demonstrating that there were no stalled business processes awaiting completion.

The consequence of these problems is that it remains difficult to re-engineer these applications to expose services which can now be accessed from an SOA. This makes the introduction of an SOA that much harder, as the earlier case study examples illustrate.

"I would rather not start from here, so let us start from scratch." Unfortunately the road to failure all too often begins here. The unavoidable challenge is that we have to start from somewhere, and work out a route forward that provides an evolutionary path for both our systems and our people.

How is it going so far? Badly

The challenge is to determine how to deal with the separate stove pipes that are deployed.

There is, unfortunately, a reluctance to re-engineer existing applications because the level of difficulty is well understood, so most organizations have started to adopt the re-write approach.

These first steps at building an SOA appear only to be focusing on the construction of objects rather than on changing the design perspective. Thus all the objects are being deployed into a single infrastructure running against a common database. All that we have created is a single large and fat chimney from our set of stove pipes, because developers are unable to break out from the functional approach of developing screen logic to access back end processes. All that is being achieved is, in effect, a change in technology, just re-writing the old applications.

There is an imperative here. We need to focus on how the design approach should evolve.

Design issues relating to SOA development

Over the last year or so we have seen an increasing discussion of the concepts surrounding a Service Oriented Architecture. Many of us in the 'integration sector' have implemented a number of the concepts associated with an SOA on projects already; we have a high degree of comfort with the underlying concepts.

Nevertheless, we find it is generally the case that we have used these ideas to address difficult problems — such as scalability, addressing performance issues or supporting large numbers of users. We implemented a reasonably complex architecture in the process.

It is our hope that an SOA will simplify these complex applications. The promise of an SOA is that separate vendors

can construct and deliver business objects which are connected only via their interfaces. In this way the idea is that one can quickly build applications by an assembly mechanism that exploits the choreographing of business processes and then configures these to invoke the underlying services that exist — by marshalling the data output from one service so that it can be carried into the next service.

This, however, is in many ways too simplistic:

- **firstly, the task of understanding the service interfaces (from different vendors) is never trivial: the requirements for mapping and transformation between data generated by one service and input to a subsequent service are almost invariably complex**
- **secondly, there is an expectation that services will operate from a common data repository; this means that either all the objects must connect to the same database with a common data model and use that data in precisely the same way, or else that there must be an underlying information integration/replication model that replicates transformations on a transactional basis.**

Both these are possible. But both involve highly complex mechanisms. Furthermore both are highly unlikely to be implemented by different vendors.

To complicate matters further, we see a parallel drive for simplicity — as if to de-skill the design and construction of applications. There is a significant danger that SOAs will be seen as a simplification mechanism for implementing the re-engineering of 'normal applications' to make better use of objects. This, in turn, runs the decided risk of failure because the underlying technology to support an SOA is more complex to set up and manage than a J2EE engine — which was the previous infrastructure for supporting applications.

To these users we suggest avoiding the use of an SOA. Using an SOA in these circumstances is another manifestation of just 'jumping on the next technology bandwagon for a technology's sake' — rather than achieving a business benefit.

Where to use an SOA

In contrast, we prefer to recommend that organizations only consider using an SOA approach where a number (preferably most) of the following criteria apply:

- **the need for an SOA has been proven by the use of a previous home-grown SOA-style approach, and the primary issue now is to provide an infrastructure that is based on industry standards so that the costs of maintaining an existing, complex infrastructure can be reduced by adopting that industry standard approach**
- **there is an acknowledged need to follow a Straight Through Processing (STP) model which supports business processes that cross substantial parts of the organization or indeed across multiple organizations**
- **there is already considerable experience of EAI within the organization and the organization understands the issues associated with implementing enterprise wide projects**
- **there are substantial program management and architectural capabilities within the organization**
- **if the level of expertise within the organization for EAI is limited, then there is some other substantial business benefit which justifies the additional costs of retaining third party experts to implement the first project and provide knowledge transfer to the internal team to enable them to take over the management, architecture and operation of the SOA**
- **there exist badly performing applications where the need to re-engineer is recognized, particularly if the business objective is higher performance or improved integrity or reliability.**

Where there is no desire for an architecture, it remains valid to use a Web Services infrastructure to underpin Internet applications. The pressure to start to develop applications any way using a web services technology for internal applications driven by technology hype will be significant, and it remains a management issue to determine how to deal with a demand for technology refresh that delivers no value to the business and value only to the programmer's CV.

Is everything a Web Service?

For some it appears that paradise is to build and deliver every function or operation as a Web Service. Is this worthwhile?

There is no requirement to implement Web Services within an application that is purely browser based and where the programmer controls both the calling and the called code.

The standard Model View Controller approach remains appropriate.

The concept of a Web Service and the WSDL definition provides an excellent and simple invocation framework for calling across the network using SOAP where a SQL call is not appropriate.

Equally, the full weight of the Web Service architecture is highly appropriate where the programmer writing (and maintaining) the service logic has no control over the application code invoking the service and must therefore provide version control and change management over the code to ensure existing programs do not fail when new releases are deployed.

Nevertheless, many people consider Web Services to mandate a SOAP implementation. It is not appropriate for good design to implement all calls from one object to another using SOAP when the objects are contained in the same application server and memory space, or within a clustered environment where more efficient lower level protocols can be used.

The Web Services invocation framework provides a mechanism by which business logic can be deployed and configured so that a Java component can have a WSDL created which can then be used to generate invocation logic that makes the optimum call for the given environment. Care is required by the designer to ensure that objects are appropriately invoked via Web Services. If in doubt it is probably preferable to define the Web Services layer of an application and require that all other invocations are 'in-memory objects' to avoid over use.

In addition to performance the other key aspect associated with invoking a SOAP based Web Service is the management of transactions. SOA frameworks do now implement compensating transactions enabling business processes that have been decomposed into a series of separate database transactions to be backed out. That said, we see very few implementations of this approach. The management of business processes, process components, transactions and compensation is a key aspect of the design of all Web Services that we allow to be published for external use.

If transaction boundaries are not carefully designed and implemented, the benefits of being able to start a transaction, make multiple database calls and then determine whether or not to commit (or rollback) the transaction as a whole will be compromised — and the database left in a logically invalid state.

Yes, all back end logic could be implemented as a .NET or J2EE Web Service call. But there are significant dangers to the integrity of enterprise, mission critical applications if the specification and design of individual Web Service interfaces is left to the junior programmer, rather than being addressed as a key component of design by the architect.

Management conclusion

The new development environments provide a productive workbench for developing new applications for the re-use of objects as if they were stored procedures.

We must be careful to leverage the productivity benefits of these new IDEs without falling into the trap of over-using features that introduce over complexity and risk into our solutions.

We must be clear whether we are just re-writing existing applications to leverage the new IDE or whether we are truly intending to adopt an SOA. It is likely that most projects will introduce the technology but will not change the fundamental design approach to derive any significant benefit from the SOA investment.

Implementing a Web Services approach against a common infrastructure will derive some improvements to the delivery and maintenance of applications, but also some vulnerabilities as all the organizations assets will be in a single environment. Implementing an SOA requires an absolutely fundamental change in the design approach. This involves focusing on:

- ***a Straight Through Processing (STP) model***
- ***decoupling business logic from screen logic***
- ***the construction of services which can be invoked by multiple clients.***

The level of design required to implement components that can operate in a multi-thread environment accessing a common database is significantly greater than is currently the normal standard today.

The potential problems identified in the case studies can be managed. But we anticipate that only a few organizations will have the capability to deliver high performance solutions that effectively exploit the SOA paradigm.

We anticipate that organizations without this capability will again fail to deliver a return on investment from an internal development.

Thus we recommend that those organizations without a proven track record of success in EAI:

- ***either focus on delivery of SOA by systems integrators if there is the potential for a positive RoI***
- ***or else adopt an SOA that is embedded in a product supplied by a vendor and supplement the SOA with components that comply with the supplied SOA.***

From all the above, Strategic Thought draws the conclusion that the adoption of an SOA must be based on a business case and not on technology for technology's sake. Furthermore, the primary application pattern for implementing an SOA architecture must be based on a Straight Through Processing/exception handling model. If any function in the business process fails it is then routed to an exception handler where the necessary intervention (preferably automated, but manual if necessary) takes place.

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 878333

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

Email and Internet

Email:
spectrum@middlewarespectra.com

World Wide Web:
www.middlewarespectra.com

ISSN 1356-9570

[incorporating FINANCIAL MIDDLEWARESPECTRA ISSN 1460-7220]