MIDDLEWARESPECTRA

incorporating FINANCIAL MIDDLEWARESPECTRA

Contents

February 2004

2 **Novartis exploits Grids for commercial benefit**
Dr René Ziegler and Professor Manuel Peitsch, Novartis

8 **OGSA: creating Grid Services from Web Services**
Anura Gurugé, Consultant

16 **Infrastructure and middleware — lessons from the front line**
Nick Denning, Chairman and CTO, Strategic Thought

24 **Remarks on managing distributed systems and middleware**
Peter Bye, Consultant, Unisys Systems & Technology

32 **The test of time**
Dr Keith Jones, IBM Software Solutions Worldwide

40 **Java IDEs versus Visual Studio .NET**
Tom Welsh, Consultant

46 **How 'New Grids' move beyond 'Old Grids'**
Mark Lillycrop, Principal Analyst, Arcati

Volume 18 Report 1

Novartis exploits Grids for commercial benefit

Dr René Ziegler and Professor Manuel Peitsch
Novartis

Management introduction

René Ziegler is responsible for the global IT governance function for Novartis Biomedical Research. He is based in Basel, Switzerland and he was responsible for the integration of the research and information management functions when Novartis was formed out of the merger with Ciba-Geigy three years ago.

Manuel Peitsch, also based in Basel, runs the infomatics and knowledge management department for the Novartis Institute for Biomedical Research. This encompasses everything from delivery of IT systems to its partners to the application of techniques like artificial intelligence, in-silico science and knowledge engineering. It also includes the delivery of paper and electronic libraries, databases, tailored information reports, etc. to all of Novartis, including the Consumer Health Division. The reason for this diversity is that anything to do with health involves information in some form, whether analysis, patent analysis or even regulatory reporting.

In this case study, Dr. Ziegler and Professor Peitsch discuss what Grid computing is already delivering for Novartis. They also describe how, in their view, there may be few problems or challenges that are not susceptible to Grid computing — even in commercial activities.

Finding Grids

How did we find Grids? This could be a long story, but we will try to keep it short. I (Manuel) worked at GSK (or GlaxoWellcome, before its merger with SmithKline Beecham). At GlaxoWellcome we were doing similar things to what Novartis is doing now, except on a smaller and more fragmented scale. We were oriented to using both Windows and Linux workstations in a Grid setting. This made me think about what was possible with Grids.

By the time I joined Novartis some of these ideas began to gel around a strategy for high performance computing that was then being evolved in Novartis. Part of that strategy was to engage and invest in PC-based Grids.

The interest came from both the business and IT sides. From the business side, we could see that pharma research was going be increasingly compute hungry — that simulation was likely to use more and more computing resources for applications like in-silico modeling of new compounds. Similarly, the complexity involved when simulating biological processes was clearly going to be immense if we were to do the research of dockings or interactions into the molecular level between entities that is now necessary.

As you might imagine, these are compute-intensive problems to address. Using conventional techniques — with large dedicated SMP machines, for example — we knew we could spend huge amounts of money if this was to be our path to providing a super or high performance computing infrastructure for our pharmaceutical research.

Thinking laterally

When we started looking around at Grids and thinking laterally, we realized that we already had a virtual supercomputer almost at our disposition. Once we added up the computing capability of all the PCs that the Institute alone owns, which stand idle for so much of the day (and night), we understood that we had an opportunity, if we could only realize it (Figure 1.1).

At that time the Institute had about 2700 desktop machines. We calculated that there was up to 80%-90% of time available on these. By scavenging these unused compute-cycles, we would have access to immense amounts of raw compute-power. The challenge was to harness this so that the problems we wished to solve could be run in a highly distributed and secure fashion. If we could do this we knew we would have an immensely powerful computing device at little incremental cost. [Those 2700 desktop PCs do not include the large numbers of laptops that also exist. In the future it is our intention to harness these as well. But, because these are mobile, we have some additional issues to resolve before this is practical.]

Within the overall Novartis organization, we probably have a further 25000 PCs that could be brought into a Grid environment. These, as yet, are not part of our 'validated Grid environment'. But these have the potential to be harnessed in due course.

Power, and delivery

We worked on what harnessing these might mean in processing power terms. Such calculations are, of course, largely theoretical. Nevertheless, our peak performance estimation is that our existing 2700 desktop PCs have the capability to provide 5 teraflops of processing. With ten times as many PCs harnessed, we think we would have well over 50 teraflop performance — which would put us at near the top end of the supercomputer rankings, at something like Number 2 to Number 4 in the world.

It is, of course, one thing to be able to state these figures. It is something else altogether to be able to use such power meaningfully on truly value adding work. To be able to access so many potential cycles we realized that we needed to implement a consistent environment on every participating desktop PC.

Here the Novartis past investment in consistency paid off. We were able to exploit two activities which nicely blended together, namely combining the Grid's possibilities with the past corporate decision to standardize our PC environment. The latter step was hugely significant. We have invested in some 60000 new PCs (laptops and desktops) in a coherent Company-wide approach. One advantage is that, wherever you are working, you know that you will have a system which you can recognize.

Equally, we know — as cycle scavengers — how each desktop (to start with) is configured. This means we know what the PC systems look like. We are not having to run around trying to find out how each machine is different or why. This is of enormous assistance. To us it offers a nice demonstration of the benefits that an integrated strategy can deliver. Without these standardized PCs, realizing our Grid virtual super-computer would be hugely more difficult.

In practical terms, the standardized PC comes in one of three models that people can choose from:

- **a standard desktop**
- **a lightweight laptop**
- **a heavyweight laptop.**

That was the hardware choice. But what was at least as important is the software image that is copied onto all these PCs. This is identical across the whole Novartis organization.

Only once this has been loaded can the applications that are customized for individuals be installed — and these are appropriate to your function, whether you are a product development specialist in the US or a research scientist in Switzerland or a sales agent in the South Pacific area or whatever.

The Novartis standard PC image contains many pieces of software. It is based around Windows XP and Microsoft Office. It has email and communications and some additional utilities, for example the ability to receive and install software pushed from a central server.

It also has the Novartis PKI infrastructure with encryption and security layered to the relevant functions. That is roughly where the basic layer stops.

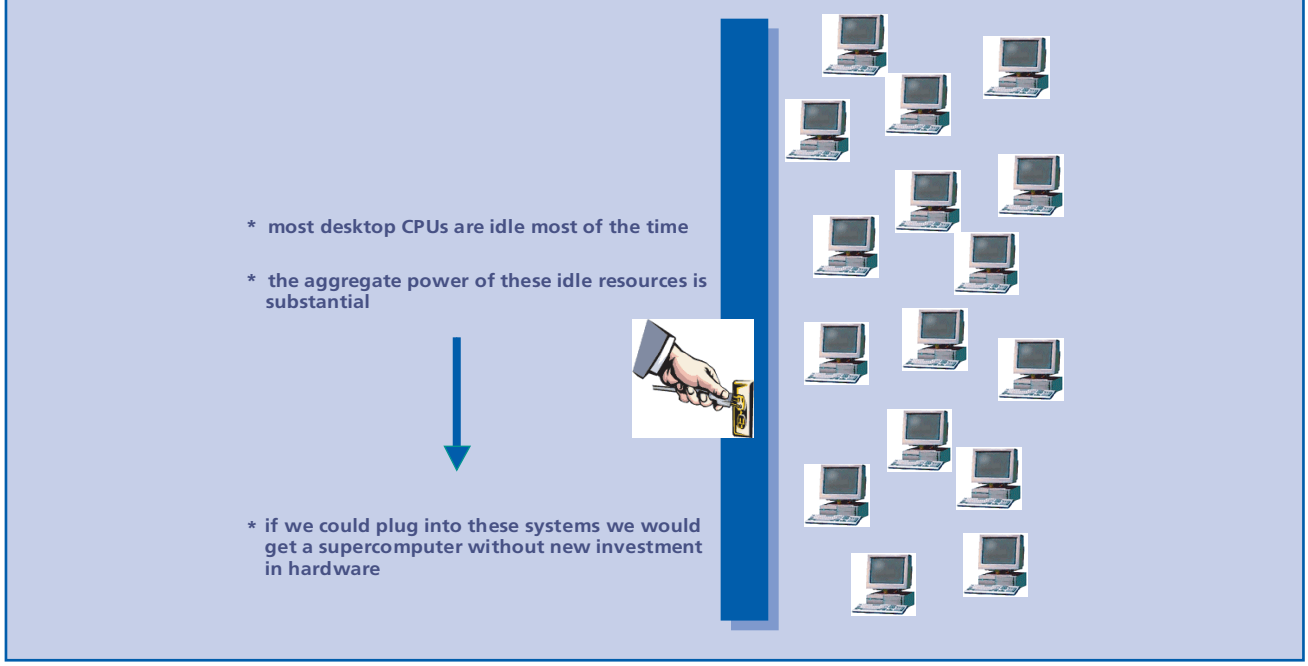
Today, at this stage in time, our Grid capabilities are not part of the standard image. Instead these Grid facilities are an additional set of functions — not unlike a specialized application — which are added once the standard image has been installed. In the future, we have the option to add Grid functions to the standard image — and will likely do so once we move out beyond the 2700 desktops in the Research Institute.

Grid-enabling Novartis PCs

One of the benefits of the standardized image is that it enables us to push software packages from a central location out to desktop, or other, machines. Essentially our Grid uses a meta-processor: this is like an application which we push to PCs running our Grid-validated environment. The meta-processor software — in effect a processor within a processor — is purchased from United Devices. Once this Grid software is installed, we can start to exploit its cycles.

The clever part of all this is, of course, the combination of being able to push out functions to the standardized platform when combined with the meta-processor software. This took us some months to refine into a Novartis 'product' that is stable enough to operate in an industrial environment. The challenge was less about the fundamentals than about some aspects of the technical environment. Anything associated with our Grid that executes on the desktop must do so in a way that the desktop user cannot access. Imagine a bubble or balloon in the memory/processor which is devoted only to Grid processing. We had to be sure that this could not be compromised, by the PC user, in any way.

Figure 1.1: Making the most of processing power



In addition, there were aspects like job scheduling, data downloading and results uploading (Figures 1.2 and 1.3) which all had to be considered and resolved in a secure manner. For example, we had to link our Grid function into our PKI security infrastructure.

The result is that, in principle, it should not matter if it is a research scientist in Basel or a sales person in Asia who is Grid-processing data. The important dimension is that neither has any idea about what is executing in their local Grid environment (all that they are aware of is that the processor is working harder than it might otherwise do). In practical terms, we have this running on our 2700 Institute desktops today.

Alternatives, and cost effectiveness

We did look at possible alternatives — from Linux clusters to supercomputers to SMP systems to dedicated proprietary clusters. All of these were immensely expensive, not least in terms of the specialized resources and people that would have been needed.

After a year of successful running we have tried to place some cost benefits on what we have done. The smallest 'benefit' is US$2M. This is calculated as the simple cost of enabling 2700 PCs compared to buying a Linux cluster of the same power (but without including the cost of a Linux cluster data center or of the extra heat and air flow handling or of the additional people for support or of the extra power consumption, etc.). At the other end of the scale, our biggest estimate is ten times as great — or more. Using other analyses, we estimate that we are enjoying a 5-fold to 50-fold cost avoidance. But, remember that this only applies to the 2700 Research desktops. Think about the possibilities when we go out to the other 25000+ machines.

The experience, so far

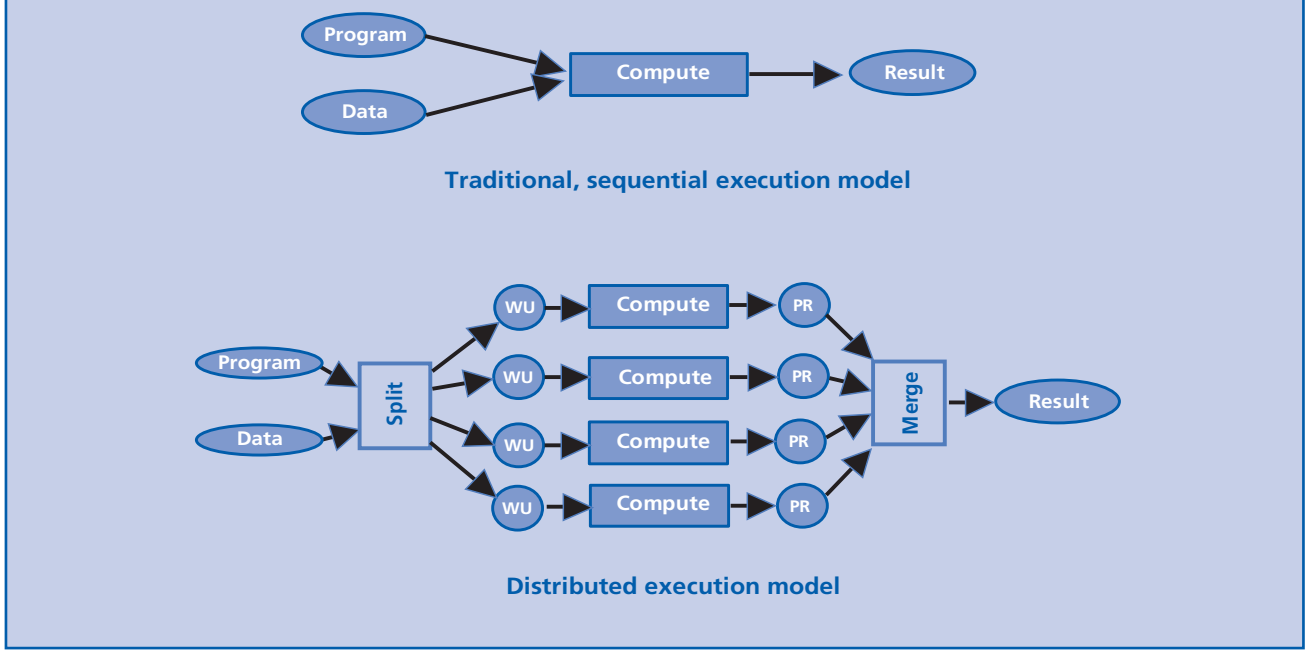
Our experience so far has been brilliant. People who use it say they cannot stop being amazed by the possibilities. These are people with 'big compute' problems to solve — often ones which they never thought that they would have the raw power to address.

As for the practicalities, even issues like job scheduling have been made simple. Essentially, if you are authorized to run jobs on our Grid, you submit these to a central queue and they are executed — often far faster than anticipated. So, if I am a user who wants to do some big computation, I queue up my job and wait until the results are delivered back to me. It is as transparent as that.

Grid-specific or not

One of the early issues that concerned us was whether it would be difficult to describe problems in ways that were

Figure 1.2: Grid-enabled distributed processing



computationally appropriate for a distributed Grid. If we could not break down the computations into units that could be parceled out and shared across 2000+ processors, then the whole initiative would not be useful.

As it turned out, we were unduly concerned. For many applications we discovered that we did not have to do much special handling of code or algorithms or optimization to Grid-enable our existing applications.

Let us give you an example. It only took 2 hours for the first application to be implemented on our Grid. One of our master developers — in computational chemistry — took just 2 hours to rework an existing application and have it run on the Grid. In effect what he did was arrange for that same application to run on each of the 2700 desktops. In fact, it was running multiple times — about a million times — spread over those 2700 machines. What we now had was the ability to deliver huge degrees of parallelized computing even within one machine.

Of course, it was not quite so straightforward everywhere. In other cases what we needed to do was to revisit algorithms, particularly those ones which had been written before the advent of parallel computing. In these areas — especially in statistics — we found that a degree of rethinking and reworking was possible so that we could parallelize those algorithms. Once this was done, we could then parcel out the computations to our Grid.

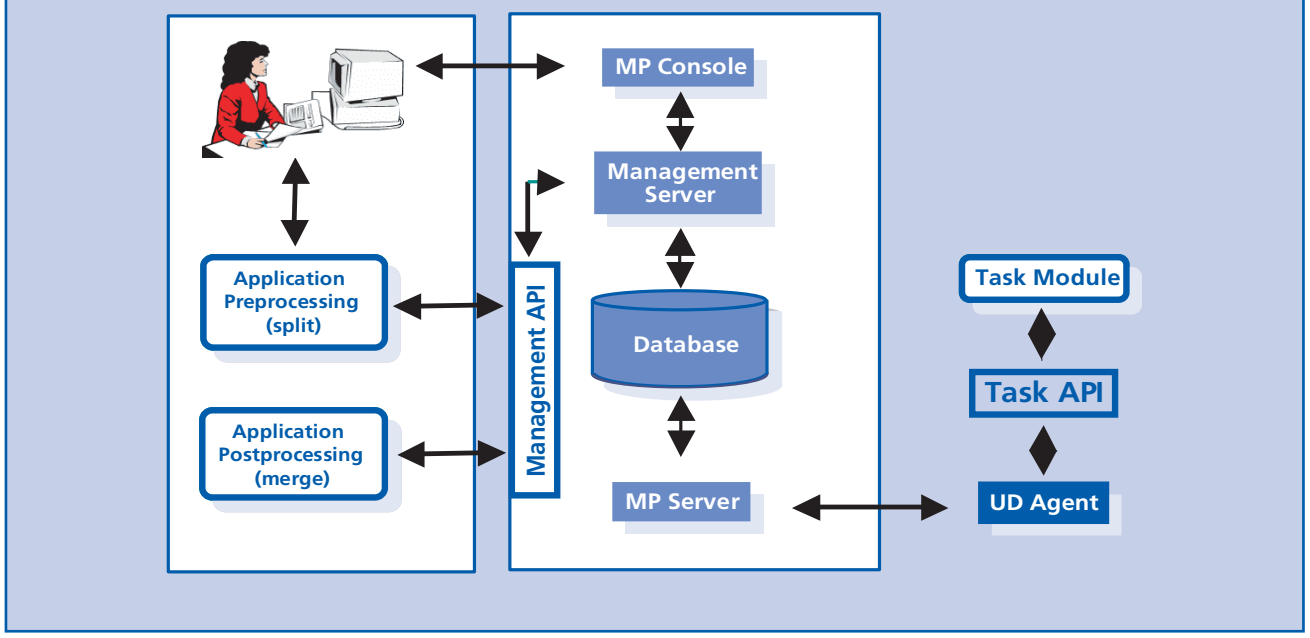
In practice, we have found this works extremely well. On the other hand, in other areas, there was no avoiding the need to do work to optimize code for our Grid. A different aspect that relates here is that a user can decide how many systems he or she wishes to involve. They may want to exploit all 2700 machines or prefer to choose to use only 100 or 500. This provides us with additional degrees of scheduling flexibility.

Downsides, misperceptions, economics and new possibilities

The largest downside, or problem, we have had is about incorrect perceptions. Convincing people that Grids are relevant to their problems is not always easy. For example, there are those who believe that quantum chemistry cannot be handled by a Grid like ours. The standard approach is to say that you need large dedicated systems with large shared memory resources — for example, using big iron like IBM P/690 machines.

We are challenging our team about this. We have asked them to come back with a solution to work around that problem. That said we do expect a solution. We do not think that we have yet discovered the limitations of where Grid computing can be applied. Indeed, we are increasingly convinced that nearly every (if not all) computational problems can be parceled out in a suitable way for computation to be undertaken on the Grid.

Figure 1.3: Application splitting and merging



Here economics cut in. Running on a Grid might only be 70% or 50% as efficient as on a P/690, for example. But the problem with something like a P/690 is that you cannot afford much more than the 256 CPUs because of the sheer cost of the hardware. If you have got 2700 CPUs and even if you are 50% less efficient, it still means you are something like 5 times faster than with 256 P/690 CPUs. Our conclusion is that we may well accept some computational inefficiencies but this will not matter because we have so much brute force behind us.

Lessons learned

Our first lesson learned is very simple to state, but is not always 'believed'. This is to ensure that the users of PCs (whether desktops or laptops) understand that these systems belong to the Company and not to them: they are not personal possessions but corporate ones whose resources can be allocated as the Company sees fit. The relevance applies to placing the meta-processor on each machine and using the unused computing cycles. You could call this a management issue but it is also a quality issue.

The second lesson is to demonstrate — probably using experimental pilots — that the Grid concept actually works. Our experience, with the first application only taking 2 hours of work before the benefits flowed in, was immensely useful in establishing credibility with the computational chemists and others. After this it is important to proceed in a step-wise manner, gradually and without biting off too much too soon.

Another lesson is that providing a framework in which people are helped to think 'out-of-the-box' matters. If they can do this, you enable them to think up ways they might use the Grid approach to solve problems that they might never have dared to think were possible to solve computationally before. We are now doing computation in chemistry and biochemistry that we could not even dream of affording to do a few years ago — just because the potential of what is there. Too many hard computational problems in science are not being tackled because everybody 'knows' that the computational power is not available. We have disproved this — to our competitive advantage.

On a more management thought, Grids do need a champion who is willing to go ahead and start with a small-scale and controlled pilot or experiment that is contained in its impact if something should not go right. Demonstrating what can be done is much better than talking or presenting. Showing people that a task that took 8 days can be done in half a day has an amazing impact. It breeds interest and involvement — especially when we have been able to show that there is neither a paradigm shift nor a major change needed to be able to exploit Grids.

Management conclusion

Novartis is a commercial organization with profit as one of its objectives. As Dr. Ziegler and Professor Peitsch demonstrate, Grids are not just for the academic or scientific community. They are also for any organization that is prepared to think about how it might be able to use the power of a Grid.

Perhaps even more startling, for the commercial community, is their belief that most challenges are susceptible to processing on a Grid, even ones that many might think are wholly in the realm of business processing (like marketing and sales analysis). If this is correct, then Grids will open up huge possibilities to use existing resources to handle peak loads with existing assets.

OGSA: creating Grid Services from Web Services

Anura Gurugé
Consultant

Management introduction

Web Services, the much vaunted but as yet still emerging 'next big thing' from the application software arena, has now assumed a pivotal role in Grid Computing thanks to the Open Grid Services Architecture (OGSA). OGSA, which was formalized by the Global Grid Forum in June 2002 — and has since been implemented in the Globus Toolkit 3 — is a standard for the overall structure and services that are to be provided in future Grid environments. With OGSA, all Grid resources, whether logical or physical, are represented as Services — where a Service is considered to be a network enabled resource that delivers a predefined functional capability that can be invoked and utilized through the exchange of specific messages.

*In this analysis, Anura Gurugé (the author of the book **Web Services: Theory and Practice** — published by Elsevier) examines how Grid Services and Web Services 'belong together'. He also describes how one might think of creating Grid Services from Web Services — something that many commercial as well as non-commercial organizations are likely to want to do in the next five years.*

OGSA

OGSA, as illustrated in Figure 2.1, is built upon Web Services. OGSA in effect augments Grid computing with Web Services. Or, to be more precise, Web Services and related technologies — such as XML, SOAP, Web Services Description Language (WSDL), WS-Security and WS-Coordination — augment Grid Services.

The fundamental building block of OGSA is a 'Grid Service', where such a Grid Service is essentially an enhanced Web Service. Thus everything to do with OGSA revolves around Web Services — so much so that Globus now positions OGSA as representing the evolution of Grid computing towards a Grid system architecture based on Web Services concepts and technologies. OGSA in essence leverages Web Services slowly but surely to move Grid computing towards a Service-Oriented Architecture (SOA) model.

Web Services, however, contrary to what is implied by their name, are not a predefined set of software services. Web Services per se are not a standardized mechanism, or a middleware repertoire of functions, for accessing services over the Web — as presumed by many. In contrast, Grid Services, by and large, are clearly specified functional entities intended to facilitate various facets of Grid computing — whether (say) this be inter-program messaging, queuing, event logging or credential validation.

Thus, just to say that OGSA Grid Services are based on Web Services can be misleading and confusing. Figure 2.2 sets out to provide a first-cut, relatively high-level clarification of the relationship between these two types of service by comparing and contrasting their core characteristics.

Introducing Grid Services

With Figure 2.2 highlighting how Grid Services are in so many ways different to Web Services, it is now appropriate to take a more detailed look at Grid Services. Everything in OGSA — for example, the OGSA architected services (as shown in Figure 2.3) — are made up of various sets of Grid Services. At present there are four categories of such OGSA architected services:

- **Grid core services**
- **Grid program execution services**
- **Grid data services**
- **domain-specific services.**

Grid core services, as indicated by their name, are the pivotal backbone services that address the mandatory requirements of a 'request-response-oriented' Grid computing architecture — irrespective of whether one wishes to use:

- **a 'request model', where a client contacts one or more servers requesting 'units-of-work'**
- **a 'dispatch' model where a server contacts various clients with new 'units of work'.**

The types of services embraced, as a minimum, within the core category include:

- **service management**
- **service communication**
- **policy management**
- **security.**

It is assumed that all OGSA-compliant Grid applications — as well as the other higher-level OGSA services — will avail themselves of these fundamental services. They are needed to support distributed program execution or dispersed data virtualization.

Service management, vis-à-vis the 'core' category, will provide the necessary functionality to automate and facilitate remotely deployed services. These services will deal with remote installation, maintenance, tear-down, service monitoring and 'on-the-fly' troubleshooting. It is also expected that there will be services, within this category, for collecting, analyzing and disseminating 'statistics' about the operation of each individual Grid.

The service communications-related functions, within this category, set out to facilitate all possible permutations of pertinent inter-service communication mechanisms — including queued message exchange, publish-subscribe and event notification. A distributed event logging system will also be available as a part of this service repertoire.

The policy management functionality sets out to provide a powerful and flexible policy creation, administration and enforcement framework that can then be used to govern policies pertaining to all other aspects, including:

- **security**
- **resource allocation**
- **performance**
- **event reporting.**

The obligatory and crucial security services class sets out to offer a comprehensive arsenal of all necessary security related functions, including:

- **authentication**
- **access control**
- **credential management**
- **etc.**

Program execution and data services

The Grid program execution services, at least at present, are intended for Grid applications that support and rely upon parallelism, high-performance computing and distributed collaboration. For such applications, this category of services will handle the specialized tasks related to job scheduling and workload management specific to the exact dynamics of the processor resources in play. The Community Scheduling Framework (CSF) specification — being worked on by the Global Grid Forum (GGF) — falls into this category of services.

Grid data services are intended to complement the program execution services by providing all the necessary functionality for data virtualization. These services are meant to provide controlled access to authorized data independent of the location of the data and the means by which it is being stored. Thus data access using these services will cut across disparate data storage technologies and provide relatively seamless access to data located within flat files, databases, application-specific documents and Web pages.

The bottom line here is that OGSA Grid Services fall into the conventional 'middleware-centric' paradigm that one normally assumes with application-related services (like print services, file services, timing services, etc). These Grid Services sit between Grid applications and Grid resources (for example, processors and storage). They provide an architected and systematic means by which Grid applications can exploit various resources and capabilities available within a specific Grid environment.

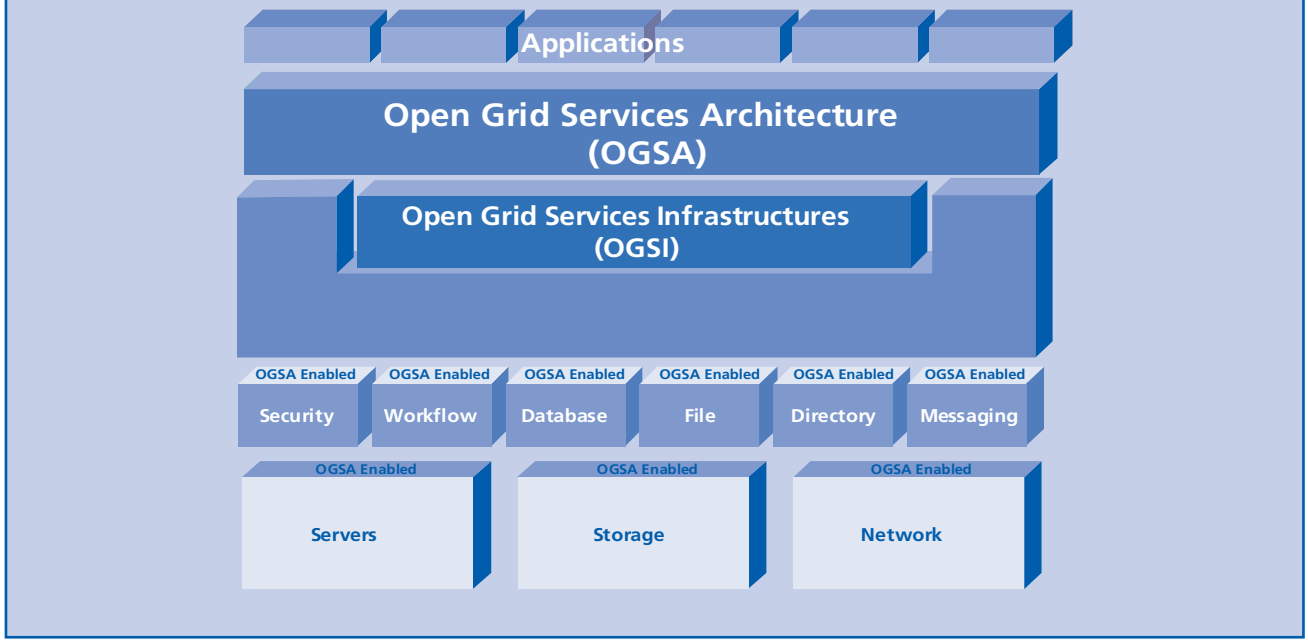
The reason it is important to spell this out at this juncture is that Web Services do not fall into this conventional application services paradigm. In the Web Services arena, there are no predefined services for service management or policy management. One can define such services — and possibly even implement them as Web Services. But, as of now, there is no architecture, à la OGSA, that spells out the various categories of core and value-added 'functional services' in the Web Services arena.

Grid Services = Web Services + OGSI

Despite this basic difference in outlook between these two clans of services, Grid Services are built upon Web Services. What this means, as will become clear once Web Services are exposed and dissected in the next section, is that Grid Services are a special class of Web Services.

In other words, Grid Services will be implemented in the form of Web Services. Or, to be ultra pedantic, they are a type of extended Web Services. For example, a Grid service for workload management on IBM p/Series UNIX machines would be implemented as a Web Service: this would be invoked using SOAP where the inputs and outputs will be XML documents that are interchanged with the calling application (using SOAP).

Figure 2.1: OGSA and OGSI



OGSA defines a Grid Service as being a Web Service that:

- **conforms to a set of Grid-computing related conventions**
- **furthermore supports a set of standard API-level interfaces.**

These conventions and interfaces — which represent the overall interaction model for Grid Services — are defined by the Open Grid Services Infrastructure (OGSI) specification. As shown in Figure 2.1, OGSI thus becomes an auxiliary, functionality-enhancing interface between Grid Services and Web Services technology. Thus, as a rule of thumb, one can think of Grid Services as being made up of extended Web Services that have been augmented with OGSI.

The OGSI interaction model provides software developers in the Grid arena with a standardized and universal mechanism through which to interact with Grid Services. OGSI, at present, provides interfaces for:

- **service discovery and identification**
- **life cycle management**
- **service creation (referred to in OGSI argot as a 'factory', in that it creates new services)**
- **service tear-down**
- **servicing grouping (or indexing)**
- **event notification.**

Figure 2.3 extends Figure 2.1 to show the Grid Services, OGSI functions and the notion of extended Web Services discussed earlier. Note that in Figure 2.3, the OGSI term ' HandleMap' is used to describe the service identification function that goes hand-in-hand with that of 'factory' in that this function helps locate an instance of a newly created service.

Exposing Web Services

The term 'Web Services' — though technically accurate — is frequently misleading especially when used in scenarios, such as OGSA, where the term 'services' is also being used to describe utility backbone, 'middleware' functions. Web Services are standards-based, re-usable software components. They are not predefined services. Nor were they intended to be predefined services. Instead they represent a component methodology.

As such, Web Services are the latest incarnation of the remote function invocation-based distributed computing paradigm postulated in the past by the Distributed Computing Environment (DCE), CORBA, DCOM and even UNIX RPC. The only major technical differences are that Web Services are:

- **totally XML-based**
- **their functionality is describable using WSDL (and UDDI, though UDDI can be used to**

Figure 2.2: Common traits, and differences between Web and Grid Services

Common traits

- client/server model
- XML-based
- platform independent
- programming-language agnostic
- relies on WSDL and WS-Inspection for describing the service being offered
- uses SOAP for service invocation, message exchange and data encoding
- able to exploit optional XML-based services like WS-Security, WS-Coordination and WS-Policy
- can be used on an Intranet, Extranet or Internet basis (i.e., they are location transparent)

Differences

Web Services

- modular, re-usable software components
- meant to provide additional value-added functionality for applications
- no pre-defined services per se
- no external interfaces other than the I/O performed via XML document interchange
- no standardized scheme for life cycle management
- no notion of QoS, whether it be for response time, scalability or reliability
- meant to be consistent, with persistent services available on-demand
- no formalized concept of multiple instances of the same service being provided by the same service provider
- no mechanism whereby a new service or an instance of an existing service can be investigated from an existing service
- management is still nascent/embryonic
- does not worry about 'state management' since each service is expected to execute and deliver a standalone, pre-defined function
- exploit UDDI to advertise and locate services

Grid Services

- services related to server, storage and network resources for use by applications
- meant to facilitate the collaborative processing of applications
- predefined and clearly articulated sets of services
- set of standard interfaces per Grid Service provided by OGSI
- well defined scheme and OGSI interface for life cycle management
- concept of QoS
- services can be transient or persistent
- possibility of having multiple instances of the same service within one Grid Computing environment
- OGSI includes a 'factory' interface that can be used to create new Grid Services
- though UUDI may be used, the need is not as pressing as with Web Services
- significant emphasis on how services are managed
- uses an extension to WSDL 1.2, referred to as Grid WSDL (GWSDL)
- uses Web Services or, to be more precise, XML, SOAP WSDL, etc., as a service definition and data exchange mechanism

describe other non-Web Services related services including services that are not even computer related, for example, rare book appraisal)

- **they support SOAP as their preferred communications means — where SOAP, a transport independent messaging scheme, can be used on top of ubiquitous HTTP to provide an easy (but at the same time potentially dangerous) interoperability with Web servers, firewalls and e-applications**

Web Services are, therefore, a Web-oriented, RPC-like, remote function invocation methodology where all of the data transfer and I/O interface definitions are done in terms of XML — given that WSDL is an XML derivative. It is these XML, RPC and SOAP-based transport 'piggy-backing' capabilities of Web Services that make them attractive as the basis for OGSA Grid Services.

The quintessential types of functionality, that are routinely associated with Web Services, include customer warranty status look-up, package delivery tracking, credit card authorization, tax or VAT calculation and flight schedule retrieval.

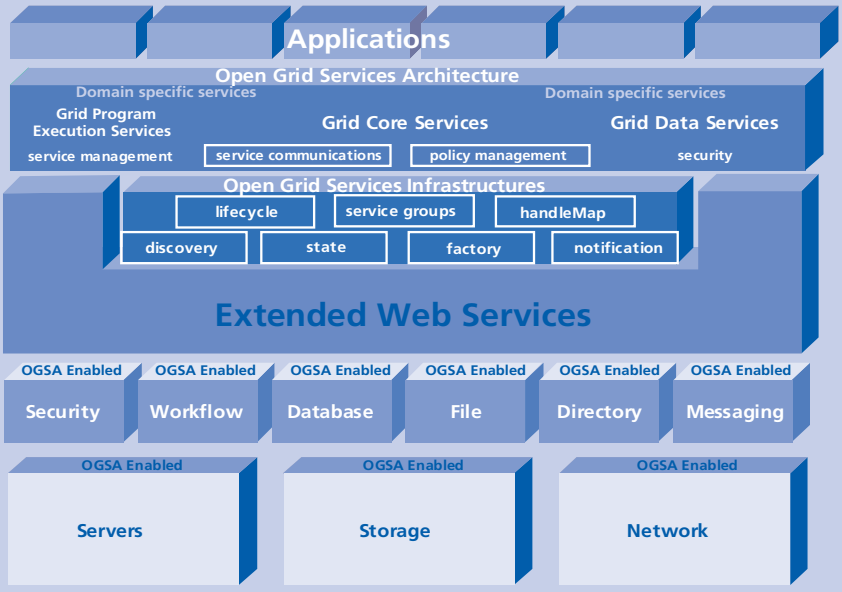
Note the absence of the typical, 'low-level', software development related topics such as database services, file services, print services, timing services, etc. Providing this type of functionality is not what Web Services are intended for. Their goal is to deal with higher-level, problem-solving functionality. In the context of Web Services, the lower-level, system- or network-oriented services will be provided by suitable application development and application execution environments — such as IBM's WebSphere Studio and Application Server or Microsoft's Visual Studio .NET 2003 and the .NET Framework.

Also note that SOAP, WSDL and UDDI are not included in the above list. This is because SOAP, WSDL and UDDI are not Web Services. Neither are WS-Security, WS-Trust, WS-Co-ordination or Business Process Execution Language for Web Services (BPEL4WS). All of these are enabling methodologies (or protocols) that are used by Web Services.

Unfortunately this distinction has been lost in the past — especially in the media. It is far too common to see headlines like 'Intel to support Web Services'. Given that Web Services, in reality, are application-level software modules (rather than SOAP or WSDL), to say that 'Intel will support Web Services' is roughly equivalent to saying that Ford has made a high-level, corporate decision to support spark-plugs and shock-absorbers in the future.

You should now have a sense as to why Web Services have encountered more than their fair share of confusion and also why it is important to make clear a distinction between Grid Services and Web Services. They are very different beasts, even though the former are built upon the latter.

Figure 2.3: Extending the diagram to include Grid Services



Web Services in practice

Figure 2.4 highlights the notion of Web Services being high-level, problem-solving functionality that facilitates and expedites the development of new, feature-rich applications. Figure 2.5 illustrates how and where XML, SOAP, WSDL and UDDI fit into the big picture vis-à-vis Web Services.

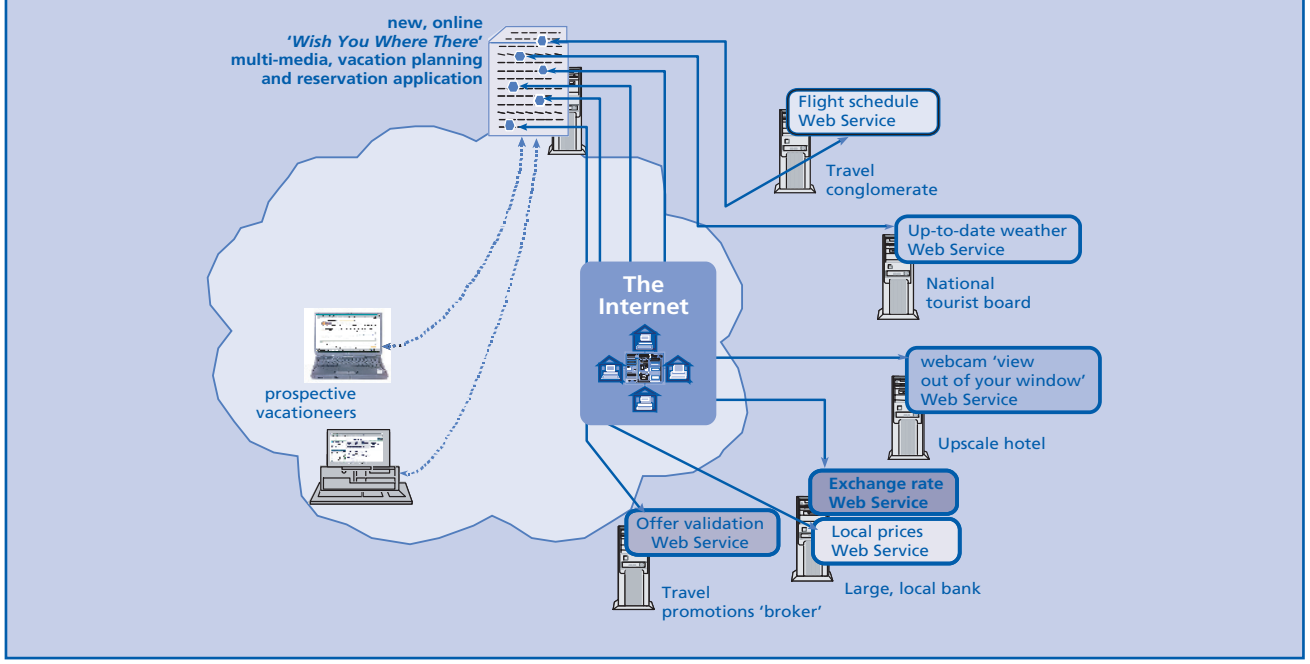
The Microsoft MapPoint Web Service [http://www.microsoft.com/mappoint/webservice/] serves as a great example of its genre. It is a powerful, feature-rich mapping application in its own right that delivers multiple geographic location related functions including map rendering, driving directions and proximity searches. It is Microsoft's equivalent of what is available at 'Mapquest.com' and is also the mapping software featured on MSN for those looking for directions, street maps and driving directions.

All of this functionality, however, is made available as a single, monolithic, XML-based Web Service. Applications that wish to offer state-of-the-art mapping capabilities do not have to try and reinvent the wheel or be forced just to provide a link to an external service such as MapQuest. Instead, the application can remotely invoke Microsoft's MapPoint Web Service, on a pay-per-use-basis, and pass it the necessary input parameters for a particular function (for example as a zip code or map co-ordinates) in the form of an XML document. The MapPoint Web Service will be available, with Microsoft acting as the service provider, on one or more servers — most likely running Windows 2000 Server or Windows 2003. The exact parameters needed to invoke a certain function will be available in WSDL form and one will be able to located this service, from scratch, by entering 'map', as the required service, at a public UDDI Business Registry node (say uddi.microsoft.com or uddi.ibm.com).

At this juncture it might also be worth visiting the Google home page. Look under 'services and tools' on offer. Under Google tools, one should discover an offering entitled 'Google Web API'. This too is a SOAP and WSDL-based bona fide Web service — intended, in this instance, to provide application developers with the capability to include all of Google's legendary Web page search capability within applications they are developing. When used as a Web Service, the trade-mark Google interface is no longer visible to the end user of the calling application.

Instead, all of Google, with its 'data warehouse' of 3 billion or more indexed Web pages, becomes a single Web Service vis-à-vis the calling application. This notion of Web Services as being high-level, feature-packed applications, as opposed to low-level utility functions, can be conclusively cemented at this point by highlighting that Amazon.com now offers the entire e-commerce capabilities of its burgeoning site as a single, XML-based and SOAP-driven Web Service.

Figure 2.4: What Web Services brings to Grid Services



The bottom line here is that Web Services extend the now common-place and compelling Web paradigm to embrace software component technology. As a technology they provide a standardized, programmatic equivalent of the hitherto interactive Web experience. Popular services routinely accessed by browser users (like the Google search or an Amazon query) can now be packaged and delivered for use within other applications to create even more sophisticated, highly-integrated applications for e-business, corporate portals and even intelligent phones.

What Web Services brings to Grid Services

By now it should be becoming clear that Grid Services do not use (or utilize) Web Services — at least in the conventional sense of layered services with higher-level services exploiting services available at lower levels. Instead, Grid Services are implemented as Web Services. Consequently Grid Services are Web Services for use by Grid applications that require OGSA specified services.

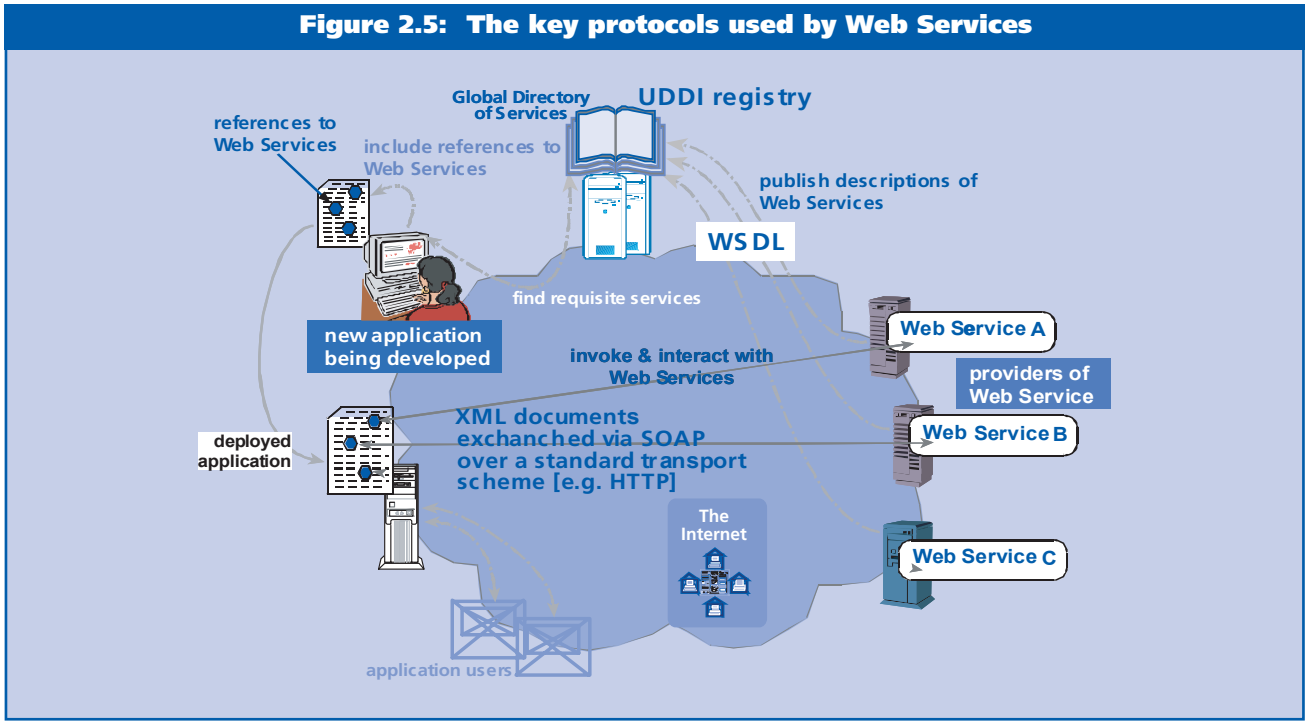
Instead of coming up with new schemes for specifying Grid Service interfaces, exchanging messages and data between services and remotely invoking services across diverse transports, the creators of OGSA decided to exploit the mechanisms already available within Web Services. They took what was available in terms of WSDL, SOAP, XML and the various nascent auxiliary specifications (like WS-Security, WS-Trust, etc.) and manipulated and augmented these now de facto standards to realize their needs.

In effect they took the proven, XML-centric service definition model available with WSDL 1.2 and extended it to cover the needs of OGSA with the GWSDL extension — which consists, in the main, of a Grid computing specific namespace and a new 'gsdl' tag (where a namespace is a standard XML mechanism to guarantee the lack of ambiguity of elements specified in an XML document without in anyway sacrificing the scope or extensibility of XML or an XML document).

The capabilities that Web Services provide to Grid Services can, therefore, be summarized as follows:

- **a standard interface definition mechanism (for example, XML and WSDL/GWSDL)**
- **the ability to exploit the global, service advertising power of UDDI using either the Web-based Universal Business Registry or semiprivate 'enterprise' UDDI implementations (for example, the UDDI functionality included within Windows Server 2003)**
- **the capability, via SOAP, to interact uniformly with remote services across multiple disparate transport schemes (for example, HTTP, FTP, TCP, SMTP, etc.)**

Figure 2.5: The key protocols used by Web Services



- **a powerful, platform and transport-independent RPC mechanism available with SOAP**
- **platform and programming language independence**
- **multiple implementations of the necessary methodologies (for example, SOAP, WSDL, UDDI, etc); these are available on diverse platforms in the form of IBM's WebSphere, BEA's WebLogic, J2EE's Web Services Developer Pack and Microsoft's .NET**
- **the flexibility to be totally transparent to network topology and node configuration (and thus to be capable of working uniformly on Intranets, Extranets and the Internet in any acceptable one-to-one, one-to-many, multicast, work flow or many-to-many node configuration)**
- **wide awareness, appreciation and endorsement within the software development community — particularly so in the open-source and Java camps.**

Have Web Services fizzled?

Web Services, though they have not yet fizzled, are indubitably a long way behind schedule in terms of living up to their once lofty expectations. When they were conceived in mid-2000, Web Services were touted as being the next big thing vis-à-vis the Web. Web Services were expected to revolutionize software, e-business and the entire Web experience — and to do so rather quickly.

It should, however, be noted that these predictions were being made during the exuberantly heady days when the stock market was trying to defy gravity and logic and when dot.coms apparently represented the next frontier of global commerce. The good news is that the true potential of Web Services is still there, and that the apparent lethargy on the adoption front can be rationalized and justified. Moreover, many of the factors that have stalled the adoption of Web Services do not apply to Grid Services.

Without doubt, the primary reason for Web Services not as yet fulfilling their expectations lies with the economic slowdown that has gripped the globe since early 2001. Web Services are an application enablement mechanism. Thus, they can only really flourish when the application software development sector is on an upswing. With IT budgets under siege, this is not currently the case.

The other big factor that has impacted Web Services, post 9/11, has been security concerns. Ironically this is where OGSA has already done a better job with Grid Services than what the Web Services protagonists (which obviously include IBM, Microsoft, BEA, SAP, et. al.) have done with Web Services.

Web Services, as yet, lack the equivalent of the Grid core services — particularly when it comes to service management, QoS, life cycle management, credential validation, event logging and policy enforcement. There are multiple efforts under way to create new specifications that will cover all these bases and more — but it is all still theory as opposed to practice. But, fortunately, OGSA Services should be able to leap-frog these current deficiencies since they intend to augment what is missing and furthermore come up with Grid-specific Services in such areas.

Management conclusion

OGSA brings the best of Web Services methodology into the Grid arena in the form of Grid Services. Grid Services, therefore, become another class of killer application for Web Services alongside portal and smart-phone applications. The introduction of Web Services into the Grid arena also highlights intriguing scenarios where Web Services and Grid Services can work in tandem — each addressing a very different need for each application.

Web Services, as repeatedly stressed by Mr. Gurugé, are a means for adding high-level, feature-packed functionality (for example, sophisticated mapping or Google searches) into an application. Such new, function-laden applications could be ideal candidates for Grid computing — given that they will require considerable processing power, especially if they are being accessed simultaneously by a large number of users across the Web.

One can now envision a scenario where an application will be using OGSA Grid Services (which are implementations of Web Services) to make use of virtualized processing power — while at the same time using Web Services remotely to access application functionality from other servers. Thus Web Services will be at play at two levels, at:

- ***the application-level***
- ***the lower, backbone utility level.***

The bottom line is that Web Services and Grid Services together represent the new face of heterogeneous, distributed computing. They complement each other, with Grid Services adroitly utilizing the best, standards-based features of Web Services to create the infrastructure for future Grid environments.

Infrastructure and middleware — lessons from the front line

Nick Denning
Chairman and CTO, Strategic Thought

Management introduction

Infrastructure matters more and more. Middleware is a key component of infrastructure. Middleware enables integration, which is why organizations — of all sizes — need infrastructure of some form, whether limited or extensive.

In this analysis, Nick Denning — the Chairman and CTO of Strategic Thought of Wimbledon, London — summarizes the lessons that he and his Company have learned in the course of delivering many infrastructure projects. These have been for governments through large enterprises as well as smaller ones.

Size and capability

Infrastructure and integration are vital to commercial companies that rely on their IT systems for competitive advantage when undertaking their core business activities. In practice there are three types of organizations — the small, the medium and the large — when one considers middleware.

The first type, the small organization, is one which typically has only a single system, plus office products. These organizations do not need and cannot support complex environments. They rely on simplicity from 'out of the box' solutions. Typically these organizations use Microsoft office products plus Microsoft-based solutions from, perhaps, one other supplier to deliver a set of integrated products. These are only modified at their peril.

With one exception, there is little call for middleware or integration. That exception occurs when their systems 'meet' the Internet. This is, however, a vital first point of application integration, for increasingly much business occurs through the Internet medium. Thus, even small organizations cannot wholly escape integration, middleware and infrastructure — for the Internet is now everybody's infrastructure.

The second — medium — type of organization has multiple platforms and multiple systems, some of which are specialist niche applications specific to their business. Despite this, this size of organization frequently does not have, or want to employ, the capability to develop large and complex IT architectures. Rather it prefers to rely on industry standards to deliver integrated solutions.

This size of organization also uses Microsoft products, particularly for its office, email and desktop solutions. It then tends to depend on a single application supplier — or small group of suppliers — for integration of the various elements of their business.

In practice these organizations are not usually particularly sophisticated IT buyers. They seldom achieve competitive advantage from their IT systems — as everyone else of the same size tends to have much the same investment or capability range. They rely on taking standard, inexpensive, technology and then using it efficiently.

Effective users of IT in this group have sufficient knowledge to understand the limits of their capability, but poor ones regularly exceed their capabilities and suffer problems. These types of organizations are classic targets for integrated solutions — such as SAP or Oracle or PeopeSoft supply — that can satisfy most of their business needs.

The third group includes the sophisticated buyers of IT. These often are large organizations with large IT departments and many systems that need to be integrated. In practice, the sheer scale of the integration challenges in these organizations is too large for most suppliers to be able to understand and deliver. This means that these large organizations have had to establish large teams of people who are employed to design and deliver infrastructure and integration and well as to mandate selected standards to suppliers.

The key point here is that one size does not fit all. The requirements of each size of organization are as different as are the capabilities to manage, operate and hence benefit from any IT investment. If a given organization is to benefit from IT, it must review carefully its need and then its ability to manage and deliver IT systems. If its capabilities are limited, then it should either wait or work with suppliers who have developed and can deploy a generic system that meets sufficient of the organization's requirements, without further tailoring.

Upgrading or replacing

Many organizations seem to believe that it is cheaper to throw out an old system than to upgrade it. Regretfully this is not the case in most situations. The assessment — that replacement is better than upgrading — often reflects a lack of understanding of the complexities associated with current systems and their ability to support existing business processes.

Experience shows that it is rarely superior to try to replace existing systems, and certainly not in a single project, unless management is prepared to adopt precisely the new business processes supported by the replacement. Sweating existing software assets almost always works better than starting anew. Building interfaces to existing systems, and leveraging their capabilities for as long as is practical, has measurable benefits.

There may be business reasons for change, where the data model and supported processes and functionality do not support the business. There may be technical justifications for change, for example to support a new GUI or because products become unsupported or to reduce infrastructure and support costs of obsolete equipment or lack of skills in the legacy software systems. The use of middleware provides new applications access to existing applications by building adapters that message-enable both.

In effect, one should migrate functionality only when that functionality needs to change anyway. There may be com-

pelling cost justifications for moving off a platform. Nevertheless, understanding and then planning is required so that the customer organization — rather than any supplier — controls the agenda.

This is vital when considering the deployment of infrastructure that defines a time line for the availability of interfaces. Existing applications can be extended by building adapters. Systems in development can have enhanced support interaction added and systems being specified should plan to be compliant with standards that will develop within the life of those systems.

Initial middleware deployments

When embarking on the first middleware deployment choose the shortest, simplest project to get middleware into production with minimum risk. Above all implement a project which involves the minimum management involvement across departments and functional areas. Never implement an inter-organization project first.

The objective of placing the middleware into production is that the operational team skills to manage middleware can be consolidated. This should always be achieved before taking on phases with greater levels of difficulty.

At the same time it should be borne in mind that no single project can ever justify the expense of infrastructure (including middleware). There has, therefore, to be a corporate commitment to development of infrastructure, with the return on investment scheduled to occur over a number of (not one) successful projects.

The infrastructure architects must, therefore, select an implementation approach which can deliver corporate benefits. A phased approach should implement a complete end to end business process, which produces specific business benefits, rather than address problems in one functional area that only 'shifts the road block onto the next junction'.

Evolutionary stages

There are several stages of infrastructure evolution (Figure 3.1). Each organization must progress through these various stages and for some there is a certain minimum time that is required that cannot be hurried (normally it is not possible to have a baby in one month by getting nine women pregnant).

Infrastructure affects every program of work within an organization. While it is possible for everyone to obtain a high level overview of the goals associated with the delivery and adoption of infrastructure, it is seldom possible for individual project teams to understand (or accept) the impact of adopting middleware unless they go through this evolutionary process, which can be summarized as:

- **start with point to point connections between individual systems**
- **replace point to point connections with a stateless hub**
- **develop processes that cross systems and introduce into the infrastructure state control of each process-step as the process proceeds through the business**
- **automate monitoring, managing and control, initially to identify stalled processes and then to re-start them**
- **extend the output to include delivery of business metrics that can support future investments**
- **develop an infrastructure architecture which will support assembly of processes from components.**

Few organizations have yet reached the last of these. Many are only part way through the first few — even though the technologies to support all six stages are available today.

The corollary to this is that it is often necessary to buy into a technology stack that is able to support subsequent evolution of an organization, particularly if integration with other organizations is anticipated. No organization should try to deploy all the technology components in a single project. The reason is simple. Such a project will almost certainly become too complex. It is also generally useful to lock in prices for the additional components as and when they are required in the future.

Top down architecture and bottom up implementation

An approach which combines a top down architecture with a bottom up implementation is vital. Excessive architecture will result in too much blue sky and insufficient delivery. Too great a focus on short term implementation will result in wrong choices that will need re-engineering later.

Phased delivery, based on a staged evolution, is essential to meet the most pressing requirements quickly. Once a phase is initiated it must be run to completion without making a change to requirements. Once a phase is complete and the capability has been delivered, then there is an opportunity to:

- **take stock**
- **adjust requirements**
- **re-assess priorities**
- **undertake any re-scoping**

before starting the next phase. Thus a critical aspect of an overall design is the 'right sizing of phases' to ensure each deliverable:

- **is big enough to be useful**
- **can also be delivered quickly enough for the feedback loop to work effectively.**

Testing focus

There are many separate software components in a modern middleware infrastructure. If each component has a new release every 12 months, this represents a significant rate of change. Big changes never happen because of the huge risks involved.

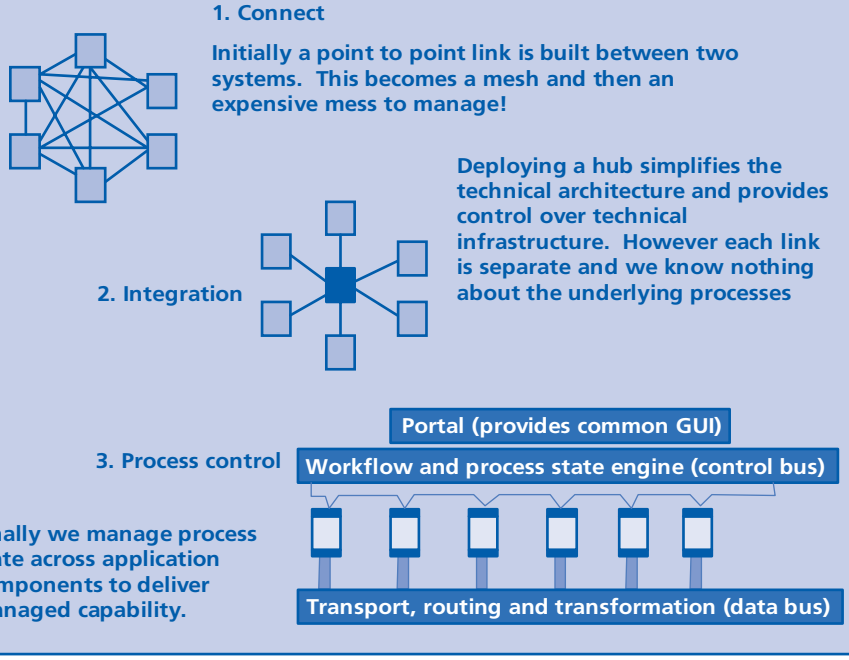
At Strategic Thought we recommend that our clients assume that a release of some component of software will happen at least every month and that they should plan for an automated regression test approach to 'prove software correct' — rather than a big bang approach. This means that, inherent in any overall infrastructure design, is the need for an interoperability which will transcend multiple versions of software being in production as a consequence of the progressive update of different systems on different timescales.

Similarly, accept that it is impossible to achieve the complete rollout of a large system in a single attempt within a large organization. This should not be attempted. Any approach that depends on all participants in a process running on the same software release is doomed.

Indeed, the biggest investment to deliver working infrastructure is found in the requirement to develop and deliver automated testing regimes to demonstrate functional correctness, performance and scalability. Almost as important, and expensive, is the investment required in the design and implementation of middleware infrastructure so that it will permit a progressive and managed deployment of software into a production environment without requiring system shutdowns.

Equally, it is vital to avoid a 'one bite of the cherry' approach. Achieving sign-off quickly for a limited set of requirements is important. The client who is signing off, therefore, has to be confident that any requirements that are left out now can, or will, be delivered subsequently if a business case can be made to address them. The 'one bite of the cherry' approach has been responsible for too much scope creep on too many projects.

Figure 3.1: Evolutionary stages



Project managing middleware infrastructure creation

Project management of middleware implementation is critical. Actually middleware and infrastructure technology is straight forward — provided you do not try to adopt too much of it all at once. Nevertheless, do not forget that the business problem that has to be solved is probably complex if middleware is needed in the first place.

This again makes it important to:

- **break problems down into manageable elements**
- **align these to an overall architecture**
- **adjust the delivery program to phase deliverables, each of which should deliver benefit(s).**

It is often the case that the initial rollout of a middleware infrastructure takes time. Often there is no apparent activity perceptible outside the development and deployment team. It is, therefore, essential to:

- **ensure proper project planning**
- **establish predefined milestones**
- **demonstrate that those milestones are met**
- **set and meet measurable RoI targets.**

Only through such discipline can an initial middleware project provide positive feedback to the rest of the organization that progress is being made. Such feedback is essential to win, and keep, the hearts and minds of those funding the program.

Risk management

Risk management has a significant role to play in effective project management. A critical change of culture for most organizations is the adoption of business procedures that are based on a risk management approach. While it is not a crime to be affected by risk, it is a serious issue to fail to identify, assess and communicate risk.

At Strategic Thought we are already seeing an increased focus on the management of risk as opposed to its pure identification. Organizations are learning (often the hard way) that risk management does not equate to risk administration. Nor is reporting it, by ritual, sufficient.

Risk management is critical to exploiting the 80:20 rule. Risk mitigation is about establishing the likelihood of a risk occurring, the possible impacts when it does occur and the capability to deal with these impacts when required — your organization's 'appetite' for risk. This can be set at organizational, personal or project levels and must be communicated to ensure 'adherence to the rules of the game'.

Another aspect of the management of risk impacts is assessing the degree to which any design includes specific mitigation features. Managing the impacts of risk and mitigation of them might involve:

- **identifying, in a controlled fashion, the scope of future requirements**
- **working out which design features, if incorporated, would enable future development**
- **accommodating certain of these features in a design — if this can be undertaken without unnecessary complication or compromise of that design — to meet the specified requirements of the first deliverable.**

The objective is to design with infrastructure evolution in mind so that one can:

- **minimize the impact on the current design of future evolution**
- **avoid over complication.**

In this context, there is significant danger when requirements are too vague, when they are changed regularly and when the impact on other requirements, design activities and risks cannot be determined because there is no cross reference mechanism between risks, activities and requirements. This highlights the danger in designing multi-purpose products:

- **the risk matrix becomes too complex**
- **the danger of over design is high**
- **some requirements may be fundamentally unreconcilable.**

A compromise between competing requirements damages fundamentally the solution. A 'jack of all trades' is the result. In many situations a monetary value can be assigned to a risk. This is what insurance is about. Risk management is about finding the right balance between time, cost and performance/quality — with the change in one impacting in some way the other two factors. Many may think that risk management cannot be applied to middleware and infrastructure. This is incorrect, as our own experiences have shown.

However this does not tell the whole story. In specific industries the situation is more complex. In the aircraft industry the holy trinity are heat, weight and fuel consumption: these are the most important factors to monitor and

keep in balance. However, more recently, the impact of some endeavors is strongly influenced by health and safety considerations as well as the potential impact on an organization's reputation (as the UK's rail disasters or NASA's loss of space shuttles has demonstrated).

Hubs and disintermediation

Use hubs to achieve disintermediation. This will require mapping of transports, protocols and message formats which often have an 'impedance mismatch' and require additional data to support the mapping.

However, remember a hub can introduce a single point of failure. Therefore specific design measures are required to distribute hub capabilities and to provide high availability (HA) and fail over. HA is not difficult to implement in an initial design and is generally a matter of cost. That said, it is usually difficult — if not impossible — to reverse engineer HA into a mature system.

High availability architectures readily exploit cluster technology where:

- **disks are mirrored locally to protect against disk failure and held in a storage system accessible by multiple computers (though only accessible by one computer at any time)**
- **two machines operate in main/standby mode; when the main equipment fails, cluster software automatically starts the standby machine which mounts the disks that have now become available and so services can restart**
- **disk contents can reliably be replicated across the network to another site where another computer is participating in the cluster and available to take over in the event of a failure of a complete site.**

This HA type of approach has historically been expensive and bulky but is now available on standard Intel hardware using Linux. The ability to replicate I/Os across a network from small or low cost equipment is an issue that needs to be investigated.

An alternative approach is to replicate data messages at the application level rather than data at the disk level so that there is always a separate copy of a message in an alternative site. The challenge then is to determine:

- **how to purge the copy message when the original has been consumed**
- **the process by which applications fail over from the primary message source to the secondary message source without missing messages or processing a message twice.**

The key is deciding how to choose between alternative approaches using the identified quality of service requirements identified. Different message protocols exist between applications to protect against failures in underlying transport mechanisms.

Quality of service

It is also important to remember that HA is just one extreme of the more general design points that relate to quality of service within an infrastructure. Different transports deliver different quality of service as well as protocol implementations, messaging and interface APIs.

When connecting these together it is often not possible to provide an interface that delivers a consistent interface and quality of service. It is, therefore, necessary for layers either above or below the current layer of a given middleware stack to provide the necessary capabilities.

There is, thus, a requirement for each layer of the interface to be a three layer sandwich. The middle layer is the defined functional interface. The components on either side of that layer provide the necessary quality of service but can float between above or below the interface. For instance if a transport layer can provide guaranteed delivery, there is no need for the calling layer to provide that. For guaranteed delivery comes at a performance cost.

However, if the calling layer also needs guaranteed order as well as delivery, and order is not part of the lower layer, then guaranteed order must be provided in the higher layer, and therefore potentially guaranteed delivery as well. If guaranteed delivery duplicates messages, its use may be avoided in the lower layer. In this case, in the lower layer one could use non-assured, fast delivery because the required quality of service is already provided.

Remember also that assured messaging between applications — using persistent messaging — should mean that messages cannot be lost. These can, however, be 'mislaid'. Normally the term 'assured delivery' is used to differentiate from guaranteed delivery. Messages can be lost in an environment where there is only one copy of the message and if there is total destruction of a site. Messages can also be unavailable in a scenario where a machine containing the message has failed, until that machine can be brought back on line.

Commonality and responsibility

The subject of infrastructure is so large that it often seems impossible for all participants to engage productively in every discussion. It is, therefore, vital for not only run-time standards but as importantly for design time consistency that the definition of common technical terms occurs. For example, each stack must make assumptions about underlying stacks. Only the dedicated architects should participate in all discussions, and it is the architects who should have the specific responsibility for ensuring consistency across discussion boundaries.

That said, clear lines of responsibility and terms of reference are required for these architects. These should report to a 'chief architect' who is responsible for the organization's overall implementation.

Buy rather than build

It is almost always quicker and cheaper to buy infrastructure products than to build your own middleware. Most of the problems that most organizations encounter have already been solved elsewhere. Furthermore, leading software products are, typically, far better tested than bespoke developed software tested against the limited numbers of test scenarios that can be specified and generated by an internal test team.

The level of quality required for a product to be sold in the market place is an order of magnitude greater than is generally achieved for bespoke systems. In addition, there is the significant management cost in maintaining one's own developments. Finally, the cost of porting between operating systems is a one-off cost shared by all customers of a given product. All these point to buying not building ...

Java and commodity middleware products

Standards are vital at all levels of the application stack. Java is now a serious contender as a standard. Today few applications now place such a huge load on CPUs that would rule out Java on performance grounds (when compared with C/C++). The performance of most applications is dominated by I/O wait at the network or disk. Improved programming construction and efficiency when using Java has overcome many of its previous potential drawbacks.

The key benefit of Java is obtained when a single application is to be run on multiple platforms. The 'compile once, deploy many times' characteristic of Java dominates the argument when use is to be made of common code across multiple platforms.

At the same time, organizations should look carefully at the commodity product providers. Technology products should not be used that are specific to a particular sector unless there are compelling reasons. The benefits from commodity software arise if every organization is using infrastructure software that intercommunicates effectively and efficiently with all other infrastructure software.

The scale of this task means that there are really only 2.5 games in town: Microsoft, IBM and 'open source'. The cost of maintaining a complete infrastructure offering and of testing and maintaining interoperability at the appropriate level of quality is so large that it is beyond the scope of all but the largest companies, who then leverage that investment to dominate the market and minimize the cost of infrastructure. Industry is littered with infrastructure companies that have not been able to keep up and have effectively been taken out of the game.

Of recent note is that Mercator, one of the leading hub technologies, has now been purchased by Ascential. So, though the likes of Vitria, webMethods and SeeBeyond have performed well, it is the opinion of this author that these are destined to be value added providers delivering business process specific logic which uses standard infrastructure tooling. Within the business sector we, at Strategic Thought, expect to see IBM as dominant, while Microsoft will own the commodity sector.

Architecture

It is vital that an overall infrastructure program is managed from a technical perspective by a single architect responsible for overall delivery, as the arbiter between conflicting interests and with the authority to direct individual teams where required. However it is impossible for any one person in a program to have a complete view of, and manage in detail, all aspects of a program.

For this reason it is vital that interfaces are defined in terms of a vertical stack — as is the case, for example, in the ISO 7-layer model — as well as take into account quality of service where specific aspects of functionality can be implemented on either side of the interface if particular implementations of an interface cannot provide the full range of functionality required. In effect each interface must have within it three layers.

It is the architect's primary role to ensure that the capability required can be developed and delivered, that components which comply with the stated interfaces are available and to introduce time, testing and change management into the program. This requires a definition of:

- **new functions and the definition of new or extended interfaces to deliver enhanced functions**
- **the timeframes in which interfaces shall be evolved**
- **interdependencies, in the timeline, to support the requirements for evolution**
- **the mechanisms that are needed to ensure that components remain loosely coupled**
- **dependencies between components, and these need to be limited to defined interfaces whenever possible (so that systems are robust to swapping in and out of various components)**
- **standards, with which all components must comply (so that they can integrate into the existing infrastructure environment and communicate with the deployment, process management and operational management mechanisms)**
- **testing processes to optimize the development and delivery of new or enhanced capabilities.**

Careful architectural design is vital to 'right size' functions if these are to present a balance between complexity and simplicity. An architecture must seek to ensure that the impact of any change on one component is restricted to its immediately neighboring components, rather than affect a whole system. This is an essential top down approach that must be undertaken within any organization.

Futures: a personal view

As I describe, I believe:

- **IBM infrastructure will dominate in large businesses**
- **Microsoft will be widely used to deliver end user applications deployed to the public or to small organizations**
- **integration between the two will be based on secure Web Services — and will be common.**

Middleware supporting application-to-application integration is largely implemented by programmers. Collaborations between people will be based on email-oriented work flows and document attachments that do not require programming effort.

The next major initiative I envision is the development of intuitive tools that will enable business users to construct and deploy work flows that integrate collaboration style content, and Microsoft Outlook tasks, with application-based work flows. The key will be avoiding the need to write and test code. I anticipate that this will require further development of GUI based work flow definition tools together with the availability of metadata from WSDL repositories. These will enable work flow applications to interact with Web Services, by validating their work flows against the metadata in the repositories. In this view:

- **programmers will develop the Web Services**
- **users will build on this capability by developing processes to invoke these services and then engineering processes together to develop collaborative capability.**

Finally I believe that 'integration at the glass' — using portal technology — has the capability to mitigate the pressure for technical change by hiding 'old fashioned' and inconsistent interfaces behind a common browser GUI. However a better business case for portals, and a greater supply of portal-enabled components, will be required before widespread portal deployments occur.

Management conclusion

No two organizations are identical, though there are similarities between organizations in the same sector, between organizations with similarity of process and between organizations of similar financial models or size. As Mr Denning describes, one size does not fit all and each organization has differing requirements and abilities to manage its resources. It is therefore critical that systems of differing capabilities, functions, interfaces, deployed on a range of equipment can be constructed from common components which can interoperate. There is also a huge historic investment in the technology available to any organization seeking to develop new capability. It should be the responsibility of any organization to seek to leverage that investment to bring capability to market that optimizes the balance between cost, quality, capability, time, risk and reputation. This approach mandates the use of existing products that comply with standards yet can be extended to meet specific requirements of the procuring organization.

Any infrastructure is a key enabler in all future procurement programs. At the same time, one has to seek to deliver the minimum amount of network glue as quickly as possible at the appropriate points to enable the needed function as well as to allow:

- ***experience to be developed***
- ***strategies to be modified in the light of this experience.***

Remarks on managing distributed systems and middleware

Peter Bye,
Consultant, Unisys Systems & Technology

Management introduction

Ever since the beginning of computing, systems have needed management. These early systems were based on mainframes, with networks of dumb terminals, and were independent of each other or had only a very limited amount of interaction. In spite of software developments in operating systems, database management, transaction processing monitors and packet switched networking, the bulk of applications largely remained in vertical 'silos' of automation.

In many ways, the management problem was (then) relatively simple, in that the owner of the system managed everything — including the network. Only a handful of systems contained any real software.

However, although the environments were simple by today's standards, the limited tools available at the time meant that systems management was extremely labor-intensive, often involving a great deal of console interaction, mounting tapes and so on. Some specific mainframe-based applications were developed to help, for example, fault tracking applications. But much relied on the vigilance of human operators and other management staff.

Starting in the early 1980s, the micro-processor revolution plus developments in telecommunications plus the Internet explosion and new software developments, have all combined to change the face of the typical IT environment. Today, there are thousands or even tens of thousands of devices with software in them, with applications distributed over many systems. In an increasing number of cases, different parts of the overall environment may be owned by different organizations. In the face of this complexity, effective systems management becomes more and more critical to the success of an organization's IT capability.

In this analysis, Peter Bye offers thoughts on the problem of managing distributed environments — and its related middleware — and discusses some of the principles his experience suggests need to be followed when enhancing systems management environments.

Systems management functions

When I talk about 'enhancing', I use this word deliberately. Few organizations start from scratch. Just as new IT systems have to take account of, and be integrated with, existing systems, so new systems management capabilities have to be introduced into an existing management infrastructure. But before exploring the problems of managing distributed systems, I would like to propose a classification of the various functional components of systems management in order to establish a framework for discussion.

For the purposes of this analysis, I have divided systems management functions into four categories:

- **administration**
- **operation**
- **fault tracking and resolution**
- **performance management and accounting.**

Security is also a systems management function. However, it is a big subject in its own right and pervades every aspect of any environment. It merits, therefore, treatment in its own right but this is beyond the scope of this particular analysis.

Administration

Administration is concerned with all aspects of managing the configuration of a system, and therefore comprises those functions associated with changing it in any way. It includes managing the addition, relocation or removal of hardware components, together with any associated software and database changes, software installation, update or removal.

For example, communications circuits and workstations are typically described in databases of some form in systems associated with them. The addition of new circuits and workstations requires the relevant databases to be updated.

Operation

Operation is concerned with keeping the system running and doing the work it is intended to do, a simple statement to make but one which encompasses a vast range of activity. Operation divides into two parts:

- **one is to do with the normal processes of managing the work flowing through the system and associated functions, such as the backup of databases**
- **the second is concerned with detecting faults or other exception conditions, and taking action to correct them or at least minimize their impact on the system's capacity to do its work.**

The first set of functions includes running sequences of batch jobs, varying the allocation of production capacity to requirements, for example reducing real-time transaction capacity at night to allow for batch processing. The second set of functions is concerned with unexpected events. For example a group of users might find it can no longer access an application. Immediate action will be required to determine where the failure has occurred, and to try either to correct it or to find a way of bypassing it.

Fault tracking and resolution

Fault tracking and resolution is the third category. Information about faults not immediately resolved needs to be recorded and used to trigger subsequent activities to fix them.

The information about a fault includes any technical details captured at the time the fault was detected, as well as guidance on its severity and the urgency of resolution. This information can be updated as work is done to resolve the fault, and warnings can be raised if an urgent fault is not being resolved.

Performance management and accounting

The final category is performance management and accounting. Statistics gathered at a variety of points provides the raw data for this activity, or rather set of activities.

Performance information is required for technical reasons, for example to allow a system's performance to be optimized by tuning it in various ways, as well as to predict future capacity requirements, so that action required to expand capacity can be taken in due time. It is also needed for commercial reasons, for example to generate billing information, to determine whether service level commitments are being met and whether service providers are meeting their commitments.

Organizational implications

In any organization, other than the smallest, different groups will perform and be responsible for different functions and will — or certainly should have — well-defined relationships with each other. Depending on the size of the organization, each group may have a number of teams to handle the different sets of functions.

Figure 4.1 shows the relationships among the various groups. Typical interactions could include:

- **administration requesting operations to implement configuration changes**
- **operations recording unresolved problems with fault tracking and resolution for further analysis; administration may then be invoked to implement changes, for example patches, to fix faults**
- **performance management may identify bottlenecks requiring configuration upgrades and will therefore work with administration.**

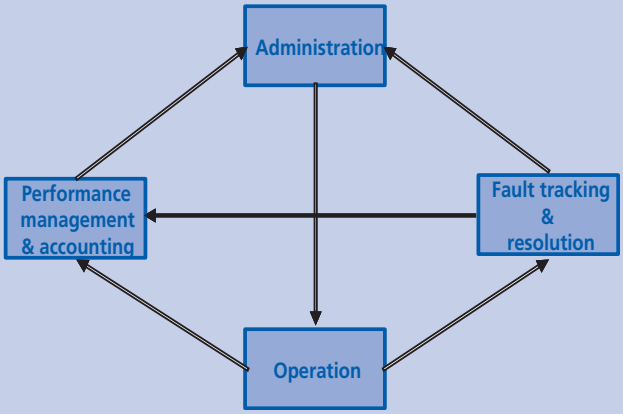
Any large organization — such as a bank, an airline or a government department — will require significant systems management resources. These resources do not consist of 'just people'. Technology is essential, not only to manage costs but, more importantly, to ensure the highest levels of performance and availability.

Large-scale environments cannot rely too much on manual processes for group to group and people to system interactions — because they are error prone. Integrated technology, involving high levels of automation, is essential for successful systems management — with human interactions reserved for the kind of complex decisions best made by people. This is true for relatively self-contained systems. Distributed environments compound systems management problems even further (an area where it can be argued that middleware complicates rather than simplifies) — making suitable automation even more critical.

Distributed systems management: the issues

Figure 4.2 represents a typical IT environment within a medium to large organization.

Figure 4.1: Relationship between systems management groups



It shows a number of application servers, typically containing a mixture of applications of various ages. Some will be mainframes, running applications that may have been in use for some years, while others will contain applications running under various flavors of UNIX, Windows and Linux.

The servers run in a data center environment and are physically interconnected using a LAN. The applications provide services to a variety of users. They do this either individually or in collaboration with each other. Assume (for simplicity's sake) that they have all been kept up to date and are able, at least in part, to communicate with each other and with external systems using distributed middleware of some kind.

External users are connected to the application servers by various networks. The first (shown as Internal networks in Figure 4.2) comprises a mixture of IP routers and switches, with some legacy network technology. User equipment includes workstations, browsers and possibly some remaining terminals. It may also include some connections to other organizations, using legacy protocols.

The second is an outsourced IP network, in this instance managed by a network provider that:

- **assumes responsibility for the network up to the connection into the data center**
- **agrees to meet defined service levels.**

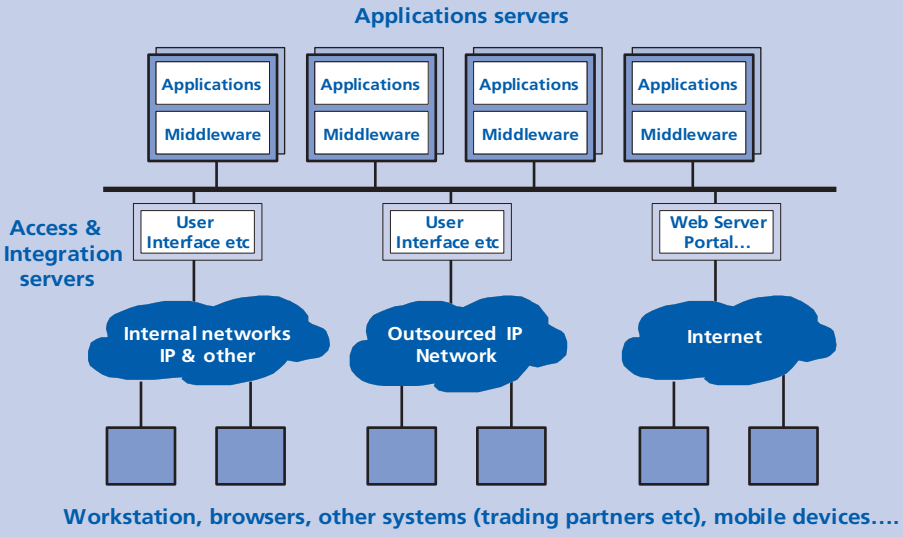
End user devices would typically be workstations and browsers, and perhaps some application systems.

Finally, the Internet supports direct customer access from browsers and other devices, such as mobile telephones. Assume also that there are Web Service connections to other systems operated by business partners. The networks are connected to the application servers, via an internal LAN, using what are described as 'Access and Integration' servers in Figure 4.2. These servers could be as simple as routers or communications processors, or much more sophisticated systems — including Web Servers or Portals.

To make this clearer still, suppose the organization concerned is a bank. There could be different applications to handle products such as current accounts, savings accounts, mortgages, investments and customer management.

Services would be delivered either by a single application or some co-operation among the applications. For example, a service to request the balance in an account would only require access to one system. A service to provide the status of all products owned by a customer would require access to a customer management application to find out what these products are, followed by requests to each of the product systems to obtain the current status.

Figure 4.2: Schematic of medium to large organization



Internal networks might connect the bank's main offices and call centers to applications. The outsourced IP network could connect all the branches to the data center, while the Internet would allow clients direct access to services and also provide connections to external service partners — for example companies providing products not directly supplied by the bank (perhaps insurance products or credit verification).

Complexity

There are several significant attributes associated with environments of this complexity. In turn, as will be seen, each can have a significant effect on systems management.

The overall environment is complex, containing many different components — hardware, software and applications. The number of systems containing software of some form — servers, PCs and so on — may well run into many thousands. Each will need some form of management on its own, for administration, operation and other functions. One issue is: how does an organization even find, never mind keep track of, everything?

The large number of components also means that there are potentially huge numbers of different conditions being reported. This raises questions like:

- **how is it possible to distinguish between the important and the relatively trivial?**
- **how can one avoid being overwhelmed by repeated instances of the same problem?**

Furthermore, it should be borne in mind that these components do not exist in isolation; they are variously involved in end to end services:

- **if one component fails in some way, which services will it affect?**
- **if one is changed, which others are or might be affected?**
- **if a service has to be changed, which other components will be involved?**

In the banking example described above, an important part of the environment is outsourced. The IP network is managed by the network provider. But its behavior clearly affects the overall service provided. A teller in a branch may call a help desk, saying that the current account system cannot be reached:

- **does the source of the problem lie in the IP network, the data center, or somewhere else?**
- **how much information is needed from this network to determine the source of the problem?**

Some services will depend on accessing external application service providers, for example as Web Services. These application services are rather more than outsourced. They are part of some other organization's system, over which you or I have no direct control. How can problems in performance and availability be tracked, and resolved?

It should be clear, therefore, that effective management of such environments requires sophisticated management tools. And, as might be expected, systems management technology has not stood still in the face of challenges.

The technology

The arrival of micro-systems in the 1980s, and their phenomenal improvements in price/performance since then, has enabled the development of a wide variety of systems management products. The same micro-system technology has also been widely used to add intelligence to a variety of storage and other subsystems, not only making them far more functional but much easier to manage as well. Storage subsystems, for example, have been revolutionized, apart from the other developments in the technology of the media — storage density and so on.

That said, there is a daunting range of products available on the market, covering all aspects of systems management. These can be divided broadly into two groups:

- **point products, which perform a specific aspect of management, sometimes confined to a single platform or group of platforms**
- **suites, which provide a more or less complete set of systems management capabilities.**

The two are not mutually exclusive. Suites may enable specific 'best of breed' point products to be integrated with them, for example fault tracking and resolution applications. And point products often possess external interfaces, enabling them to feed other products and/or to receive feeds from them.

Some organizations have constructed effective management environments by integrating point products. Others have adopted suites as the basis of management, perhaps supplemented by one or two point products integrated into the suite.

There is no clear best approach. Success depends more on the starting point and, particularly, on the implementation.

I have already discussed the importance of automation in systems management. It can be applied from external management systems.

For example, Unisys has a product called Single Point Operations (SPO) which can respond automatically to system events or questions. Another example is batch job scheduling across multiple platforms. The whole work flow is set up and can then be managed by a suitable product, without needing operators to start and stop things.

Many management functions can be, and have been, automated in the same way. The increased sophistication of products such as disk and tape library subsystems greatly helps reduce manual intervention as well.

Automation from within

But automation can also be applied within systems. In most cases, automation in the form of self-correction has been available for a long time, provided by the operating system and other software. Some operating systems are very good at it.

But current thinking goes beyond this. There is much interest in self-management within systems, referred to as autonomic management. The aim is to allow systems to be self-healing, detecting and correcting problems, reporting the events for the record and perhaps requesting repairs or other corrections later. Exactly how far this can extend is still open to question.

Self-management is not confined to error recovery. Administration alone can be a major burden. It is therefore highly desirable to have systems generate the configuration they need, and otherwise adapt themselves, when connected into the environment.

A simple example is DHCP, which allows devices (typically PCs) to acquire an IP address when connected to a TCP/IP network. In the large environments around today, and with people moving around different locations with a laptop, the problems associated with having to configure IP addresses manually would be overwhelming.

Principles for managing distributed environments

In the light of all the challenges referred to above, how should an environment such as that shown in Figure 4.2 be managed? As noted at the beginning of this analysis, most system environments will have been created or evolved over many years. This equally applies to the systems management infrastructure and its supporting organization.

The primary concern should with enhancing what is already there, not starting from scratch. Ensuring continued effective systems management is a never-ending process. Applying new technology or making better use of technology already in place can always improve it.

What is possible is likely to vary widely, depending on the technology already installed and the processes followed by the systems management organization. There are, however, a number of guiding principles, which can be applied to help move in the right direction.

Principle number 1: scope

The first concerns the scope of the different management functions. In this context, scope means the amount of the environment over which the management functions apply. Clearly, all management functions — administration, operations and so on — could apply over the whole environment, controlled from a centralized location. Given the complexity involved, and the fact that parts of the environment shown in Figure 4.2 are not owned by the organization itself, this is usually not possible or practical.

It is possible, however, to establish a workable approach by dividing management into those functions that:

- **change an environment in some way — either administratively or through operational commands**
- **do not change it, but collect information about status and behavior.**

To understand what is happening, and to facilitate any corrective action, it is important that a view of the status of the whole environment be maintained. In the environment shown in Figure 4.2, for example, the outsourced IP network is a critical element to overall performance of the system; it could be the bank's branch network. It is, therefore, essential that application systems operations people know what is happening in the network and are able to extract additional information from it, which can then be correlated with the operational status of the applications systems.

Such information enables corrective action to be taken proactively, ideally before any loss of service occurs. It also allows help-desks to respond to questions from remote

users, who may for some reason not be able to access key applications.

The widespread monitoring principle can be extended to cover what we may think of as horizontal layers across the environment, for example middleware. The applications servers shown in Figure 4.2 contain middleware (which is used for various kinds of co-operation among those applications) via message queuing, procedure calls of various kinds and perhaps some Web Services technology, such as SOAP.

This collection of middleware is a distributed infrastructure, which has to function in its entirety. It requires careful monitoring, as failure or other malfunction in one system could cascade through the whole environment. It is, therefore, just as important to be able to see the status of this whole layer of software, to ensure that connected applications continue to function and deliver the required services.

Furthermore the middleware environment must be started (and stopped) in a controlled manner, ensuring that all the pieces come in production in the right order. In some ways, this problem is analogous to managing batch applications across many different systems. In fact, batch job schedulers have been used with some success to manage the start-up of middleware spread across multiple systems.

On the other hand, administrative and any other changes to the environment should be carefully constrained, with the whole environment divided into autonomous domains of some sort. For example, the outsourced IP network should already have a management group, able to make changes; no other group should be allowed to do this. But that does not stop others needing to have visibility about its status, both for immediate operational reasons and for performance monitoring (for example to decide if service level agreements are being met or not).

Similarly, it is not possible to change the Internet service provider's part of the environment, nor any of the remote connections such as partners providing services. But it is desirable to be able to know their current status. The principle of restricted change capability but wide status visibility could also be used within the organization itself, by dividing it into autonomous domains.

Principle number 2: automation

Automation is a second principle. As I have noted more than once, a high level of automation is essential to reduce costs and improve overall quality. Automation permits the filtering out of what matters from what is of minor importance. It also acts to minimize manual intervention, which many studies have shown to be highly prone to errors.

In ever more extensive and complex distributed environments, the scope for error increases. In a high proportion of cases, the manual actions performed — simple responses to console questions, for example — can be automated. The goal should be to leave human decisions to what is handled best by people, and cannot be automated, or require human-level policy decisions to be made.

Principle number 3: end to end service provision

My third principle concerns end to end service provision. The environment comprises a large number of different components. Each of these will report status and any status changes, and be subject to administrative and other externally applied changes.

However, these components do not exist in isolation; collectively, they deliver services to users. It is, therefore, essential to:

- **know the impact on services of any component failure**
- **identify the potential impact on services if any component is changed.**

There are various 'things' we can do to gain a better picture of end to end service status. One simple approach is to execute test transactions from a number of places into critical applications and to measure the results, which can then be used for performance analysis. There are also products on the market that allow components in an environment to be related to the services that depend on them.

Principle number 4: applications can contribute to systems management

A fourth principle is that applications can contribute to systems management. One way is to ensure that applications are able to log what is happening.

Take performance monitoring, for example. To obtain a complete picture of performance, it is essential to know the performance of every step in a message flow. All the message performance measurements could be put into the message, and passed around with it. Alternatively a unique message identifier could be inserted in each message, so that performance logs from different servers can be pulled together to trace the performance of individual messages.

Another example is fault detection. If there is an error detected somewhere, knowing who was the end user — and what was the input message when the error happened (and maybe was the cause) — is valuable information. Applications can be provided with libraries of common services, allowing them to log and report events easily. These services could also extract additional information to supplement that provided by the application.

Principle number 5: evolution

The final principle concerns the way to go about enhancing a systems management environment. I support the adoption of an evolutionary approach. New technology can be applied with great benefit, but it should be introduced incrementally, as it is not possible to anticipate in advance the effects of its introduction.

What is currently installed is important, as interesting technologies may not be easily applicable; for example, there are no agents available on some systems. Organizational procedures and processes are also part of the in-place infrastructure, requiring change if new technology is introduced. Much better results can be achieved more quickly and with lower risk of disruption through a series of small sets of enhancements, which are reviewed regularly by systems management teams before moving onto the next set.

Management conclusion

Systems management is critical infrastructure in today's complex, distributed environments with all their middleware. Just as these environments will in most cases have grown up over a long period, so the systems management infrastructure will have similarly developed.

This applies not just to the technology and tools used, but also to the organization using them. The normal requirement is, therefore, to enhance what is in place rather than build a new management environment. Improvements can always be made, and much new technology and standards are available or are in development. But they do have to be applied carefully.

Although each environment is different, presenting different challenges, there are guiding principles that can be applied when enhancing the systems management environment:

- ***pay careful attention to the scope of the different functions; monitoring should be widespread, while the capability to make administrative and other changes needs to be carefully constrained***
- ***seek to automate as much as possible, for cost reasons and, more importantly, to improve the quality of operations***
- ***take an end to end view: there are many components involved in delivering services, and understanding the effect of any component failure on end to end services is vital***
- ***do not forget or overlook applications, for these are as critical as the software infrastructure; the way applications use the environment, for example middleware, has many systems management ramifications (it is essential to know when failures occur, as well as the application taking corrective action)***
- ***finally, follow a controlled, evolutionary approach; this is more conducive to success than implementing large-scale changes to the way systems management works.***

The test of time

Dr Keith Jones
IBM Software Solutions Worldwide

Management introduction

It has often been said that middleware is the mechanism by which one captures the best of IT thought leadership over time. Structured procedures, relational databases, client/server networking, object-oriented languages and message-oriented infrastructures have all been integrated into the middleware products currently available to solve particular classes of problems. Many enterprises have examples of each and every one of these technologies in production today. Many have withstood the test of time.

At a time when executives are focused on reducing costs and maximizing value from IT systems, several middleware thought leaders have been busy developing solutions that address the biggest challenge of all for IT departments — namely the cost-effective integration of business systems using standards-based technologies deployed across heterogeneous systems platforms. Such service-oriented products are now emerging and being deployed in a growing number of successful installations.

In this analysis, Keith Jones positions renewed IT interest in 'event' programming and asks:

- ***if this is yet another alternative middleware technology that addresses an existing class of enterprise problems***
- ***or whether event-driven infrastructures will earn their place in history as being essential to delivery of the kind of competitive responsiveness needed today in an on-demand world.***

Middleware is everywhere

If you accept the definition that middleware is the layer of software above the operating system but below the application system, it is easy to accept that middleware is absolutely everywhere in today's business systems. The primary rationale for having such a layer is that it dramatically simplifies the task of writing business logic by providing reusable library functions that have been tried and tested over time.

Many of today's business systems have evolved over years (if not decades) and comprise a mixture of asset types:

- **modules**
- **procedures**
- **records**
- **objects**
- **schemas**
- **services**
- **flows**
- **transformers**
- **collaborations**
- **and many more.**

The problem is that, while all these assets are used to model real-world business processes, there is often an appreciable gap between the time that:

- **a business event happens**
- **the software model reflects that event.**

One solution to this problem is incrementally to re-engineer critical business systems to shorten that gap in time. But this is really not a viable option for most enterprises. It requires too much.

A better solution might be for middleware to provide a mechanism by which:

- **business events can be extracted at, or near, the time they are first recorded**
- **new business logic acts on critical events as these happen (without the need for any re-engineering).**

It is an on-demand world

The need to identify and close the time gap between real-world events and their software systems counterparts is brought into sharp focus by the highly competitive marketplaces that most enterprises exist in today. Everything, it seems, is now time-sensitive — zero time to market, zero time to fulfillment, zero customer service time — all are goals for the agile, real time, on-demand enterprise.

Fortunately event-driven middleware technology exists and is coming to the fore as a potential solution. How could this apply?

Event-driven systems

Event-driven programming has gained considerable popularity in the software engineering world over the past few years. Systems based on this approach have been implemented over several decades, in particular in a large class of GUI application systems and in a large class of real-time application systems. An equally large class of event-driven business applications may soon be added.

One reason for this rise in popularity is that event-driven architecture and development methodologies facilitate separation of key concerns in application systems. This separation is achieved by almost completely decoupling components in deployed event systems. Another [more recent] reason is that an event-driven architecture offers a mechanism for time-based integration of business systems.

What is an event?

An event is an abstract concept. Each event is characterized by a recognizable state change in an environment and is usually also associated with a source location and a moment in time. In practical terms, the click of a mouse is a very common event in today's computer desktop environments. The passing of a steel ingot through a laser beam is a common event in an industrial steel mill environment.

A business event, by analogy, is characterized by a state change in a real-world business process. The shipment-of-goods, in the fulfillment of an order, is a common event in a retail business environment.

At first this seems very straightforward. Billions of events can be observed each day given the appropriate equipment and the motivation.

However, almost all events (especially business ones) can be decomposed into patterns of finer grained events, or combined into larger grained events, depending upon the focus needed:

- **the mouse click for example may be recognized as a pattern of a mouse-down event, followed by a series of [optional] mouse-move events and a mouse-up event**
- **the steel ingot in motion breaks the laser beam with a beam-off event at its leading edge followed by a beam-on event at its trailing edge**

- **the shipment-of-goods events may be decomposed into goods-picked, goods-on-palette, goods-delivered-to-docking-bay business process events.**

The notion of an event is a powerful concept. Events are a tool that we may use to observe and record what is actually happening in time. But note that, in this world, events have not happened if they have not been observed. Once observed they may or may not be recorded in a digital medium. Many middleware products are emerging to support this concept.

In middleware software and application systems, deciding upon which events to observe and record is a key step to becoming event-driven:

- **finer grained events occur more frequently and may give more information of a particular kind**
- **coarser grained events occur less frequently and may be less costly to use but may not contain all the information needed for a particular purpose.**

What is an event-driven architecture?

Observing and recording events may serve a limited purpose. The real value in event-driven systems comes from the actions taken when events are recognized and processed in IT systems. Figure 5.1 shows the essential components needed to build an Event-Driven Architecture [EDA].

When all of these components are deployed in some form, an EDA can be used to record events and react to them in highly responsive and creative ways. Many middleware products are either already available or emerging to provide the infrastructure needed for such deployment in electronic business scenarios.

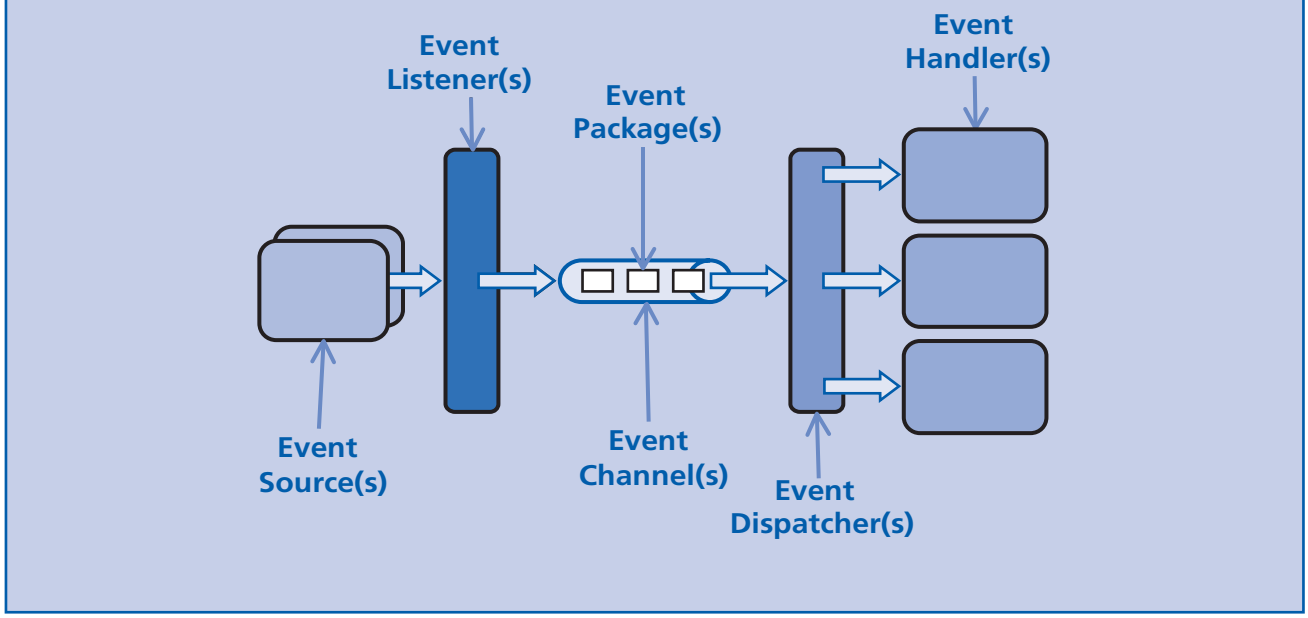
Event sources

A large number of different sources for events can be involved in EDA systems. Some will be sensors in the real world: mouse-clicks, bar-code readers, cell-phones, traffic sensors, smart card and RFID readers for example. Many more will be in the software world: Internet browsers, Web Servers, business logic, data access logic, integration servers, legacy systems and databases, for example. In the future yet more sources might include event handling components as they are developed. The challenge in EDA systems is to:

- **identify correctly those events that should be recorded (and their sources)**
- **then build the 'handlers' to react appropriately.**

In an EDA system, event sources generate events without

Figure 5.1: EDA components



any prior knowledge of which other components will consume them. This is an extremely important principle of EDA, because it removes coupling that might otherwise have existed between the source and processes that handle the events generated.

Equally important is the principle that events generated must carry just enough information correctly to characterize a state change of business significance but absolutely no more. Ultimately the value of events is their use in a wide variety of (possibly unforeseen) applications and surprisingly this can be limited by carrying too much or too little information.

Some event sources actively generate events whilst others passively generate them. Business logic that uses an event generation API at critical points in an execution cycle actively generates events. Such logic does not exist in large volume in existing business systems. However, in future, active sources will be created by:

- **injecting event generation API 'probes' into existing business logic by hand**
- **using development tools that support this concept.**

Passive event sources exist in much larger volume today. These are the many software components in production today that function in customer response, supply chain, enterprise resource and line-of-business systems. They are passive sources because they routinely process transactions and other interactions in large volume that could otherwise generate events but do not because they lack an embedded event 'emitter' mechanism (Figure 5.2).

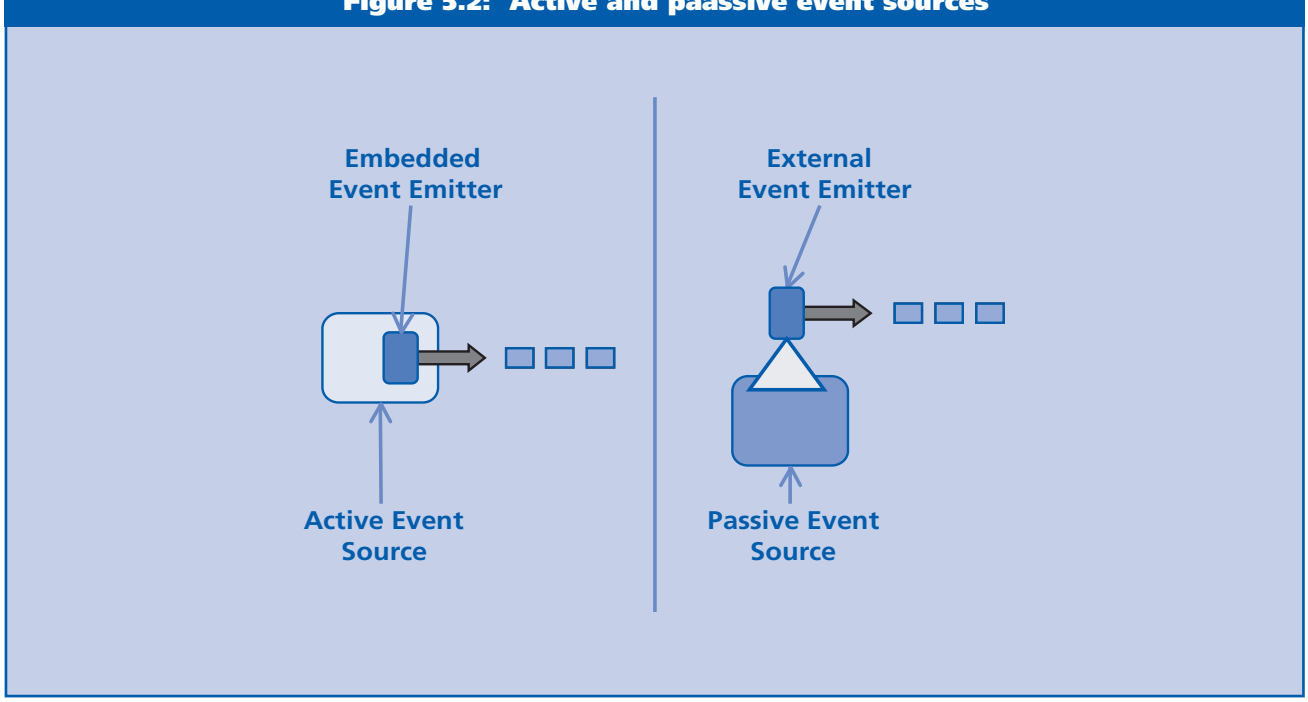
For example, an existing order processing system might routinely update an order database but it does not currently generate new order events. A middleware event emitter component could convert such a passive source into an active one by automatically generating new order events at runtime each time the order database is updated. New middleware is already emerging to provide this kind of non-intrusive configurable capability.

This is an exciting possibility for integration projects as it promises new (implied) function for application systems without having to modify existing business logic. However, care must be taken to configure the emitter technology to watch for the most appropriate internal events if it is to be effective. For example, database updates may not be the best activity to monitor in a transactional environment because of possible roll-back actions. A better approach might be to generate events corresponding to end of transaction indicators (if that is the intent).

Event packaging

Each event in an EDA system records critical information

Figure 5.2: Active and paassive event sources



about the change in state that has occurred. The architecture does not prescribe specific packaging for that information. However, middleware products offer a number of different options for packaging event information and APIs that might be used for event programming.

If events are to be delivered across process boundaries — but not delivered outside an application system — then proprietary packaging is the obvious choice. However if events are to be delivered across application systems or platform boundaries then industry standard packaging such as XML documents would be a more valuable choice.

Over time industry consortia may define standard event schema. This would facilitate interoperability between heterogeneous platforms both within an enterprise and between enterprises in particular industries. In the latter case the packaging of event data might be in the form of SOAP XML documents for transport using Web Services (and I will return later to the relationship between event-driven systems and service-oriented systems — see Services And Events).

Event listeners

Once events have been chosen and their sources correctly identified in relevant business processes, event listeners must be deployed in any EDA system. Such listeners may be tuned to recognize specific types of event or to act more generally on all events flowing through in a delivery channel. The most intelligent listeners may only recognize events that local event handlers are interested in — and not waste time on others.

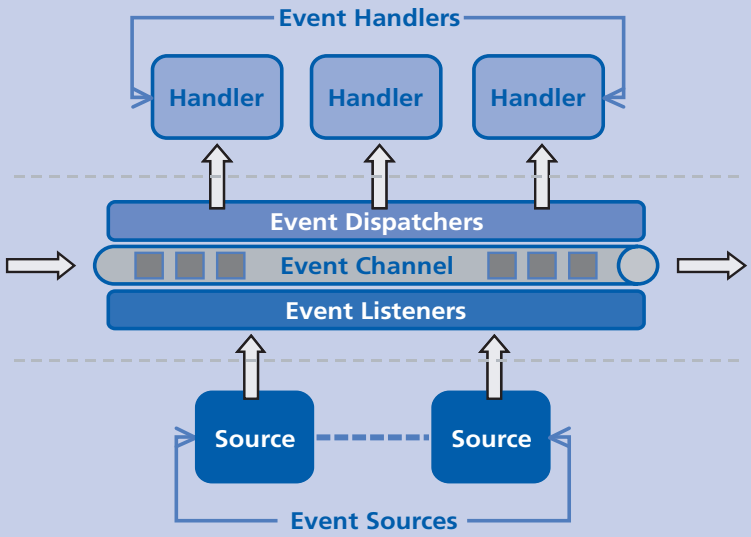
It is important to understand that event listeners may work in different ways according to the way in which they recognize events. In some systems, events have a simple 'universal' identifier that enables them to be recognized by table lookup or similar mechanism. Other event systems rely upon content-based recognition technology. The latter are more expensive but tend to be more suitable for an open cross-platform design as it allows for recognition based on different types of information recorded in an event and is configurable at deployment time.

An additional degree of flexibility and sophistication is offered by event listeners that not only recognize classes of event (all events matching certain content based rules) but also specific instances of event. For example, a listener that recognizes criminal activity by specific credit card number would need the instance level of sophistication. The intelligence built into listeners must be supported in turn by a mechanism by which event handlers register their interests at deployment time.

Event channels

When an event has been generated it must be delivered to

Figure 5.3: Event driven infrastructure



event handlers so that action can be carried out. A number of different event channels will be available within middleware infrastructure in the near future (Figure 5.3).

The simplest event channels will be procedural calls or object method calls between components. Remote procedure calls and distributed object communication mechanisms — such as IIOP, RMI or CORBA — may also be available for delivering events. These are all largely synchronous — near-instantaneous in nature.

Asynchronous, and optionally queued, delivery mechanisms will also be available for events. MOM middleware is already available from several vendors to provide such mechanisms for events packaged as messages. The characteristic that distinguishes event messages from other kinds of message (such as document, image, command, conversational) is that the payload is small and the delivery is intended to be one-way and relatively quick. Many MOM systems can already deliver messages with these characteristics and are well positioned to support EDA systems components.

Note, however, that EDA sources generate events without knowing in advance which components will receive them. There is no notion of events being targeted to specific end points. Events are not 'pushed' to their destinations. Rather the event channel must allow for events to be 'pulled' by components that have registered (or been registered) as the handlers for such events. This may mean that for certain events a multi-cast, broadcast or publish and subscribe protocol might be the most efficient delivery mechanism available. Once again MOM infrastructures are well positioned to provide this kind of efficiency but others may work as well.

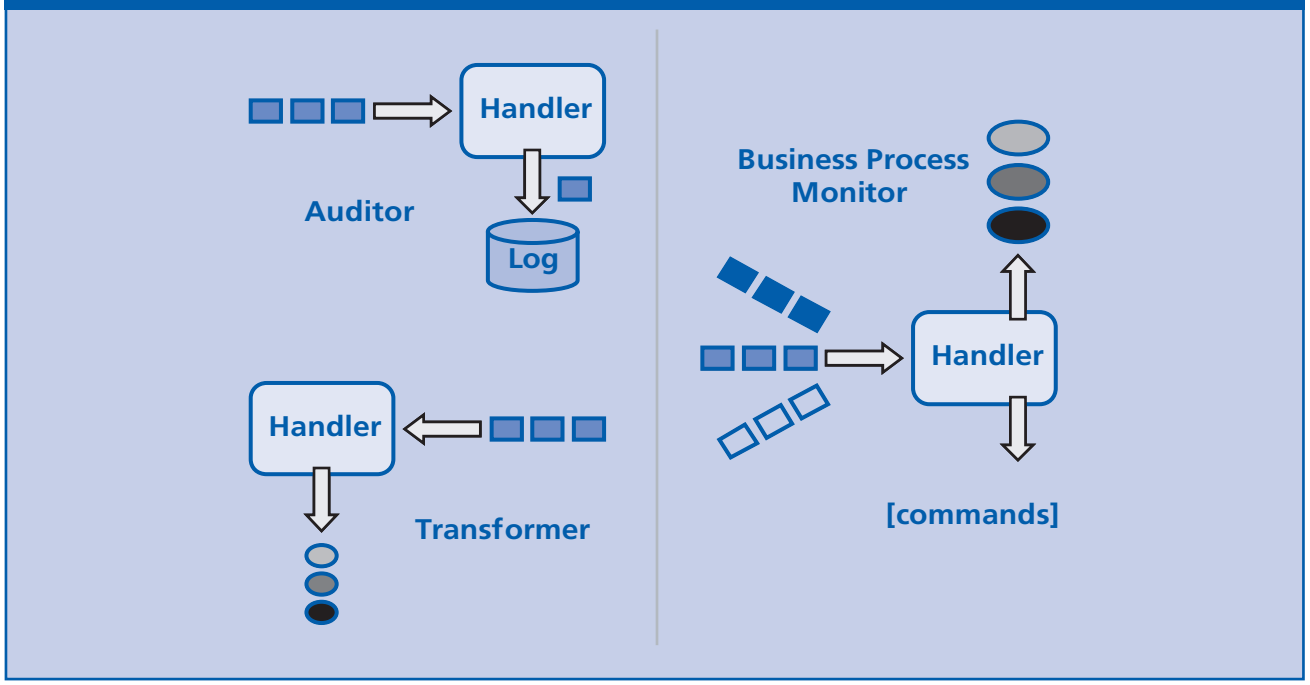
Event dispatchers

As events are recognized by listeners they must be dispatched to local event handlers or passed on to other components in an EDA network. The dispatcher component adds efficiency and scalability as the volume of events to be handled increases.

Listeners and dispatchers are often paired to provide this efficiency as they must share the event handler registrations. This combination acts like a message broker by correctly routing events to registered event handlers as they arrive.

Listeners and dispatchers are also often distributed in pairs with a transparent registration sharing mechanism. This can facilitate event message flow optimization in highly active systems. (Note, however, that there is no concept of a session between source and handler(s) in an EDA system. The nearest concept is that a kind of mirror image of session state exists between the handler(s) and the listener(s) that represent their interests. The placement of listeners

Figure 5.4: Event handlers



near to sources potentially represents larger constituencies of handlers, whereas those nearer handlers represent smaller handler constituencies.)

Event handlers

The event emitters, delivery channels, listeners and dispatchers may all be provided by configurable middleware deployed as an event infrastructure in an EDA system. Such an infrastructure (Figure 5.3) supports the goal of providing appropriate actions soon after an event or pattern of events has been recorded. The action to be provided is application logic for business events and may be systems logic for other kinds of events (such as infrastructure management events). Event handlers are the components that provide the necessary action.

For the most part, event handlers follow a simple programming model often encapsulated by software vendors in a development framework. Each event should be handled as quickly as possible with the minimum use of systems resource. This model is characteristic of all event systems. It delivers fast response where needed and scalability in high volume systems. However, this simplicity does not deny sophistication or capability in EDA systems. Many different kinds of handlers may be deployed from simple relays to complex pattern matching functions.

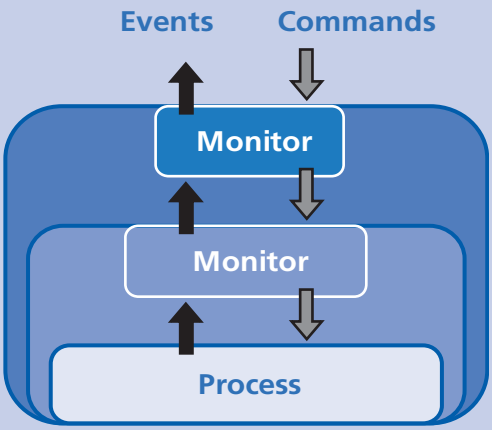
Some event sources provide streams of events that indicate business activity such as just-in-time arrival of supplies at a manufacturing location. Such events may be handled simply by writing the event data to an external log that is processed later.

Other streams may be handled in a similar way to provide an audit log. Simple one-in, none-out handlers will be commonplace in EDA systems. Many other simple handlers will transform one (incoming) event into a different (outgoing) event according to the semantic meaning of the incoming event in a given business process context. Such components act as both handlers and sources for events (Figure 5.4).

More sophisticated handlers will monitor patterns of events as they arrive. These handlers will often be developed as finite state or flow machines. As each event is handled it will trigger a well-defined transition from one state to another. Whilst not the only model that can be used, this allows for prescribed patterns of events to be recognized and for actions to be taken at critical points (Figure 5.5).

The state maintained during the handling of each active pattern can be made persistent using a database or local queues. Handlers of this type may be deployed as monitors for business processes. Such monitors could report either process metrics or process errors or exception conditions at critical stages — by emitting coarser grained events for higher level handlers to process.

Figure 5.5: Recursive monitor model



Monitors can be deployed at all levels in an enterprise system:

- **low level monitors might be deployed to ensure that a CRM system is maintaining customer response times and service levels**
- **higher level monitors might be deployed to ensure that time-to-fulfillment is being maintained by SCM and production systems working together**
- **top level monitors might be deployed to ensure that enterprise goals are being met across business businesses.**

Basically the same recursive model is being applied (Figure 5.5). This provides new levels of integration and management control.

Each monitor is handling events from process sources and, where necessary, using commands to reconfigure for better results. This model is familiar to most as a control feedback loop. The difference is that the event-driven infrastructure has been used to realize a business process system that is as responsive as is possible on a macro-scale using this model. (Note that this same model is used in systems management software and for implementing autonomic computing systems: this is good news as it implies a possible convergence on a set of technologies and skills that will be re-usable over time.)

Services and events

The emergence of new middleware to support events soon after similar middleware to support services is prompting some people to ask whether or not events and services are alternatives to solve the same problem or are complementary technologies intended for future business systems.

The answer is that events and services are intended to solve different problems and are fundamentally complementary. Event-driven infrastructure provides the mechanism for exposing significant business events as soon as they occur so that actions may be taken much earlier than legacy business systems might otherwise have allowed. This in turn will facilitate an enterprise strategy to improve responsiveness in a on-demand world.

Service-oriented infrastructure provides the mechanism for exposing significant business functions as re-usable services. In conjunction with industry standard Web Services technologies, A Service Oriented Architecture facilitates an enterprise strategy to improve IT productivity through re-use of software assets and by lowering the costs of integration across platforms. As valuable business functions are identified and deployed as re-usable services, they may also be enabled as event sources. Services may also be deployed as event handlers in future systems. In this case Web Services technologies may provide the event packaging and transport using already established one-way message exchange patterns.

Management conclusion

Event-driven programming has been in use for decades in certain classes of system — namely in GUIs and in real time applications. Some work has already been done to standardize on programming models for events — such as the J2SE event model, the Microsoft Windows event model and the CORBA event model although more work is needed on all of these. Existing middleware products already support event programming for presentation logic and MOM products provide support for event style messages. Leading edge enterprises have already implemented some event-driven business logic using these products.

Recent interest in event-driven systems has been inspired by growing competitive pressure to close the gap between dynamic real-world enterprise business and relatively static and unresponsive IT systems in support. Using an event-driven architecture it is hoped that critical business events can been exposed at source and reactive business logic defined for better control and response to external stimulate. If successful the momentum toward a more event-driven enterprise could lead to competitive advantage.

New and enhanced middleware products are already available to support an event-driven enterprise. MOM and Application Server products — from IBM, BEA, TIBCO, seeBeyond, Microsoft and others — will support deployment of event-ready infrastructures. Products like Neon Shadow Event Publisher have already demonstrated the feasibility of emitting events from passive sources. Yet more products from Network Systems Management, Business Integration and ESB vendors are also emerging with support for event programming. Although the class of event-driven business applications is not yet large it certainly looks as though the move to incorporate event-driven logic into business systems is destined to accelerate in the next few years.

Java IDEs versus Visual Studio .NET

Tom Welsh
Consultant

Management introduction

In the five years since it was launched, Microsoft's Visual Studio interactive development environment (IDE) has firmly established itself as the world's dominant programming toolkit. Building on the ideas that allowed Visual Basic to carve out a mass market millions strong, Visual Studio automates as much as possible, saving developers time and effort. It does have some limitations, though: not only is it limited to Windows, it is open to criticisms of 'dumbing down' the programmer's craft.

Meanwhile the popularity of enterprise Java has been damped by what some see as the confusion of the Java tools marketplace. Borland's JBuilder, the market leader, is efficient and powerful but is also complicated and expensive. Together with Visual Studio, it may be vulnerable to the growing popularity of Eclipse and NetBeans, the open source Java IDEs.

In this analysis, Tom Welsh considers what has been happening in the recent past. He also considers what is likely to happen in the future in this area that is key to middleware usage and deployment.

Integrated development environments

The IDE was one of the many new applications made possible by graphical user interfaces (GUIs). Compiler vendors hastened to add programmer-friendly features like source code browsers and debuggers to their own toolsets, leading to popular IDEs like Microsoft's Visual C++ and Borland's Delphi. IDEs turned out to be especially useful for creating presentation-layer code, as most developers (not unreasonably) prefer 'dragging and dropping' icons and other visual controls to writing page after page of program statements.

Microsoft pioneered the automation of middleware development in the shape of Visual Basic. Although Visual Basic was sharply limited in the type of middleware it could create — essentially, remote database access — it did allow such applications to be built with a handful of keystrokes. This was a big step forward from writing pages of source code to handle all the details explicitly.

Today's IDEs are expected to generate a wide range of middleware automatically, in response to quite high-level programmer statements. For instance, Visual Studio .NET allows Web Services, COM+ invocations and .NET Remoting calls to be created with one or two lines of code. Similarly, Java IDEs hide most of the laborious coding that would otherwise be required to make use of such features as JMS, message beans, RMI, RMI-IIOP and Web Services.

Typically, a basic IDE consists of:

- **a code editor**
- **a project library**
- **a debugger**
- **a build system (usually comprising a compiler and perhaps some kind of deployment facility).**

More sophisticated IDEs draw together many other categories of tools — including:

- **version control systems**
- **test harnesses**
- **documentation libraries.**

The idea of 'pluggability' has led to some IDEs becoming launching-pads for virtually all the applications a developer needs. This means that a particular IDE's desirability can depend on the selection of third-party products available as plug-ins. For instance Microsoft's Visual Studio .NET and IBM's WebSphere Studio Application Developer (WSAD), as well as the open source Eclipse (upon which WSAD is based), benefit from being the only IDEs tightly integrated with IBM's Rational's XDE modeling tool. Instead of creating UML designs in a separate modeling workbench, developers can do everything from within their favorite IDE.

Another important trend has been the growing popularity of open source IDEs and 'IDE toolkits' — software frameworks that provide all the necessary components for building or tailoring custom IDEs. At present there are two outstanding examples of this category:

- **Eclipse (initially introduced and promoted by IBM and allies)**
- **NetBeans (which has come from an original Czech development sponsored by Sun).**

Although Eclipse and NetBeans are both written in Java, they have been extended to support development in different languages and on multiple platforms.

The importance of the Visual Studio model

Microsoft did not, however, lead the way in fashioning IDEs. Arguably, Digital's VAXset and HP's SoftBench did that in the 1980s. Nevertheless Microsoft had watched what could be achieved and it has been the main beneficiary. The availability of comprehensive, easy to use and relatively inexpensive development tools has done much to strengthen Windows' position as the dominant operating system of the past 20 years.

That said, the IDE market has witnessed a thoroughgoing shakeout since 1999. Back in 1996-1997 there was vigorous competition between Borland, IBM, MetaWare, Metrowerks, Microsoft, Oracle, Sybase, Symantec, and many others. The writing was on the wall, though, when Microsoft began bundling its products together, with Visual C++ joining Visual Basic, Visual J++, Fortran and other compilers to make up Developer Studio (which has evolved into Visual Studio).

Soon Visual Studio included virtually all Microsoft's development tools — and it was priced to sell. It was extensive. It had five different language systems complete with:

- **editors**
- **debuggers**
- **compilers**
- **source code control**
- **a repository**
- **a development database**
- **and lots more.**

All this was delivered in a single package — costing about the same as competitors' single-language IDEs. More and more software developers took the low-risk option. They traded in their old tools for Visual Studio — in droves. With the whole gamut of Microsoft languages and tools installed, they could hardly go wrong.

Soon many of the millions of Visual Basic developers — the first mass developer market in history — had upgraded to Visual Studio. Ever since, Microsoft's competitors have tried and have been trying to find ways of tapping into that mass market — so far without much success.

Factors applicable to IDE choices

There are valid reasons for their failure. One factor that is routinely neglected in evaluating IDEs — this also applies as well to database management systems, operating systems and many other software products — is the level of expertise of the target user. Programming tools span a huge range of skills, equivalent to that between a pedal car and a Ferrari. Choosing between them involves several trade-offs.

Not only must an IDE be matched to the developer. The developer should also be matched to the project. Giving a novice an Eclipse download archive and expecting him or her to enable it to work, let alone use it to build complex distributed systems, is foolish. Yet telling a real expert to build that same system with Visual Basic might be a mistake of a different kind.

To make matters worse, there are different dimensions of expertise. One developer may be a red-hot programmer, but no good at (or uninterested in) design considerations. Another may be an average coder, but quick to grasp business requirements and architectural principles.

Managers without personal programming experience tend to over-simplify these issues, focusing instead on the overall objectives — ensuring the project is in on time and within budget. There is nothing wrong with that, but someone at some level has to weigh up the opposing arguments and take responsibility for the eventual outcome.

There has also been a strengthening trend to expand the market for programming tools by making them simpler and easier to use — usually by the introduction of:

- **menus**
- **graphical techniques such as drag-and-drop**
- **'wizards' (so called 'software advisors' that step the user through a series of decisions).**

Much of this can be traced back to Visual Basic, PowerBuilder and beyond. Today's culmination is Visual Studio .NET.

On the face of it, making IDEs easier to use allows a wider community of developers to work with them. That looks like good news to CIOs and project managers, because they are released from dependency on a small cadre of elite programmers. They can hire younger, less skilled and experienced, staff — and pay them less. Naturally, the idea also appeals to IDE vendors, who dream of coming up with a product that (like Visual Basic) sells millions, rather than thousands, of copies.

Other things being equal, it is obviously desirable to make programmers more productive by letting them work at a higher level of abstraction. Setting up a database table and linking it to a visual display used to take hundreds of lines of carefully crafted code. If, instead, it can be done merely by dragging a couple of icons onto a canvas and drawing a line between them, that is pure gain.

The risk arises, however, that if this approach is taken too far, programmers will be forced to 'drive in tramlines' — doing things only the way prescribed by the IDE. That can result in bulky, inefficient code and, potentially, in applications that cannot support their specified workloads.

Because, after all, any IDE is only a means to an end. It is the finished application or system that really matters.

The Visual Studio .NET value proposition

Visual Studio .NET is one of the most ambitious projects that Microsoft has ever undertaken. As the development environment for Microsoft's .NET Framework — the Company's alternative to J2EE — it bears a huge load of responsibility. Microsoft stands or falls by its success in cultivating the developer community. If a lot of useful, affordable applications are written for a platform, that platform will thrive. Microsoft's goal is to see that Windows thrives best of all.

Visual Studio .NET has to do two fundamentally different objectives that are not easily reconciled. It has to:

- **provide comprehensive support for Web Services and the rest of the new .NET functionality**
- **deliver an upward compatible migration path for developers who just want to carry on coding standalone and client/server applications.**

Throughout Visual Studio .NET, Rapid Application Development (RAD) techniques are used to simplify complex tasks using visual tools like the Windows Forms Designer and the Web Forms Designer. Programmers drag and drop controls onto a form, add business logic written in the language of their choice and then click a button to execute and debug the application.

Visual Studio .NET extends the RAD approach to the creation, testing and deployment of server-based solutions. Developers can drag and drop server components such as middleware, message queues, remote servers and Windows services onto a design surface — then add code, in any language, to connect the applications.

One of Visual Studio .NET's greatest strengths is the speed with which Web Services can be built. In principle, all that is needed is to add a single line to a class or method definition — which then becomes available as a Web Service. Other middleware services such as COM+ and .NET Remoting can also be added quickly and easily. Similarly, ADO.NET hides most of the complexity associated with back-end data sources and the Database Designer lets database developers browse their schemas graphically and add, modify, or remove objects such as tables, columns, indexes, views and stored procedures.

While freely admitting that Web Services can be written for many platforms — such as IBM z/Series (mainframes) or i/Series (AS/400s), Macintoshes and Linux — Microsoft relentlessly hammers home the message that 'it is far easier to create and assemble Web Services with Visual Studio .NET'.

How Java IDEs are fighting back

Enterprise Java has enjoyed an unexpected measure of success in corporate IT. In the long run, though, it is unclear whether Java can resist the enormous competitive strength of Microsoft. This is where IDEs come into the picture: where developers lead, applications and users will follow.

But Java IDEs by themselves cannot hope to win sales from Microsoft. At most they can aim to cancel out the strengths of Visual Studio .NET. To achieve even this limited objective, however, they must become easier and more intuitive to use, and — ideally — much less expensive than Visual Studio .NET.

The key question is whether an organization wishes to code its applications in Java. Visual Studio .NET allows the use of more or less any language, but only on the .NET Framework — which is at present confined to Windows. In contrast, Java IDEs support development and deployment on more or less any platform — but only in Java.

Borland's JBuilder is the clear market leader among Java IDEs — especially since the downfall in 2002 of WebGain (its Studio was JBuilder's closest competitor). The top end JBuilder Enterprise Edition comes with support for seemingly everything but the kitchen sink. It is mature, reasonably easy to use, well documented and supported, highly customizable and extensible and is integrated with many leading third-party products. In terms of price, though, it is being dangerously undermined by open source packages like Eclipse and NetBeans, which are essentially free.

BEA, one of the leading J2EE application server vendors, has sought to differentiate itself by proposing a new development technique. Codenamed Cajun, this project was first revealed in December 2001 when BEA's chief strategy officer Bill Coleman described it as 'a way of bringing Java to the masses'. Adam Bosworth and Tod Nielsen — ex-Microsoft employees who joined BEA when it acquired their startup, CrossGain — waxed eloquent over how as many as eight million Visual Basic developers, disillusioned by the difficulty of mastering Visual Basic.NET, would flock to BEA instead.

WebLogic WorkShop (the production version of Cajun) has a visual development environment that generates Web Services backed by J2EE components. It uses a special format called Java Web Services (JWS) — plain Java, with certain annotations added as Javadoc comments. A single line of JWS can represent 50-80 lines of source code.

JWS was wholly proprietary to start with, meaning that WebLogic WorkShop could be used only with WebLogic Server. BEA met this objection head on, submitting JWS to the JCP as JSR 181 ('Web Services Metadata for the Java Platform'). It has already been incorporated in Apache Axis, and IBM has expressed interest.

NetBeans

NetBeans was a Czech startup founded in 1997 by Roman Stanek, whose big idea was to write an IDE in pure Java, thus making it portable to almost any platform. Sun bought NetBeans in October 1999, and offered it as open source in June 2002, since when a sizeable community has grown up. (Stanek went on to launch Systinet; its WASP is one of the best Web Services development tools on the market today). NetBeans offers two distinct products: the NetBeans IDE and the NetBeans Platform.

The NetBeans IDE consists of the Platform plus additional

modules, such as an editor, a debugger, build control tools, version control and other development facilities. Visual design tools, wizards and automatic code generation are built in, as well as support for:

- **JSP**
- **HTML**
- **XML**
- **RMI**
- **CORBA**
- **JINI**
- **JDBC**
- **servlets.**

The current programming language repertoire includes C, C++ and Java. Drivers are supplied for most of the leading RDBMSs, including IBM's DB2, Oracle, Microsoft's SQL Server and mySQL and PostgreSQL (both open source). Thanks to advanced features like refactoring and user-configurable code templates, NetBeans advocates maintain that it is comparable with leading proprietary IDEs such as Borland JBuilder — with the advantages of full access to the source code and a zero licence cost.

The NetBeans Platform provides the underlying services required for most large desktop applications — window management, menus, management and storage, file access and the like. Since the NetBeans Platform is written in pure Java, it runs unchanged on a wide range of operating systems. In addition, NetBeans' adherence to 'pure Java' principles has important consequences. For example, it uses the Swing GUI (whereas Eclipse does not).

Today NetBeans has rather more than 30 third-party plug-ins. While impressive, this represents about one tenth of the corresponding figure for Eclipse. In 2002, the project recorded 370,000 downloads of the IDE and 25,500 of the NetBeans Platform, indicating widespread usage even if a proportion of those downloading the software are merely 'tire-kickers'.

Novell and Sun are among the most prominent companies to have re-used NetBeans as part of their own products. Compuware's OptimalJ — one of the best-known Model Driven Architecture (MDA) toolsets — includes the NetBeans IDE as a way for developers to add to the generated source code.

Eclipse

Demonstrating yet again that the first-comer does not always win in the long run, IBM announced Eclipse in November 2001 — a year and a half after Sun had open-sourced NetBeans. Yet the response to IBM's initiative was far greater, and today Eclipse easily dominates NetBeans. For instance, Eclipse.org claimed 2.5 million downloads in its first year— seven times NetBeans' run-rate.

The Eclipse.org board resounds with well-known names, including:

- **Borland**
- **Fujitsu**
- **Hitachi**
- **HP**
- **IBM (Eclipse has moved beyond IBM control)**
- **Merant**
- **Oracle**
- **Red Hat**
- **SAP**
- **SuSE**
- **Sybase**
- **and more ...**

In an unusual move for a standards consortium, the Object Management Group (OMG) joined Eclipse.org. Soon after, plans for UML plug-ins were announced.

Industry watchers were quick to notice the symbolism of the name 'Eclipse', suggesting that IBM's real motive was to put Sun in the shade. Sun was certainly upset that IBM seemed to be going head to head with NetBeans. (This was all moonshine, however, as the name 'Eclipse' allegedly refers to the last track on Pink Floyd's celebrated album 'Dark Side Of The Moon').

Like NetBeans, Eclipse began life as a proprietary product. It was the work of Object Technology International (OTI), a small Canadian company acquired by IBM in 1996 (but allowed to continue working nearly independently). OTI has an extremely high reputation in the object-oriented community, and it was there that much of IBM's VisualAge family of development tools originated.

Realizing in 1998-1999 that VisualAge for Java was showing its age, IBM and OTI began work on a next-generation Java IDE — based initially on the VisualAge Micro Edition (which was already implemented in pure Java). The idea was to create a highly extensible Java IDE capable of competing with Microsoft Visual Studio.

Among other things, that meant that one objective was to provide faster performance on desktop computers. The development team came up with the Standard Widget Toolkit (SWT) as an improvement over the normal Java GUI libraries — Abstract Windowing Toolkit (AWT) and Swing.

There are today 300+ publicly available plug-ins for Eclipse, including :

- **the Assisi GUI Designer**
- **Systinet's WASP Developer**
- **the Eclipse Modeling Framework**
- **Apache Tomcat**
- **the CodeBeamer team collaboration system.**

There is even a C# plug-in from the French company Improve SA. Furthermore, the Eclipse Tools Project has plans for a C/C++ IDE, a COBOL IDE, an Automated Software Quality platform (Hyades), a Graphical Editor Framework and a Java/XML framework.

The relevance of standards

While ease of use, automation, uniformity and cost are all important buying criteria, the choice between Microsoft and Java ought to involve some more abstract, longer-term considerations. Contrasting attitudes to standards are the most important of these.

Many journalists, and even some analysts, take a simplistic view of what they call the 'battle' between .NET and J2EE. In this interpretation, it is Microsoft versus the rest — with Microsoft having the advantage of being alone and thus having undisputed control over its software. Open standards, and the open discussion from which they emerge, are seen in a purely negative light — as hindrances to rapid, decisive action. Such an analysis treats the software industry as a kind of military struggle. But that is to see only part — and not the most important part, either. If vendors want to wage little wars among themselves, they are fully capable of doing so.

Arguably the industry's real function, however, is to supply the rest of us with the most powerful, reliable and secure software possible at affordable prices. In short, it should be trying to facilitate the ongoing computerization of our world. That function can be carried out successfully, in the long term, only through the unhindered operation of a free market.

Other factors that seem to have a beneficial effect are the open inspection and debate that take place, particularly in the open source community, and the open, vendor-neutral standards that evolve. The value of standards has never been better captured than in Dr. W. Edwards Deming's admonition: "Use standards as the liberator that relegates the problems that have already been solved to the field of routine, and leaves the creative faculties free for the problems that are still unsolved."

Standards are most valuable when they can be counted upon not to change arbitrarily, and this is where Microsoft's 'de-facto standards' are weakest. Precisely because the Company has absolute control over them, it can and does change them (or even replace them altogether) whenever its competitive strategy demands this. The Java world, on the other hand, is entirely built on vendor-neutral standards such as J2EE and CORBA. It also has a substantial — and steadily growing — overlap with the open source community, which is most obviously embodied by the NetBeans and Eclipse projects.

Management conclusion

In IT, as in many other fields, there will always be a dynamic tension between one stop shopping and a more eclectic approach to choosing the best products for each purpose.

Microsoft excels in meeting the needs of those who prefer 'one stop shopping', and if anything is going steadily further down this road. This strategy appeals to many decision-makers, especially perhaps those who are not inclined to get their hands dirty with technical details. In return for a single, known payment, many of the difficult decisions facing a CIO can be made to go away. Naturally this way of doing business suits Microsoft down to the ground, too. 'One of everything, sir? Certainly, sir — an excellent choice, if I may say so!'

The arguments in favor of a Java solution are less sweeping, but some find them even more persuasive. Once committed to developing with Visual Studio .NET, an organization may find that all too many options are closed to it. With Java, there is a wide choice of runtime platforms, operating systems and hardware.

It is not black and white, though. Big vendors like BEA, IBM, Oracle and Sun can offer combined software, service and support packages that are nearly as integrated as Microsoft's. Apart from Sun (and, of course, Microsoft) the whole industry is moving steadily towards the adoption of Eclipse as a standard IDE — and much more.

Last but not least, the difficulty of migrating from one J2EE server to another — from IBM WebSphere, say, to BEA WebLogic — is being reduced, notably by MDA tools. In due course, these will no doubt generate code for .NET as well as J2EE, making the adoption of either much less of a permanent commitment.

How ‘New Grids’ move beyond ‘Old Grids’

Mark Lillycrop
Principal Analyst, Arcati

Management introduction

About ten years ago, IT analyst Xephon (Mark Lillicrop’s former employer) produced a publication called the Dinosaur Myth. The slim booklet set out to refute the then-fashionable notion that downsizing from mainframe computers to distributed platforms (UNIX or PC based) was both cost-effective and efficient.

The Xephon publication argued, among other things, that one of the big drawbacks to small servers was (and indeed still is) their low utilization rates. While mainframes could — with effort — be driven up to more that 90% utilization, and typically run at over 80%, a UNIX server could easily be idling along at 15-20%. Meanwhile desktop PCs tend to be inactive for more than 95% of the time.

From the point of view of selecting the most cost-effective platform, this is a continuing and significant issue. Capacity planning and application sizing have always been arduous and complex activities outside the mainframe world. Indeed, for the last fifteen years or so, companies have seemed to prefer to throw hardware at the problem without considering how efficient any underlying architecture might be.

Asset management

The problem with the Xephon argument (compelling though it was) was that it became increasingly irrelevant. Hardware was becoming cheaper and, as long as a server or desktop PC performed its job at a reasonable cost and improved business productivity, utilization rates were of academic interest only.

In the event, Xephon need not have worried. Most mainframes were not replaced, as they continued — and still continue — to do their job extremely well. However their role changed. Increasingly they provided the secure back end server and data repository. In contrast, new applications (CRM, ERP, e-business and the like) sprang up in packaged forms on UNIX, then Windows and latterly Linux servers — thereby creating in the process a highly complex and even more expensive (to manage) infrastructure.

If this seems a little like a history lesson, it is extremely relevant to the current development of Grid computing. The point is that, all of a sudden, utilization rates matter again. In today's cost-conscious 'do more with less' enterprise, IT managers are looking for any 'incorrectly sized applications' and searching out opportunities to re-allocate processing resources in a more cost-effective way. It is no longer good enough to be good enough; we need to be seen to be looking for better ways of doing things, and streamlining IT operations.

Asset management (which, let us face it, has been a routine IT activity for at least a decade) has suddenly taken on strategic status, driven not just by economic considerations but also by the need to comply with corporate governance legislation such as Sarbanes-Oxley and Basel II. IT departments now need to know exactly where their IT resources are, show these are charged for, show (and when) these are utilized and what spare capacity is available**.** The bottom line is that IT managers are looking for:

- **ways out of the mind-numbing complexity of the average heterogeneous IT environment**
- **any means of aligning IT resources more closely with business requirements**
- **automated mechanisms which can keep track of changing resource requirements across multiple platforms.**

Equally relevant, they also want ways to pay for exactly the resources that are used and yet retain sufficient processing flexibility to apply additional capacity when this is needed.

The vendors' initial attempts at responding to these requirements — through capacity-on-demand and more flexible pricing — have never really rung true. But coupling the concepts of utility computing together with the technological potential of Grid computing is a much more compelling argument. The only problem is that, in many ways, there is still a sizeable gulf between Grid in its purest sense and on-demand IT services. For the time being, I fear, we will have to live with concepts like 'Old Grid' and 'New Grid' (of which, more later).

Grid computing in the commercial world

When Grid computing first came on the scene three or four years ago — with projects such as SETI@home — the value of spare computing cycles barely merited consideration. Grids were about super-computing, about scalability and flexibility, about the sheer technical achievement of being able to harness spare capacity on a grand scale and apply it to whatever scientific computation needed crunching. This 'Old Grid' was the province of universities, research departments and academic institutions — where the cost-effectiveness of IT is rarely high on the agenda.

What has happened in recent months is that, while 'Old Grid' continues to evolve (attracting growing enthusiasm for large-scale applications in the medical, life sciences, analytical, and automotive areas), vendors are focusing on making 'New Grids' more applicable to the cost and organizational issues facing commercial customers. As was written in the November 2003 **MIDDLEWARESPECTRA** (page 38), there is a long way to go before Grid computing (in the traditional sense) becomes suitable for transactional computing.

Quite apart from the practical difficulties involved in breaking down and distributing transactions between available processors and partitioning databases on the fly, co-operative processing across company boundaries is not culturally acceptable to businesses that place great store by watertight security and 'competitive edge'. There is a natural suspicion in the board-rooms of the world about any technology that relies on mutual support and a sense of community.

Support for transactions, as was concluded at the time, is likely to evolve as the new breed of Web Services converge with Grid Services at some point in the future, probably through the Open Grid Services Architecture. But while transaction support itself is a long-distance goal, there are many areas of commercial IT that can benefit from the application of Grid concepts.

Indeed, a parallel development is already underway, as the

principles and benefits of 'Old Grid' computing are absorbed and re-applied to existing commercial applications. The definition of Grid is gradually blurring, as vendors use the term to steer customers towards their particular flavor of utility computing. In some cases, Grids are being promoted as a natural extension of commodity clusters; in others, Grid techniques that are filtering into organizations to support numerically-intensive analytical applications will be gradually absorbed into mainstream IT.

The IBM approach

IBM is generally the best place to gauge changing trends in Grid computing, since Grid projects of every shape and size are going on somewhere within the Company. It was IBM that, back in January 2003, gave us a structure for applying Grid to a number of vertical markets, with 'Focus Areas' containing a growing number of packaged offerings for research and development, engineering, business analytics, enterprise optimization and government development.

The emphasis here was (still) on numerically intensive heterogeneous applications. But IBM was pointing towards a number of different 'entry points' for Grids in a commercial enterprise. The most notable of these (commercially) were in the areas of financial analytics and data mining.

As said earlier, Grid computing is only likely to be warmly embraced by CIOs if there is a clear financial benefit from re-organizing IT resources along Grid lines to regain lost (or unexploited) capacity. IBM's answer is resource virtualization. The IBM paper Grid computing: Accelerating the search for revenue and profit for financial markets by Matthew Friedman (*http://www-1.ibm.com/industries/financial services/doc/content/landing/973028103.html*) gives some indication of its current thinking.

Financial services companies are clearly a prime target for IBM, with their need for rapid analysis of financial data from multiple sources, and their newer obligations to put their house in order and comply with risk exposure legislation. As Mr. Friedman puts it: "Compute and data requirements will be massive for financial markets firms whose approach to Basel II [the international accord that obliges financial services companies to measure operational risk] is to adopt an advanced internal ratings-based model for risk. With Grid, they can lash together thousands of systems to tackle those massive and critical jobs quickly and efficiently." What IBM casually refers to as "lashing systems together" is no small undertaking. Introducing a Grid approach within a large financial services company is likely to require a great deal of external assistance (such as IBM's recently announced Grid consultancy services). That said, the rewards can be high and the Friedman paper cites RBC Insurance of Canada as being able to reduce a 2.5 hour job to 10 minutes and an 18-hour job to 32 minutes through Grid enablement of existing applications.

In cases such as RBC, the real enabler of Grid is sophisticated scheduling, and this is where IBM has an advantage. With its unparalleled experience of automated operations and 'autonomic' computing, IBM clearly has an ideal opportunity to develop ways of identifying underutilized capacity on the fly and applying it to the task in hand. IBM also has relatively mature partnerships with a number of specialist Grid companies — DataSynapse, Platform Computing, Avaki etc — that are helping to build on these advanced systems management capabilities in different vertical markets.

Oracle and Grid

Most of the highly-publicized Grid applications are still numerically intensive. To Web-enable commercial database and application server apps. requires a different approach. Vendors are currently wrestling with a whole range of data management issues.

For the next stage in Grid development, many eyes will be on Oracle, which has made Grid a cornerstone of its strategy. Oracle's approach, 'vision' if you prefer, is nicely described by the Company's Grid guru, Benny Souder, in the Web article On the Grid *(http://otn.oracle.com/oramag/webcolumns/2003/opinion/souder_grid.html*). Mr .Souder describes an enterprise where IT resources can be quickly and easily reallocated to different tasks as demands change — such as seasonal variations. His example is an Internet retailing business which needs to pump its IT resources into handling sales transactions before Christmas (when analysis needs are low) and then analyzing sales data and reporting after the festive season (when sales volumes are low).

The biggest obstacle to success with this scenario, in Oracle's view, is the impracticality of managing data and sharing resources across two or more isolated SMP systems, which is where the e-business and data warehouse applications might typically be located. By mapping these systems onto a commodity platform — such as Linux running on blade processors — it becomes feasible to build a database environment with the necessary level of flexibility.

This is the thinking behind the Grid focus of Oracle 10g, the latest version of the Company's DBMS. That software is beginning to provide the data management functionality — for example Transportable Tablespaces — which will facilitate Grid-style resource provisioning.

Oracle claims that the great advantage of its approach is its total commitment to Linux and commodity hardware, and the fact that Oracle has a single code base and API set. This eases the customer transition to Grids, although it does tie users firmly to one platform. (IBM has similar functionality for DB2, but IBM inevitably takes a more heterogeneous approach, more complex to support but allowing more scope for legacy support and interoperability of dissimilar platforms.) Oracle openly admits that the 10g approach to a Grid is a far cry from the inter-enterprise Grids espoused by the scientific community. The Oracle approach is, in effect, a Grid extension to its existing Real Application Clustering technology.

Marketing Grid solutions

Like the Hive technology from Tsunami Research (discussed on page 38 of the November 2003 **MIDDLEWARE-SPECTRA**) and indeed the Grid developments that are emerging from Sun's N1 and HP's Adaptive Enterprise, other vendors are indicating that there are many different ways of applying horsepower to specific tasks as and when required. This is in marked contrast to the traditional IT approach of tying CPUs and memory to specific applications and upgrading whenever the workload peaks reach unacceptable performance levels.

'Old Grids' have provided the theory and an impetus. 'New Grids' are gradually applying the technology to practical data center scenarios. Right now, everyone seems to want to climb aboard the Grid bandwagon and to prove that their technology offers a cost-effective way to provide auto-provisioning of IT resources and data virtualization. As a result, we are seeing the emergence of many islands of Grid capability, each of which demonstrate similar long-term commitments to open standards and blade technology but have little else in common.

In the short-term, this is not necessarily bad. Most current enterprise users have a substantial investment in one or several server bases. The last thing they want from a Grid is a leap into the dark. Indeed, the overwhelming user reaction to Grids seems to be that — however compelling they may be as a way of making more efficient use of IT resources — they have to be able to do so without adding cost and complexity in its own right. Ergo, Grids need to be seen as a way of evolving towards a new way of scheduling and utilizing IT resources, rather than as a fundamentally new approach.

Despite the vendor enthusiasm for 'New Grid', and the very substantial achievements in the academic and scientific worlds, Grids are still going to be a hard sell in the wider commercial environment — at least until business and economic conditions recover. In light of this, most vendors are now falling into two groups in the way that they promote Grid solutions. There are those that position Grids as a natural, beneficial extension to existing database products, application servers and operating systems — to be introduced gradually and with a minimum of disruption to existing processes and resources; this approach is relatively easy to achieve, and requires less direct sales pressure from vendors, but results may be slower and less profound. Then there are those who encourage initial implementations of 'Old Grid' approaches (where the benefits are clear and demonstrable in advance) for specialized, numerically intensive applications that can be readily sold to an organization; thereafter a push begins to introduce 'New Grid' solutions for other commercial activities which produces both a more aggressive approach (to Grid implementation) but may also produce more rapid results in the right organizations.

Oracle clearly wishes to be seen as the key player in the first group. We will likely see more and more names joining this group as Grid becomes a handy epithet for spicing up existing clustering solutions.

As for the second group, IBM is really in the best position to use its experience and success with scientific Grids to extend these capabilities into more commercially-oriented areas. Big Blue is also able to use its consultancy services to steer large organizations in the right direction, as it has been doing with RBC Insurance and Charles Schwab. Sun and HP probably belong in this second group too. Each has had success with scientific Grids, and you should expect to see them making more play for the financial analytics market — which is already proving to be a lucrative bridge between scientific and commercial Grid applications.

Management conclusion

In the end, of course, the proof of any pudding is in the eating. At a time of extreme caution and cost-consciousness in the marketplace, vendors need to be careful that they do not over-sell Grid as a universal cure to utilization and capacity planning problems. On the other hand, it is undeniable that most organizations are sitting on a vast pool of existing, paid-for yet under-utilized IT assets. Experience from pioneering Grid users should not be dismissed without close consideration.

Grid techniques are here to stay. The only question is how quickly they will be adopted by an understandably skeptical commercial IT user base. But adopted they will be, irrespective of whether this is via approaches shaped by 'Old Grids' or by 'New Grids'.

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

USA and Canada:
Spectrum Reports, Inc.

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

UK and Rest of the World:
Spectrum Reports Limited

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

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

Email and Internet

Email:
spectrum@middlewarespectra.com

World Wide Web:
www.middlewarespectra.com

ISSN 1356-9570

[incorporating FINANCIAL MIDDLEWARESPECTRA ISSN 1460-7220]