MIDDLEWARESPECTRA

incorporating FINANCIAL MIDDLEWARESPECTRA

Contents

May 2006

Volume 20 Report 2

Discovery with Tivoli

Kristof Klöckner
Vice President, Development
Tivoli Software, IBM Software Group

Management introduction

Kristof Klöckner is responsible for development in Tivoli Software within the IBM Software Group. He has overall responsibility for developing the tools and frameworks for system management — which embrace both operational management (products that automate individual management tasks) as well as the framework that pulls together information, processes and people's work environments to accomplish the tasks of managing IT services.

In this discussion, Dr. Klöckner focuses on 'discovery'. As he describes, this is an area of systems management and middleware which can have an enormous impact on how IT is managed, especially as informed management of applications and services becomes an ever greater requirement. If it is not undertaken systematically (and in an automated manner), organizations run the significant risk of operating with incomplete information and thereby taking inappropriate decisions.

Evolution leads to discovery

When you look at the evolution of system management, which is the term that most people would have used or talked about in the past several years, I find that customers are moving more and more:

- **from the traditional systems management view of managing individual resources or collections of resources (like individual servers or networks or storage or even applications)**
- **towards managing the delivery of services to their end customers (whether internal or external), for example over the Web.**

This is being driven to a large extent driven by the enterprise demand for IT infrastructure (including IT departments) to deliver a value that is both measurable and visible in the context of the specific needs of that business or organization. This is a very different scenario to the one in which traditional systems management operated in (say) the late 1980s and through the 1990s.

If IT, and IT departments, satisfy such a demand the promise is that organizations will obtain visibility of:

- **what is their infrastructure**
- **what does that IT infrastructure comprise**
- **how does it (the IT infrastructure) fit together**
- **the detail and information which can clarify what are the right questions to ask, the possible choices and then the right priorities**
- **facilitate (and then record) the changes — and consequences — of the changes that have been decided and are then made.**

This is why concepts like Configuration Management Databases (CMDBs) are attracting so much interest. When implemented thoroughly such a CMDB is a federation of all the information that an enterprise might own about its IT assets, their use and their configuration. Such a 'tool' represents an enormous jump forwards — much as a library, as an organized collection of information and pleasure, represents so much more than many books existing together in one place.

CMDBs and discovery

Another dimension to CMDBs, and the one that I am going to focus on here, is that they are gaining importance because of the implications of improved 'discovery'. This is occurring because discovery has the potential to be the necessary, automated enabler for 'feeding' CMDBs (Figures 1.1 and 1.2) with the type of data that is a prerequisite to make modern IT management possible. This is why it is becoming so important.

So, when we look at where discovery, and configuration management, will play a role we find that they have many diverse influences — from systems right through to business IT alignment:

- **what is in the many data stores (most organizations possess many more than one database, for example)?**
- **where are data and information located?**
- **what applications use which data stores?**
- **is IT delivering the kind of value that organizations need, especially when examined in the context of audit and compliance obligations?**
- **etc.**

If an organization does not even know what it possesses, how IT hangs together or how IT's many resources are used, that organization cannot realistically describe — or prove — that what it thinks it is doing is actually happening. Indeed, as the general business environment continues to become more complex as well as sophisticated, the need for informed insights rises just as the volume of detail available may be increasing even faster — to a point that can reach beyond the ability of the human mind to cope with, unless it has computerized assistance.

For example, I often find when I talk to IBM customers about (say) critical service interruptions that the root cause of those service interruptions is failed change management. Digging deeper, and this occurs more often than not for comfort, changes have failed because the people making them did not know or understand what the impact or implication of the change they are about to make might be. If they do not know what the impact might be, it is almost obvious to say that this is most likely attributable to the fact that they do not know, or understand, how 'what is relevant' hangs together.

The key point is that, in a world where we are dealing with services that undergo frequent change about whom they interact with and in a world of composite applications that always pull on a wider variety of resources, unless you possess this kind of automation and this kind of visible representation across multi tier relationships, you will exceed the capacity of humans to keep up with the complexity and change. We have all seen the spaghetti diagrams from large organizations that purport to show how applications connect to each other — but which are, in practice, incomprehensible because there is simply too much information. What would you rather do:

- **remain in (blissful) ignorance?**
- **make decisions based on presumptions of a presumed 'state of play'?**
- **try the impossible, and maintain these relationships manually?**
- **use automated tools which scan or crawl the incredible infrastructure maze to see what is happening and then analyze the data to uncover patterns that do not conform to an enterprise's policies — especially where it finds change (particularly those that may not have been authorized)?**

The reality is that spaghetti diagrams are common to most customers (if they ever bother to try to draw them). Yet these are real and do indicate the magnitude of the problem. What it boils down to is that:

- **it is not so important any more that you can manage individual elements in an optimal fashion**
- **you can manage end to end how all elements work together to provide a given service**
- **these services are able to adapt, all the time as they need to change.**

When you then look at some of the major trends — like the virtualization of infrastructure — the human can no longer keep track of where (say) a workload orchestrator has actually put the job or which piece of infrastructure has been assigned to what service. Everything is coming together.

Enterprises need to possess an understanding of the topology of your services and resources. The only way you can actually do this is by gathering the greater part of the information through some form of automation, with enough intelligence applied so that you can either fill a given services schema or suggest how services and infrastructure work together.

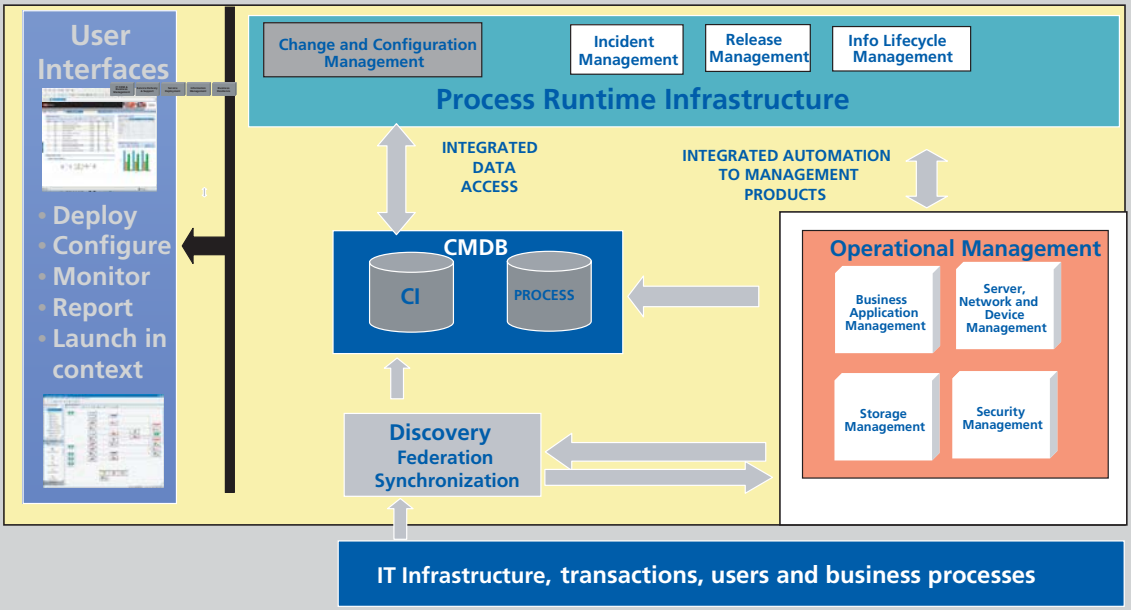
What has changed?

In the past, systems management played an important role. However, in today's terms this was relatively limited. What has changed?

In well run organizations, the systems operations people do know what is happening. But this is very much on the basis of snapshots — taken at selected times. At the same time, life is becoming more difficult — specifically customers are finding it ever harder to keep abreast of the degree of change that is occurring within the IT infrastructure, something that has steadily become more difficult as more devices and applications and services and end points are introduced.

The reality is that most organizations want to know pre-

Figure 1.1: CMDB concept



cisely what its people are doing with IT. Yet, without some form of mechanism to enable you to check reality against assumptions about reality, most organizations are almost flying blind. (You could liken this to having an active bank account, looking at it at the beginning of the year and 'just expecting' there to be money at the end of the year.)

What we find, even in the best run organizations, is that customers usually do know what the individual elements of their infrastructure are — but they rarely know, at the level of detail that is required for informed decision making, how everything connects or hangs together. For instance most well-run operations shops will know how many physical devices they have, at least if you look at the larger physical devices like storage, CPUs, network attached devices, etc. This is now familiar and, for most data/IT centers, routine. But if you start to ask about the next level of managing a business system — and then managing these resources from the perspective of the business system — most customers thus far do not have sufficient information.

Let me offer some examples. What our research finds is that customers do not necessarily know:

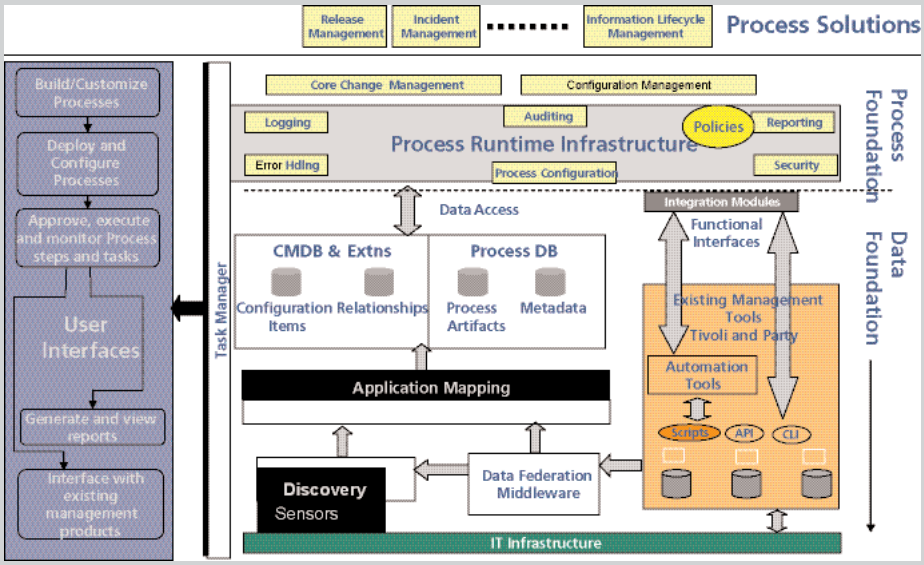
- **what database sits on what physical device**
- **what applications use that particular database**
- **what services use which physical resources**
- **and so on.**

Furthermore, even if they do know it (to some greater or lesser degree), the challenge that they face lies in keeping up with the volume of constant changes that are constantly occurring:

- **has an application moved from instance one of a database to instance three?**
- **has instance two of a database been moved to a different physical machine (perhaps because it needs more, or less, processing power)?**
- **which storage devices hold the data managed by a given database?**
- **is this being duplicated or even backed up appropriately in the context of the business and its vulnerability?**
- **etc.**

In one situation I was personally involved with, a number of years ago, an organization had an outage because a particular data device went down. It took this organization quite some time to understand, even before starting recovery, which applications were, in some form or fashion, 'touching' the device that had failed. When I talked to them recently about our new discovery capabilities — and how you can now build a resource/relationship tree — the reaction was that, if that capability had existed at that earlier time, it would have been so much easier to determine and then isolate the impact. In turned out that it would also have made it much easier to recover.

Figure 2.2: CMDB within IT service management vision



While this is just one example from one organization, it is not unique. Furthermore, increasing complexity breeds this kind of challenge.

What we find is that customers just do not know what is occurring within their overall IT infrastructure. Business units are constantly adding IT, even if it is not described as IT (you only have to think of mobile devices and sensors to appreciate how true this is). I have had more than a few encounters of the kind I have just described ... and I expect plenty more.

Discovery: what it is

Discovery is not new. In some form or other it has been around, in many different ways for a long time. For example, anyone going out and pinging a server is performing discovery at its simplest.

What is new in today's modern discovery is that not only do you discover individual items but you also discover how these fit together. In Tivoli (Figure 1.3) what we are doing is blending multiple technologies to capture configuration data with domain specific knowledge about the information that these technologies capture. In this context our sources have to be many and varied, including:

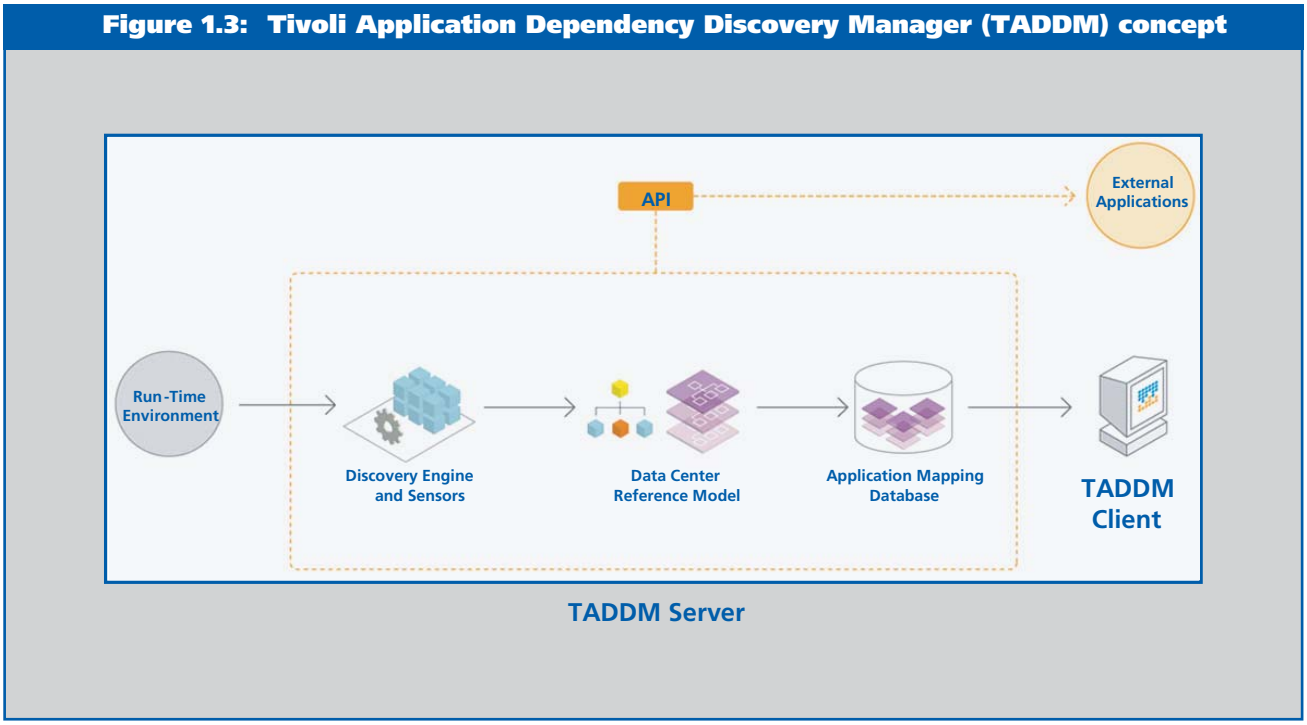
- **agents, which possess information about the end points on which they sit**
- **sensors**
- **sniffers**
- **existing instrumentation**
- **JMX**
- **SNMP**
- **SQL**
- **you name it ...**

The big difference here is that structured discovery departs from the scattershot approach, where you randomly discover things, and from the snapshot approach. In Tivoli we have the concept of building a model which shows the topology of everything that can be methodically discovered and how all this hangs together.

This does not represent any new understanding. But it does reflect a realization that has become more and more important in the past five years as we have talked more, and understood more, about SOA (Service Oriented Architectures), composite applications or even, at still lower levels, of functions like virtualization or even SysPlex. Unless you have a model that assists you to put in order what you have discovered — and then helps you to reconcile the many data pieces, to merge them, to analyze them for implicit as well as explicit relationships — all you are doing is scratching a surface.

What we find customers are asking for today is: 'can you:

Figure 1.3: Tivoli Application Dependency Discovery Manager (TADDM) concept



- **provide me with an end to end view?**
- **give me a view of a selected business service (like (say) mortgage processing)?**
- **tell me, for a given business process, what applications it uses and which data stores it touches?**
- **show me (visualize) how the whole fits together?'**

IT departments need this information if they are to make sure that they can deliver the appropriate level of service that is required by the organization. Equally, without this type of information, IT departments are operating blind or in guess-work mode. Increasingly this is unacceptable today.

Indeed, let me take this further. Discovery protocols can also be used for audits. In a sense, discovery enables IT defragmentation — it feeds the central repository of authoritative information (the CMDB), which in turn 'informs' the management products with consistency. As discussed later, without automated discovery, this task cannot realistically be accomplished by manual means.

In fact, our experience with business systems management (BSM) tells us that establishing configuration information and keeping it up to date is the greatest roadblock to company-wide BSM. Most customers I know have only managed to establish BSM for a small number of core processes — primarily because they do not have the depth of information about all their infrstructure and processes that they need.

Key changes

From this I will argue that that there are the two key changes:

- **creating and then populating the model, the description of how it all hangs together at a given point in time is one change**
- **maintaining the accuracy of the model on an ongoing basis (by automating whether anything has changed and, if so, documenting and recording what has changed) is the second.**

Snapshots may have been useful. But nowadays they are insufficient.

What is more, if you have this kind of model, you are able more easily to develop a sense of what your preferred (in terms of the business requirement) configuration should look like. Rather than build from the IT assets and infrastructure up, you can invest in IT from the organizational or business requirement down. Plus you can develop ideal templates or patterns or build standards ... and compare existing configurations (or possible configurations) to such gold standards. In turn this enables you to observe polices, as well as to enforce compliance.

Indeed, compliance is becoming an additional reason for discovery, the CDDM and related disciplines. As one customer recently told me, it wants to use IBM's discovery products to ensure that all its 500+ WebSphere images have their parameters set according to that organization's established best practices.

Possessing tools which can, for instance, flag differences from a 'gold standard' configuration is critical. Thus discovery has become a complement to automating the setting of these parameters, and can catch 'escapes' (differences from what is expected).

Building the model: top down or bottom up?

This begs the question: how do you derive the model to start with? To me there are various ways of approaching this. What I have personally seen is that there is no perfect answer for all situations. There are top down approaches and there are bottom up approaches. Each may be more or less appropriate to any given organization's specific needs.

What we do find, however, is one common denominator. Customers almost always want to start by understanding how their most important, their most visible application functions, fit together. Although not always, these are most likely to be applications relating to customer facing activities. If your organization is a bank, this is likely to be the customer portal; if you are a manufacturer, this may be your order entry system or delivery system.

This is something that has changed significantly over the past year. My belief is that this change represents an opportunity, or a threat, to alter what constitutes the value promise of any brand.

If an organization's priority systems are not under control, then they are capable of being messed up. If they can be messed up, that brand value can be placed in peril. This realization has driven an understanding that organizations must manage their business systems, and especially the IT shop, from the perspective of their prime business activities. This is the top down approach.

To make sense of what is needed, organizations must

understand how the functions that are delivered by multiple applications and their supporting systems hang together. Only when this exists can you correlate what you have discovered in the IT infrastructure with what is happening in the overall business process.

Sometimes, however, and we often find this inside infrastructure-driven activities, the other approach — the bottom up approach — may be as valuable. Think about port-scanning or some other low level mechanism. You look for what is out there. Then, through some mechanism, like observing the message traffic between systems, you find out what are the true (rather than supposed) interaction patterns. From this data you can then build your configuration database.

This bottom up approach is also useful when you have had 'disorderly' growth over a certain period of time and you want to consolidate. We have one customer example, where a large service provider wanted to consolidate data centers. The reality was that IT did not really quite know what was installed or where. More importantly it did not know what impact any changes might have. What do you dare do if you need to make changes and you are not quite sure what will happen once you start making those desired changes?

Whether you start from the bottom or the top, you have to discover what you have and how everything relates to everything else. Without this, how can you understand the consequences? Would you like your surgeon to start cutting before he had looked at the MRS? Do your executives wish you to expose the existing brand value to unnecessary uncertainty?

Discovery — a technology or an approach?

I believe that thinking about whether discovery is a technology or an approach enables me to isolate one particular point I wish to make. I have been asked on several occasions about whether it is one or the other — and this is a powerful leading question.

In my view, discovery is the two combined. Yes, there is a set of technologies that discovery uses. But, unless you really tie these back to a model that represents what you discover (the model of your data center(s), infrastructure, applications, network, services and whatever is relevant), you cannot fully utilize what discovery technologies provide. Equally, without an approach which models what is discovered, all the sexiest discovery technologies that exist will not count for much.

At some point in time you do have to start with human activities, especially for a top down approach. In a perfect world you would have already employed a standards-based tools model. You would use this to build the descriptions of your business processes and their various task implementations. If you already have this kind of information you can extract it and re-use it as a starting point for a model. But at some point in time you need to have invested in such descriptions.

Alternatively, if the bottom up approach is more relevant, the need is to deploy analytical discovery software tools against what exists. These gather data and information which is stored and sorted and then analyzed to create the model. In an ideal world this would be 100% automated. In the short term, this will likely be accomplished in a semi-automated way, using human interactions to assist the interpretation in the initial building of the model.

This bottom up approach is feasible, but not in all circumstances — yet. It is certainly feasible if you are looking at physical relationships, if you are looking at what device has a network connection to what other device. If you want to look inside IP networks, for example, Tivoli has had lots of functions to offer in this space for many years.

Will organizations use one approach over the other? I think they will use both. Indeed there is some beauty in this because, in the end, all the effort will be converging — with the output being the description of that organization's topology of services and supporting resources, and stored in one or more (federated) databases.

In this context the CMDB/database 'in the middle' that not only receives all the 'discovered' information but also, using additional tools, monitors and records what is happening all the time. Thus it records changes as these occur. It is no longer a collection of snapshots but an evolving record of what has happened (and why).

Thus, in a CMDB you would expect to describe configuration information plus information about simple relationships (at least to start with). In addition, the CMDB, if properly populated and established, would also offer an ability to define other relationships and extend and aggregate the attributes of the various configuration items.

Then, using policies, scans, specialized discovery technologies and the information collected by traditional systems management tools in the process of their normal operation, the entire environment will constantly be 're-discovered'. From such a base it is practical compare and contrast what exists with what is changing. In effect discovery

becomes a never-ending automated process of its own that continues to run and run. In so doing it can then provide the information upon which informed decisions can be based.

I should also re-emphasize a point I made earlier. The capabilities and output of traditional system management tools remain highly relevant. They are an essential input to the CMDB. Discovery is not a situation where rip and replace has to apply. Instead, existing investments in IT management must be used to deliver ever more value (Figure 1.4).

In effect this is 'infrastructure crawling', exploiting the various differing degrees of technology intelligence built into the IT infrastructure. There are devices that announce themselves; there are management tools that introduce agents — which we know from experience work well and offer excellent contextual knowledge about what they do, about 'their' devices and 'their' end points, 'their' environments. Yet to mount all this and then reconcile it and make decisions about who and what to trust is not easy: for one, there is a huge amount of data. The beauty of tools like those now offered by ourselves is that you can:

- **either decide, as a policy for this attribute, always to trust 'Tool X'**
- **or you can surface discrepancies and reconcile these manually — if you feel you do not want to trust a particular tool.**

The significance of Collation and MicroMuse

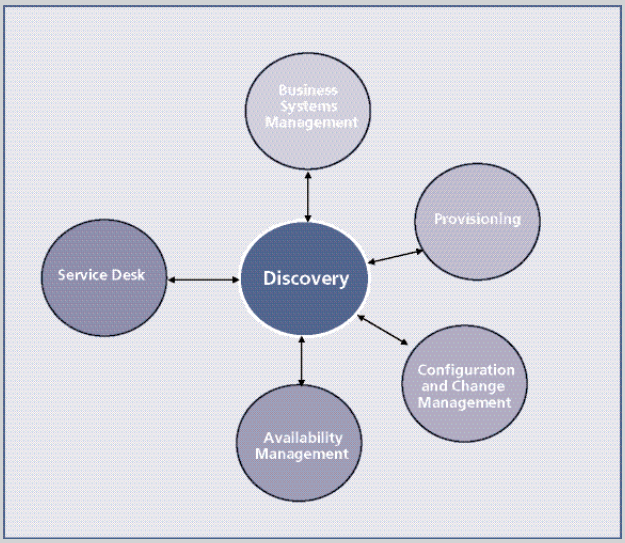
In 2005 we bought Collation. We did this because Collation added different discovery technology with its intelligent, model-driven semantic understanding of complex IT environments across a broad spectrum of today's technology stack.

Collation had invested 5+ years in building sensors and collectors for literally hundreds of environments. These were many and various:

- **from Web Servers**
- **to databases (from the obvious like DB2 or Oracle through to lesser known ones like Postgres or mySQL)**
- **to common application suites (like PeopleSoft or SAP)**
- **to other less common third party applications to servers**
- **to secondary sources of structured information (like directories)**
- **to elements of the physical infrastructure, like load balancing, storage, etc.**

Interestingly, the other big acquisition that we made recently (MicroMuse) had already struck an OEM relationship with Collation to augment and complement the various discovery capabilities that MicroMuse offered for the

Figure 1.4: Discovery complements existing IT assets



network with application-related dependency discoveries. Add this to what Tivoli was already offering — in discovering and presenting, for instance, mainframe relationships — plus the relatively recent shipment of the capability to discover relationships between services (by looking at the message traffic between services) and IBM now offers a pretty complete discovery portfolio.

Discovery is only as good as the picture is complete

That said, what is important here is that, while we now offer a fairly complete picture, we also realize that Discovery is only as good as it can make the picture complete. We understand that there will also be other infrastructures where the 'element owner' of this infrastructure will have better information than we can discover.

Tivoli is happy to work with such element owners to make use of their understanding. Tivoli's discovery is not an absolute. To obtain the fuller picture we know we must, and will, work with third parties — including other system management vendors. What matters is the information generated and collected, not who found it.

This is why it becomes critical to use standards which enable the federation of a variety of existing information sources. This is one of the things that Tivoli is going to drive forwards. We are going to encourage and pursue agreements between the larger owners of infrastructure, and infrastructure management, software to make it easier to exchange and import and to federate information that can be gathered.

This may sound hard. I do not think it needs to be. Basically, at least for infrastructure description, we are all operating off the Distributed Management Task Force (DMTF) Common Information Model (CIM).

We have added, and I am sure we will continue to add, extensions. We will agree on how we add extensions to models — particularly how do we deliver information into and obtain it out of these models. This makes commercial sense — because it is what our customers are asking us to accomplish.

Ultimately, when you look at how you will want to make end to end decisions, you need to be able to correlate information from multiple sources. We know that customers will find themselves in situations, for instance, where they may be using (let us say) IBM software configuration management but they also use a help desk that collects travel tickets from (say) Remedy or Peregrine. They may have an asset management system from yet another vendor.

When you have a crisis situation you want to know what is happening and why. The monitor may shows a poor response time in one area of the infrastructure. To resolve the problem you will want to know if:

- **anything changed lately**
- **has somebody already opened a trouble ticket**
- **is there a problem in the network**
- **has a component failed**
- **it is a downstream (or upstream) dependent application causing a bottleneck or slowdown**
- **and so on.**

Lessons learned

My first, and personal, lesson learned is a general one. Discovery is ultimately about integration. Since it is an integration task, the federation of information and the integration of information gathering tools, requires standards and an eco-system approach. No one vendor is going to be able to cover it all and, though this is a general description about the state of the industry, it cannot be overlooked.

As to what discovery (and integration) projects are successful (or not), I have found that there are some significant indicators that emerge from experience. The first one is that discovery always happens in a context. That context is almost invariably a cross-function initiative that requires executive support and visibility; thus, in order to achieve traction, it needs top level support.

When we look at some of the Collation customers (for example) it is obvious that many of them were successful because they were CIO or CEO driven initiatives, with an objective to deliver a concrete business value. Using discovery can expose relationships and make visible what is actually happening in terms of change; indeed understanding change becomes also a catalyst for change.

My last lesson — and it is almost ironic that this applies to any kind of business integration — is that the projects that succeed are those where the customer starts deployment in the most critical and visible application areas. Customer-facing applications, those where any management effort directly relates to the value of an enterprise and its ' brand', have an impact on an entire organization. There is a will to succeed in these projects, because not succeeding is too expensive for everyone. Successful discovery projects rarely are initiated as pure infrastructure projects; rather they:

- **start with a given pain point**
- **focus on one or more of an enterprise's most important application areas**
- **deliver value to the 'enterprise's customer' (and thereby to the enterprise)**
- **then become the foundation upon which to build.**

Management conclusion

As Dr. Klöckner describes, the need is to look at discovery as a foundation and enabling technology to manage integration. The role of discovery is to create, and the keep up to date, the state of an enterprise's total infrastructure, from services to applications to hardware to communications.

Successful users will build a model of the enterprise — whether manually or semi manually — and then use the tools and approach that discovery represents to:

- ***flesh out that model***
- ***prove where it is wrong and where it is right***
- ***add the pieces that were not known***
- ***keep updated what has changed, and not changed.***

In so doing discovery overcomes the human limitation that persists when complexity becomes the barrier. By starting with the human element, it is possible increasingly to automate discovery and to use ever more sophisticated tools to build a better an better picture of how an enterprise works. In turn, this means that managing becomes possible again — because there is sufficient real information on which to base appropriate decisions.

Change and discovery in the IT infrastructure

Mark Lillycrop
Principal Analyst
Arcati

Management introduction

Managing change has become ever more important. Change is occurring constantly in most organizations. Discovering what changes have occurred, and why, is of increasing importance.

In this analysis, Mark Lillycrop examines how the management and documentation of change has been evolving. He specifically considers discovery as the key starting point. He then continues by considering the implications for IT (and vendors). Finally he provides a brief commentary on the four major vendors in this space — BMC, Microsoft, CA and IBM.

Change ... it is constant

Change is the one unavoidable challenge faced by all large enterprises. Any organization that refuses to adapt to new technologies or to seek out new commercial opportunities is likely to disappear rapidly from view.

For IT departments in 2006, and beyond, this involves a constant battle to meet more and more exacting service management goals and to do so against a backdrop of continual change. As the organizations they serve re-invent themselves, go through mergers and acquisitions, abandon declining markets and pour resources into developing more lucrative ones, the IT operation must continue to maintain the infrastructure on which the ever evolving organization depends for survival.

Within this ever-changing environment, one of the most critical functions of middleware is to keep track of all IT assets — and to make sure that the organization knows exactly:

- **where they are**
- **what they do**
- **how to exploit them to the full.**

This means monitoring a whole range of IT elements that might or might not show up on the traditional system management screen:

- **physical components (hardware, software instances, release levels, etc.)**
- **logical components (for example, code which is being adapted and re-used in various ways by different development teams)**
- **data (particularly metadata and performance information which might be presented to the organization inconsistently in multiple formats by different tools).**

The key to managing this vast pool of information resources is automated discovery:

- **locating IT assets and so-called configuration items**
- **establishing their role and their relationships with other items**
- **noting every change that occurs to these relationships.**

An essential component of any discovery mechanism is what has come to be known as a 'centralized configuration management database' (or CCMDB), where assets and relationships are recorded. The level of sophistication and scope of this data store will vary from one organization to the next — but in all cases it should provide a single integral source of information on the use of IT resources which can then be used for reference by developers/programmers, call center staff, end users and even applications themselves.

A fully integrated change and configuration management architecture — where every IT component is tracked throughout its life-cycle — still remains a pipe-dream for most organizations, and most vendors of system management tools do not yet have all the pieces in place to support such a capability within their frameworks. Nevertheless, regulatory compliance and 'best practices' — such as those defined by the IT Infrastructure Library (ITIL) — are placing increasing pressure on CIOs to put formal procedures in place to manage the end to end process.

The first stage is to identify physical devices and components as well as individual instances of installed software, and then to move on to 'discover' applications, services and processes. Let us look at the various steps in turn.

Step 1 — discovering physical resources

Network infrastructures in today's large organizations are very much more complicated than they were a decade ago. The basic communications protocols may be more standardized, but the list of attached devices is often in a constant state of flux. One of the first decisions that any company must make is how accurately (and how often) it needs to know exactly what is connected to the organization's IT backbone.

At the most basic level, hardware detection is now well understood and widely applied, although it is often claimed that large data centers do not know exactly how many distributed servers are under their control nor where they are located. This may be true in some cases, but with the sophistication of today's asset management tools there is no excuse for losing track of any device that is permanently network-connected.

The major problem has arrived with the plethora, and constantly expanding population, of portable and mobile devices that connect or dial in to the corporate network — sometimes behind the firewall. Such devices pose a whole range of potential security and administration problems, which are compounded by the prevalence of hand-held client devices and the increasing use of wireless connections.

How these security issues are addressed are part of another discussion. From the asset discovery perspective, the important thing is to know what devices are exist — and what they are doing. As before, devices that are installed and managed by central IT are far easier to track than those introduced by 'virtual office' staff who administer their own hardware and connect up to access corporate data via a range of different means.

Accurate hardware tracking is also critical from a network performance perspective. Every fault, performance degradation and service loss must be traceable to the router or hub responsible. It is increasingly important to establish exactly how every network component is behaving, often on a 24x7 basis.

Step 2 — discovering software assets

Once the hardware has been identified, the focus moves to the system software and applications that are running (or can be run). Software asset management came into its own during the run-up to the Year 2000. Across the world, users of legacy software realized that they had to test it for date compliance. But they could not test it until they had worked out where the software was.

Consequently a vast software discovery exercise took place, resulting in the largest software inventory known to man. It remained valid for, at best, 24 hours.

In retrospect, the IT industry as a whole abandoned a unique opportunity to maintain the vast software configuration database that it had created. Instead of knowing, day by day, where all the software was, what version, release and patch it was up to, and exactly who was using it and for what purpose, most organizations returned to a complex mixture of basic SAM tools and techniques — augmented by some educated guesswork.

Over the last couple of years, SAM has become a big issue for large organizations again, driven by several factors:

- **cost management: IT departments are under constant pressure to cut costs and to do more with less — and discovering duplicate software instances or underused software is one of the easiest ways to save money**
- **software compliance: compliance is a minefield, and there are an increasing number of regulations affecting software licensing; large organizations tend not to infringe software agreements deliberately but errors are widespread, either because of underlicensing or lapsed support agreements or because the business has simply lost track of the way software is used following an organization merger or major reorganization; moreover, senior managers face vicarious liability for software misuse by their employees, and junior staff are notorious for swapping and 'borrowing' software to which they have no legal right**
- **security: outdated updates and patches can lead to software vulnerabilities that potentially threaten the whole of an organization's network which explains why many organizations have strict software distribution processes that permit patches to be applied as soon as they are available; this process, however, has to be fully integrated with the software discovery mechanism to make sure that every instance of the software is located in order that the appropriate patch is applied.**

Software asset management is inextricably linked with change management. In order for call center — and other support staff — accurately to be able to locate the cause of any given service problem, it is essential that they have the tools are in place to:

- **discover changes that are occurring in the use of software assets**
- **be able to gauge the impact that these changes will have across the organization.**

Step 3 — resource discovery in the development process

Hardware and software asset discovery are relatively straightforward issues. There might be specific problems or challenges in identifying certain components but essentially the items to be discovered are well defined and changes are not difficult to track.

When it comes to discovering instances of code and relationships between applications, much becomes more complicated. Changes to applications occur regularly within any large enterprise or organization, and responsibility for those changes rapidly, and often, becomes a political issue.

One of the key tenets of IT governance is that every change must be 'owned' by someone (usually the person initiating it). This person has to take responsibility for the impact that the change has on any other user. Within a very simple IT architecture, changes can be managed fairly easily. Requests are typically received by the call centree passed to IT for approval, implemented and tested.

But, in a more complex scenario, any significant change has to go through an impact analysis process. In such a situation, it is critical to record:

- **why the change was requested**
- **how it was implemented**
- **how change affects each relationship.**

A whole layer of middleware is gradually emerging to manage this process in larger enterprises, with tools to manage change, configuration, asset discovery and end to end impact. For more involved changes, there is also a need to manage successive software releases as well as to co-ordinate changes made by different teams of programmers in different locations.

As organizations develop more complex change requirements, there is a clear need to:

- **automate the process**
- **remove any manual intervention that might, however inadvertently, lead to error or inconsistency.**

Step 4 — service and process discovery

In today's service-oriented environment, discovery needs to make a further leap in sophistication. The whole justification for 'services' — rather than applications — is that they:

- **streamline the process of re-using software functionality**
- **enable discrete functions to be discovered and assembled on the fly.**

This means that automated detection of services is absolutely critical. It moves the whole focus of 'development' onto the configuration repository where service functions and relationships are recorded.

Web Services, in particular, demand a totally reliable structure for locating appropriate functions as well as assembling them into workable applications. Often the logic itself, and the relationships between component services, will cross organizational boundaries. This highlights the need for a single standardized approach to harnessing these resources.

What really differentiates the SOA environment from other 'users' of discovery services, however, is the automated and dynamic nature of the search. Services need to be able to search for each other on the fly.

The key element of this search, of course, is some kind of business registry where all relevant services are cataloged and categorized. When a service is made available for general use, its details are recorded in the registry where they can be discovered by 'consumer' services. The two pieces of logic then communicate via messages, the consumer using the provider to access specific information or to perform a given task.

Because such services are, by their nature, platform-agnostic, the discovery mechanism that is built around the business registry is the only practical way to:

- **decouple the service consumer from the service provider**
- **enable the two sides to communicate in a dynamic fashion.**

In the standards-based Web Services environment, much effort has gone into providing a universal mechanism for service discovery — built around the Web's lingua franca, XML. UDDI (Universal Description, Discovery, and Integration) is an industry-wide initiative to provide a universal business registry, under the control of the OASIS standards body. It is populated with service definitions that conform to WSDL (Web Services Description Language) and uses SOAP (Simple Object Access Protocol) as the transport.

This small cluster of standards has become the de facto means of bringing common businesses services together. Although it is not the only way of building a service oriented architecture (and indeed is not necessarily appropriate to organizations that have more specialized requirements), it does provide a practical way of building business services that are dynamic, scalable, and entirely flexible — thereby enabling IT departments to respond to changing business requirements in a way that had hitherto been impossible.

Managing the discovery and provision of IT services

Within the large enterprise, service-oriented solutions have a great deal to recommend them and there can be few IT-focused businesses that are not considering ways of building SOA capabilities into their existing infrastructure. From the point of view of the traditional systems management environment, though, the challenge is expanding the current enterprise toolset to accommodate this new paradigm.

For many large data centers (and the vendors that support them), the whole change/configuration/asset management

process has been about starting with a static IT infrastructure and then tracking any changes that might or might not have an impact on the service delivered to users.

This approach has been fine-tuned to a very high level of maturity in many large organizations. But it no longer fulfills the requirements of sophisticated IT users. The whole point about SOA is that it allows IT requirements to be expressed in business terms. This decouples the provision of IT services from the underlying infrastructure. In turn, this means that IT departments and their vendors have to find a way to:

- **move away from the hardware/software-centric view of the enterprise**
- **express the performance and availability of IT resources in a way that does not restrict or obstruct the dynamic, scalable use of business services throughout (and beyond the boundary of) the enterprise or organization.**

Not surprisingly, the vendors of middleware and management tools and services have responded to the need for a business focus by embracing 'best practice' service management frameworks such as ITIL. To varying degrees, they are attempting:

- **not only to identify a common ground in their discussions with business users**
- **but also to find a way of managing established IT assets and new services in a consistent way, so that they can deliver a single business-oriented view of IT resources.**

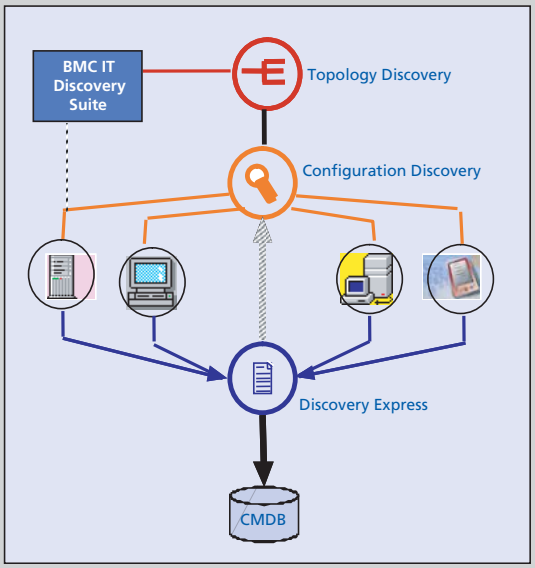
How are the vendors shaping up?

Looking across the whole enterprise management space, middleware and system software vendors are broadly moving in the same direction. BMC, for example, is one of the strongest adherents to the ITIL message, and its IT Discovery Suite closely reflects the best practices expressed within the ITIL literature. As can be seen in Figure 2.1, this company has tightly integrated its tools for managing IT discovery, configuration and topology with its configuration database (CMDB).

The database is the centre of the service management environment, and provides a single source of reference on the state of IT resources and the impact that any change will have on the relationships between IT assets. BMC is particularly strong in data management and is, arguably, in the best position to integrate the tracking of data integrity in and across traditional and service-oriented applications.

Microsoft has a very similar approach to BMC in many respects, although — always keen to take a Windows-centric view of the world — its change and configuration objectives are expressed in terms of its proprietary MAF

Figure 2.1: BMC and its CMDB



(Microsoft Application Framework) rather than industry-standard service management frameworks. Again the CMDB is positioned at the heart of the change management environment, but with a more distributed approach to gathering configuration data.

CA is another giant of the system management world that is changing its approach and product strategy to reflect the business requirements of service-based asset discovery. Following many years of acquisition and product integration, CA shows particular strength in the programming environment. Here its AllFusion change/configuration tools are well positioned to support the tracking of code objects through concurrent and parallel development. CA has a real opportunity here to extend its code-tracking technology into the SOA arena.

As usual, IBM has multiple solutions in this area, and its change and discovery strategy spans multiple tool-sets spread between Rational, Tivoli and new acquisitions such as asset management specialist Isogon. However, as shown in Figure 2.2, IBM is using an SOA Governance message at the core of its systems management strategy: asset discovery and change/configuration management play an important role within this environment. There is also a discernable shift:

- **away from the hardware/OS view of the world embodied in Tivoli**
- **towards the programming/service-orientation of the Rational products.**

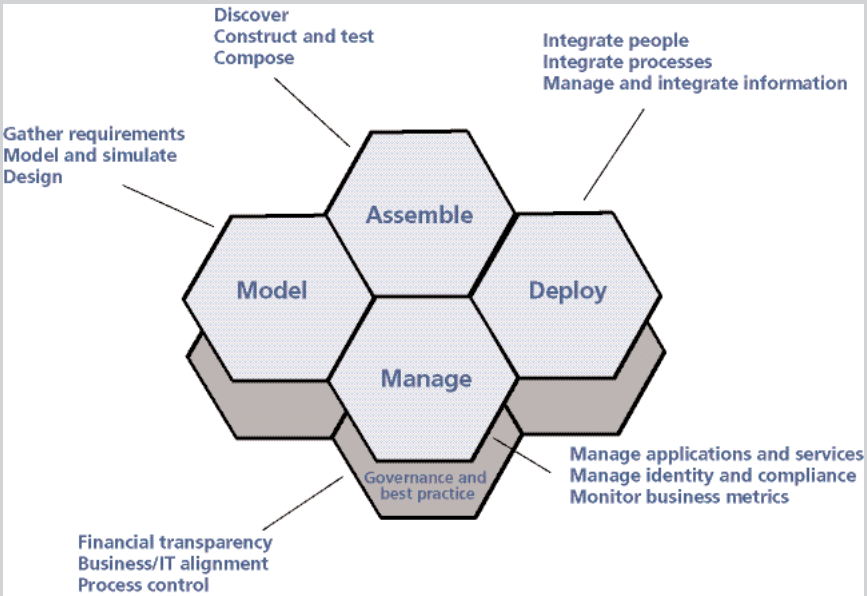
Management conclusion

Clearly there is a fundamental change going on in large organizations as business users seek a more flexible way to apply IT services, and data centers and call centers try to support them. There is a need for both legacy applications and the new world of services to be managed in a more consistent fashion. Much of this has to be achieved through automation in middleware which is just beginning to evolve.

Change/configuration and discovery services are a critical element of this change, and in the months and years ahead IT will see composite configuration databases emerging which reflect the state of hardware/software assets, application code and Web Services in a much more consistent way than is possible right now. Once vendors reach that stage, IT departments will be in a much stronger position to implement the more challenging aspects of service management. It is at this point that they can truly begin to talk to their business users in a language that they (the business users) understand.

Such a desired state of affairs starts with discovery. It is made possible when discovery is not treated as a one-off exercise but an ongoing work which provides the base for many complementary capabilities.

Figure 2.2: IBM and SOA Governance



Asynchronous JavaScript and XML (Ajax)

Tom Welsh
Consultant

Management Introduction

Ajax has become a topic of widespread discussion in the world of developers, although many decision-makers may not yet have heard of it. That is hardly surprising, as the name itself was coined just one year ago. Nevertheless, it is already having a profound influence on ideas about the design of Web applications, and millions of users are feeling its benefits every time they enter Google Maps, Google Groups, Gmail, and other popular applications.

As Tom Welsh explores, the key questions are:

- ***what impact will Ajax have?***
- ***how much does it matter?***

What is Ajax?

New software technology arrives thick and fast these days, and it can be hard to distinguish the wheat from the chaff. Ajax is a fine example of this syndrome, for it is by no means universally recognized, even in the IT industry. And, even among those who have heard of it, not everyone agrees about its value. There is no shortage of information — or hype, as sceptics would say.

A search of Amazon's computer section reveals 28 books about Ajax. AjaxWorld Magazine and Ajax Developer's Journal are full of news about events like the West Coast Real-World Ajax Seminar, due to take place on April 24th in San Jose.

David Heinemeier Hansson, a Danish programmer who enthuses about Ajax, has been described by Wired as "The Hottest Hacker on Earth". Jesse James Garrett, who coined the name Ajax less than a year ago, has become one of the blogosphere's most fashionable gurus.

Ajax has built up a remarkable developer base, spreading across the world like wildfire and driving a growing number of high-profile applications. Google was the first big adopter, using Ajax in Gmail, Google Groups, Google Maps and Google Suggest. Apart from its undoubted technical merits, one reason for this phenomenon is that — like Java and the Web — it is not a proprietary technology controlled by a single vendor. As in the early days of the Web, all the basic essentials are freely available so that new adopters can begin experimenting without significant expense.

Yet the leading vendors have not yet finalized their Ajax strategies. Indeed, some of them do not even have Ajax strategies:

- **Microsoft claims that it worked out most of the underlying technology, but its Atlas toolkit is still in pre-beta form**
- **IBM does not have a product, but supports OpenLaszlo and is leading the Open Ajax initiative**
- **Tibco, alone among middleware specialists, offers a high-profile product suite**
- **then there is a collection of heterogeneous Ajax toolkits and frameworks — over 70 according to one industry expert.**

Compromising between the 'rich client' and the 'Web client'

The fundamental idea behind Ajax is to strike a practical compromise between the 'rich client' and 'Web client' models. This is an admirable objective.

The rich client (today's more respectful name for what used to be called, dismissively, the 'fat client'), is a PC or similar workstation with one or more client applications installed. (Microsoft has its own variation on the 'rich client' theme, which it calls the 'smart client'). The rich client's drawbacks include the need for a more powerful PC with more storage, greater administrative overhead and security risks and sheer intrinsic complexity. On the other hand, it gives the developer a rich set of graphical and computational resources plus offers the end user a fast response time.

In contrast, the Web client (see the left hand side of Figure 3.1) is simpler and cheaper. Instead of applications residing on the client's hard disk, they are concentrated on one or more servers. Users access them through a Web browser, by going to a Web page or portal. As well as simplicity and lower cost, this set-up has the advantages of being more or less platform-neutral and easier to learn.

The downside is that Web applications tend to be slower than local ones, and the HTML user interface is quite limited. Every time the user clicks on a link or completes a form, the browser has to contact the server and request a whole new HTML page. As we all notice from time to time, it can take quite a while for some pages to download and then paint the screen. This is especially annoying when we have made a very minor change, such as ticking a check box.

The right hand side of Figure 3.1 shows how Ajax modifies the Web client to behave more like a rich client in some respects. When the user initiates a session — by clicking on the given link — the server application sends an Ajax engine instead of an HTML page. The engine, usually written in JavaScript, then runs continuously on the client, intercepting and handling all user requests. So, when we tick our check box, instead of mindlessly sending the change to the server and waiting for a whole new page to come back, the browser invokes the Ajax engine, which quickly and efficiently updates the screen — just as a rich client application would.

The Ajax engine splits the dialogue between the user and the Web Server into two halves, as illustrated in Figure 3.2. Now the user talks to the Ajax engine, and the Ajax engine talks to the server — but only as and when it needs to do this.

This is like the difference between communicating directly with a car manufacturer, and going through a local dealer.

Most of the time, the dealer can answer queries, fix technical problems and even sell products without referring back to the manufacturer. Above all, the dealer's interaction with the manufacturer is conducted asynchronously to its customer dialogues. The parts manager sets up a regular monthly order to keep stocks topped up; so when a customer comes in and asks for a given spare part, it is available immediately. Of course, it is critically important that the parts manager's reordering algorithm is well chosen — and this consideration applies to Ajax programming.

Ajax user interfaces obviate the need to rely on browser plug-ins for Java, Macromedia Flash or even Microsoft's ActiveX. All self-respecting browsers support JavaScript, and nowadays most of them can handle XML too. That is really all that is needed to make a start.

However Ajax tools are not necessarily free of charge, like — for example — Firefox or the Apache HTTP Server. High-end product runtime licenses command substantial prices:

- **Tibco's General Interface Framework, for instance, starts at $25,000**
- **JackBe NQ Suite 4.0 starts at $50,000.**

The Garrett and Obasanjo views

Garrett, who has been dubbed 'the father of Ajax' since he publicised the name in February 2005 (www.adaptivepath.com/publications/essays/archives/000385.php), makes clear that "Ajax isn't a technology. It's really several technologies, each flourishing in its own right, coming together in powerful new ways".

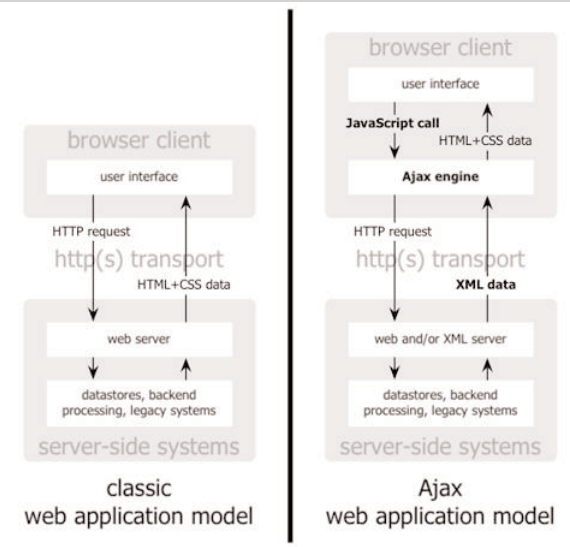
According to Garret, Ajax incorporates:

- **standards-based presentation using XHTML and CSS**
- **dynamic display and interaction using the Document Object Model**
- **data interchange and manipulation using XML and XSLT**
- **asynchronous data retrieval using XMLHttpRequest**
- **JavaScript binding everything together.**

Adaptive Path confirms that it does not own the Ajax concept, or sell any related software. When asked why he coined the catchy name 'Ajax', Garrett replied that he needed something shorter than "Asynchronous JavaScript+CSS+DOM+XMLHttpRequest" to use when discussing this approach with clients. The act of naming the approach contributed to its rapidly increasing popularity — if only by making it possible to refer to it at all.

Like Web Services (which do not necessarily use Web protocols, and are not really services), the name 'Ajax' sacrifices exactitude to snappy marketing. There is no such

Figure 3.1: Classic Ajax and Web application models



thing as 'asynchronous JavaScript' — the two are inherently orthogonal. In addition, Ajax tools need not use XML at all.

Dare Obasanjo, a well-known Microsoft developer, made the following blunt comments in his blog last year. "AJAX joins SOA (Service Oriented Architecture) in ignominy as yet another buzzword created by renaming existing technologies which becomes a way for some vendors to sell more products without doing anything new. What I find particularly disappointing about the AJAX hype is that it has little to do with the technology and more to do with the quality of developers building apps at Google."

There are five main components of Ajax. These will be examined, one by one, in a little more depth.

XHTML and CSS

XHTML is, essentially, a reformulation of HTML as an XML application. In other words, it is technically XML although it behaves more or less exactly like the equivalent HTML.

Webmasters who want their sites to be less prone to obsolescence, and more open to future improvements, should make sure that they are implemented in XHTML. IBM and the World Wide Web Consortium (W3C) have migrated to XHTML, although Microsoft, Sun and Tibco (among many others) still adhere to HTML.

Cascading Style Sheets (CSS) are a (relatively) simple mechanism for adding style (such as fonts, colours, and spacing) to Web documents. This separation of form from function enables the (X)HTML pages themselves to distil content, with no concern for presentation on the screen or page.

Document Object Model (DOM)

The DOM is a model in which the document or Web page contains objects, such as elements and hyperlinks, that can be manipulated in various ways. Some DOM objects may be active — for instance, JavaScript modules that perform functions when invoked.

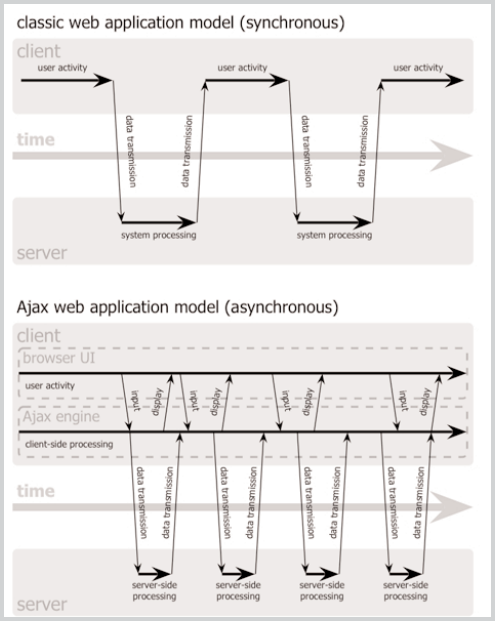
The combination of (X)HTML, CSS and DOM plus a scripting language like JavaScript, is often referred to as Dynamic HTML (DHTML for short).

XML and XSLT

Without XML, Ajax would be very different — although it would be possible, and there are some Ajax-like tools that manage without using XML. However, it is XML which gives the whole architecture a standard and consistent basis, as witness the various XML applications that vendors have evolved as part of their products. Ajax frameworks also generally use XML to exchange data.

XSLT (XSL Transformations) is a standard language for

Figure 3.2: Web client (synchronous) vs. Ajax (asynchronous)



transforming one XML document into another. This is important where there are different schemas or formats in XML documents.

XMLHttpRequest

Behind the formidable name, XMLHttpRequest is a simple enough piece of software. This is, however, key to the Ajax formula because it drives the asynchronous communication between client and Web server.

Originally written by Microsoft for Internet Explorer, this is now supported by several other browsers.

JavaScript

JavaScript — which bears little resemblance to Java — was invented by Sun and Netscape in 1995. It was standardised by ECMA — as ECMAscript (ECMA-262) in June 1997.

Strictly speaking, languages like JavaScript — and Microsoft's JScript — are implementations of the ECMAscript standard. But many developers still call the language itself JavaScript.

The most important thing about JavaScript is that it is a scripting language which is built into most of today's browsers. It is, thus, a portable programming tool.

Unlike Java, it is not object-oriented. It does have various technical shortcomings — but none of these have greatly harmed its popularity so far.

The Ajax tools market

Even if many software vendors do not yet have Ajax products to offer, they have plenty to say on the subject. Press releases are flying, and in large numbers. For example:

- **BEA is putting Ajax into its portal and it is adding it to its application server**
- **Cape Clear and ClearNova are working on Ajax tools that are expected to debut sometime this year 2006**
- **Information Builders is to build the next release of its WebFocus business intelligence product on Bwindows, an Ajax framework**
- **even Adobe has announced FABridge ('Flash — Ajax Bridge') which enables JavaScript to invoke Flash objects**
- **although Microsoft is quickly trying to rearrange its marketing messages to make room for Ajax, the truth is that it has several disparate initiatives without much coordination between them: the best known is Atlas, an Ajax framework — which was announced in June 2005 and whose relationship with ASP.NET 2.0 and Windows Presentation Foundation (WPF) remains unclear**
- **Tibco has made a splash with the announcement of its General Interface, which comprises a development environment (Builder) and a runtime (Framework); to deliver this it acquired the software when it bought the company of the same name in October 2004**
- **Sun has added Ajax support to the latest version of its Java Studio Creator.**

Other well-known Ajax products are marketed by:

- **JackBe (Figure 3.3)**
- **Backbase (Figure 3.4)**
- **Zimbra.**

There are also several Ajax open source projects, including:

- **Kabuki (currently incubating at the Apache Foundation)**
- **OpenLaszlo from IBM's partner Laszlo Systems**
- **Open Ajax, which was recently set up by a group of companies led by IBM.**

Also committed to Ajax open source initiatives are:

- **the Apache Foundation**
- **BEA**
- **Borland**
- **the Dojo Foundation**
- **the Eclipse Foundation**
- **Google**
- **Mozilla**
- **Novell**
- **Openwave**
- **Oracle**
- **Red Hat**
- **Yahoo!**
- **Zend**
- **Zimbra.**

So far, however, Tibco has stood aloof from open source invovment. Possibly it sees itself as having too much to lose as the market leader.

Cross platform considerations

The 'Musings from Mars' blog has an interesting analysis about the importance of cross-platform Ajax support, together with an up-to-date assessment of Ajax toolkits on a scale of A to E (www.musingsfrommars.org/2006/03/ajax-dhtml-library-scorecard.html). The following toolkits score an 'A, meaning that they work on Internet Explorer, Firefox, Safari, Opera, and all other DOM-compliant browsers:

- **DHTML Kitchen**
- **Dojo Toolkit**
- **DynAPI**
- **How To Create**
- **Javascript/Ajax Toolbox**
- **jQuery**
- **Moo.fx**
- **Open Cube**
- **Prototype**
- **Rico**
- **Sardalya**
- **Script.aculo.us**
- **Tacos**
- **Todd Ditchendorf's DHTML Gallery**
- **Turbo Widgets**
- **TwinHelix**
- **Walter Zorn**
- **Wicket**
- **X Library**
- **Yahoo! User Interface Library.**

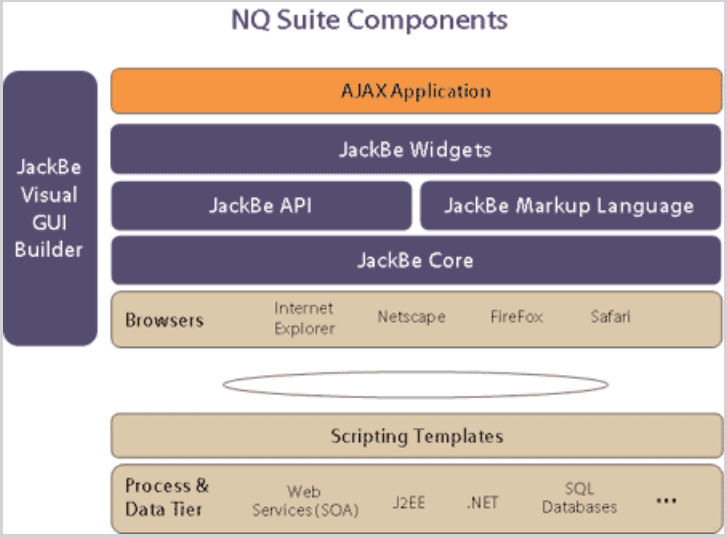
It is interesting to note that Microsoft's Atlas and Tibco General Interface receive E grades. This means that they do not work with any browsers other than Internet Explorer.

'I wouldn't start from here if I were you'

As the well-known (but apparently anonymous) saying has it, 'every large system that works started as a small system that worked'. The software industry is forever thinking up new ideas of varying degrees of originality. Few of these become instant successes. More often they are partially successful, or are discarded to languish in the 'tried and found wanting' pile. Future innovators, in their turn, are apt to pick up ideas from this pile and try them in new contexts — sometimes with astonishing success.

Consider the World Wide Web — which has become so familiar that we now just call it 'the Web' (or even, less reverently, 'the web'). When Tim Berners-Lee published the fundamental specifications of HTML and HTTP, just in time for Christmas 1990, he almost certainly cannot have realized that he had made the next big breakthrough in software technology. He was just trying to solve the problems of some researchers at CERN and elsewhere, who wanted to be able to read each others' papers online and, ideally, follow up references in real time.

Figure 3.3: JacbBe Ajax



It turned out, however, that the 'World Wide' part of Berners-Lee's chosen name was the most important. The Web gave a human face to the Internet. Within a few years he had made it available to millions of ordinary people all round the world. What it was became secondary to the overwhelming fact of its sheer existence.

Once the Web had become ubiquitous and universally familiar — which took only a few years — innovators began trying to build their own ideas on top of it. Sometimes the Web was a good fit to their entirely different requirements; sometimes less so.

As we have seen, for instance, many designers were impatient with the rigid principle that the client, not the server, should be in control of presentation. If the Web were to be used to deliver commercial messages, it needed multimedia impact: color, movement, and sound. But Berners-Lee's Web insisted that the server could only deliver information; it was not allowed to specify how that information should be displayed.

In part this explains all the baroque excursions with DHTML, scripting languages, Flash, ActiveX and Java applets. These were attempts by software designers to square the circle — by using the Web to do things that were quite incompatible with its original architecture. What they wanted was simple, understandable — and often illogical.

But there comes a point when so much effort (and money) has been invested in such attempts that they may actually succeed. If that happens the Web will be 'forked — splitting into two different parts: the original Web and the new 'Web'.

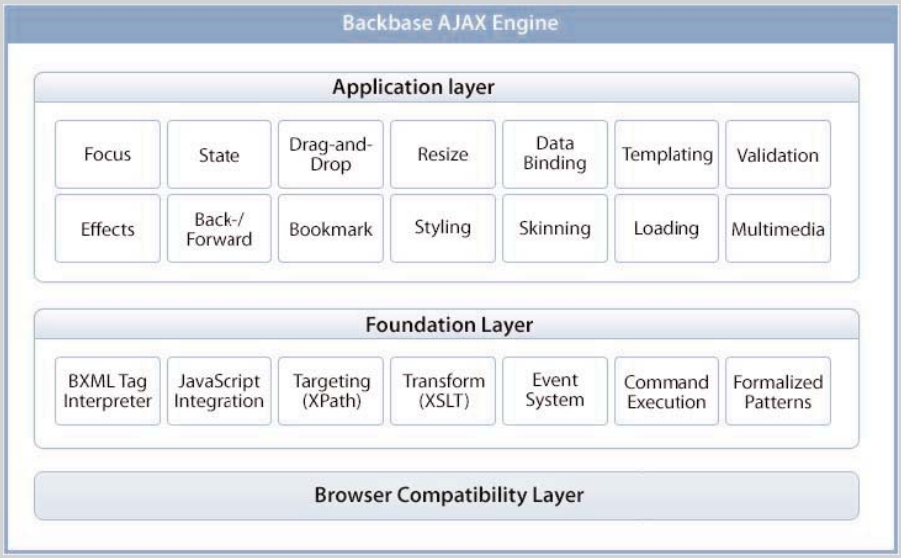
The significance of this digression about the Web is that this is what Ajax looks capable of achieving. It may 'break' the usual behavior that users expect when they hit their browser's 'Back' button because, instead of displaying the last page viewed, an Ajax application will probably do something else. Furthermore, bookmarks will cease to work in the way we are used to.

These are serious objections. But Ajax advocates feel that the sacrifice is worthwhile. As Garrett reflects, "...if we were designing the Web from scratch for applications, we wouldn't make users wait around" (for the next HTML page). It is hard to resist pointing out that (1) it wasn't 'we' who designed the Web, it was Berners-Lee; and (2) as already noted, the Web was specifically not designed for applications. You might equally well argue that 'if we were designing the BMW 1 Series for use as a submarine, we would have made it airtight and given it a periscope and a snorkel".

Management conclusion

Because Ajax is so much more efficient than plain vanilla

Figure 3.4: Ajax from Backbench



HTML when used for Web applications, it can greatly reduce average and worst-case response times when used skilfully. That may not seem to matter with today's fast networks, but latency is independent of network bandwidth. Moreover, because each Ajax client requires far fewer resources to provide a given response time, many more clients can be supported by a given hardware configuration.

Thanks to increasingly powerful automated development tools, it is getting quicker and easier to create sophisticated Ajax applications. Integration with Java and .NET is taking a little longer, but it will be complete within the next year or two. Already, products like Exadel's Visual Components Platform are showing the way.

That said, however, it is vital that developers remember that — in the words of Burton Group senior analyst Richard Monson-Haefel — "less is more" when it comes to using Ajax. Far from being a simple-minded panacea, it must be used judiciously. Otherwise users may become disaffected, and turn their backs on Ajax by the simple expedient of switching off JavaScript support in their browsers.

Companies like Backbase and Tibco are already up to version 3 of their Ajax products, while JackBe is at version 4,. Yet Microsoft is still working towards a beta of Atlas. Nobody — even at Microsoft itself — seems quite sure whether this matters, and if so how much. Is Atlas a flash in the pan, yet another of the IT industry's technical enthusiasms? Or could it be the start of something big? The latter seems more likely, although the technology is not fully-baked yet.

Re-use and SOA

Dr. Keith Jones
IBM Software Group

Management introduction

Re-use is once again a leading topic of the day. Critics of Service Oriented Architecture (SOA) are ready to declare failure because its biggest potential benefit — re-use — has not been realized and worse still, they say, it never will be. Is it really too early to make this judgment?

While thought leaders in industry and academia have puzzled over software re-use, a steady stream of 'push-down' enhancements to the middleware layer over time has resulted in considerably improved developer productivity. These enhancements have been 're-used' many times over as new generations of business logic have been deployed.

In this analysis, Keith Jones puts re-use — at the service level — into perspective and asks: what can be done to maximize its potential? A disciplined approach to design and implementation of service interfaces within enterprise SOA scenarios is possibly the best technical solution to this age old problem.

Under the microscope

1968 is often cited as the year of origin for Software Engineering. At a landmark NATO conference, thought leaders of the day met to debate the 'software crisis' and the steps needed to develop high quality software in sufficiently large quantity to satisfy burgeoning commercial, governmental and academic requirements. That same crisis is still with us almost 40 years later in spite of the best efforts of the worldwide IT industry.

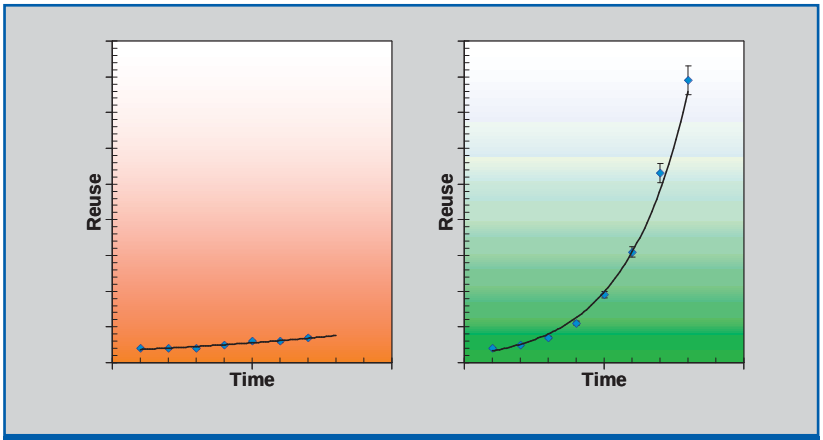


Figure 4.1: Re-use: reality and expectation

Approximately every five years since the software crisis was first identified there has been renewed focus on 're-use' as one of several techniques by which software can be produced more efficiently and software development costs reduced. Once again, re-use has become a hot topic in the industry. But this time the debate is closely aligned with the availability of early results from Service Oriented Architecture (SOA) deployment projects.

Critics of SOA are ready to declare that this latest approach to software development does not deliver the re-use that is widely expected. Yet the results that are emerging do show considerable promise, even though for most projects the timing of any judgment is surely premature. SOA does provide a mechanism by which software components are being re-used to deliver high quality systems in support of commercial and governmental goals.

For most, the expectation that comes with re-use is that existing software components can be deployed to satisfy business requirements instead of developing and deploying new components for that purpose. By avoiding the cost of building new components the total cost of software development can be held down and productivity can be seen to rise.

Unfortunately the methodologies that are normally used to build software systems do not always produce inherently re-usable components. Academic and industrial studies over many years have shown — repeatedly — that development methods must change if re-use is to become a significant factor in software economics. Without such change, the reality is that only ad hoc re-use can be achieved with the best intentions in a development organization against an expectation that is out of reach (Figure 4.1).

The question is whether or not SOA methods and technology can satisfy the expectation that comes with re-use. Since every service should provide re-usable function and services can be composed into high-order business logic, the promise is tangible. But as SOA negotiates the trough of disillusionment many observers in the industry have their doubts.

Re-use is everywhere and nowhere

The search for re-use has changed since 1968. In the days when commercial business logic was built using Assembler language, compiler technology was the focus of software engineers. Re-use of code fragments written in Assembler — for such functions as calculating tax and maintaining general ledger files — was limited only by the hardware being used and the operating system that provided the software execution environment.

When high-level languages (HLLs) became the preferred technology for building structured business logic, many of the algorithms that had been developed using assembler language were automatically generated to support new business logic. Re-use of those algorithms was widespread throughout business systems written in COBOL, PL/I and other HLLs at that time. Many are still running today. Many users have also been successful in building libraries of re-usable business logic routines in those languages.

Avoiding the cost of developing low-level software written in assembler is no longer considered to be a significant indicator of re-use even though the benefit has been real for a large number of enterprises. This style of re-use is everywhere in the world of IT systems.

The successor technologies that followed structured programming, high-level languages and libraries of common logic were Application Generators and Object Programming. These focused on capturing abstractions of business

procedures and business entities and, like HLLs, are still in common use today.

Application Generators have been extremely successful at delivering re-usable business logic within the scope of their design. However, their success has been constrained over the years by that very scope which was focused on certain types of general purpose business system.

Object Programming provided a level of abstraction that was lower than Application Generators (Figure 4.2) but has proven to be more powerful because it can accommodate a much wide gamut of business requirements. Object class libraries and frameworks have filled the gap between Application Generators and high-level languages in the abstraction spectrum and may yet deliver considerable benefit as a re-use technology.

However, as Object Programming has evolved to produce re-usable software, component technologies and re-usable abstract design patterns have come to the fore. The industry has moved away from centralized vertical 'stove-pipe' application systems to more horizontal distributed application systems:

- **first there were 2-tier client/server systems**
- **then came 3-tier business systems**
- **now we have n-tier, Web applications that pervade current IT systems.**

During the course of this evolution in programming and systems technology, middleware has fulfilled an important role. Over the years, the expanding middleware layer (Figure 4.3) has absorbed a significant number of application functions to serve basically two purposes, to:

- **simplify application logic and improve programmer productivity**
- **enrich, and make more expressive, the HLL and object languages used.**

For example, database systems like Oracle and DB2 have provided sophisticated information storage and retrieval logic that application programmers re-use on a daily basis. The logic provided is accessible through APIs that are integrated into the current generation of HLL and object languages — such as COBOL, C++ and Java.

A similar but less obvious migration of function from the middleware layer into underlying operating system and hardware layers has also occurred in current IT systems. Over time, this has provided a relatively coherent foundation for the middleware components and distributed n-tier applications on heterogeneous platforms as diverse as Windows, AIX, Linux, AS/400 and z/OS (for IBM's zSeries mainframes).

This 'push-down' migration of key application logic into the middleware layer (and supporting functions into the operating systems and hardware layer) has been a major source of benefit attributable to re-use — even though it is not often counted as such. Instead the industry continues to focus on application component re-use.

The service approach

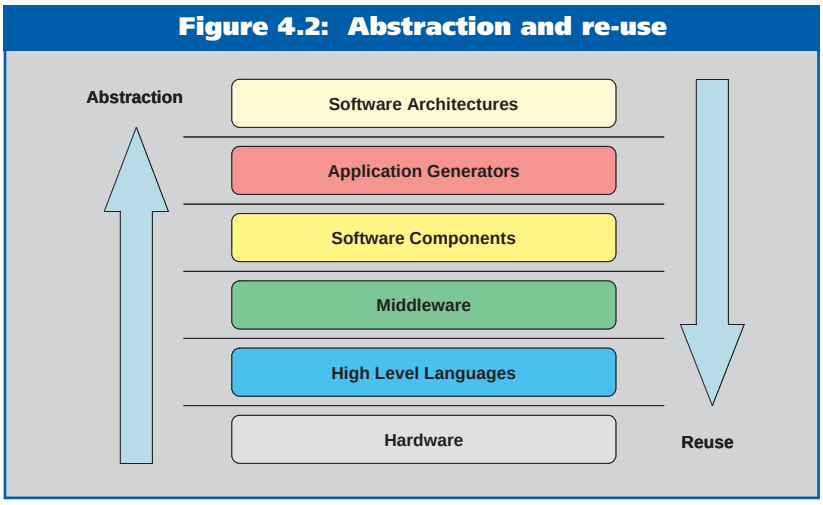
The most recent advance in thinking about re-use has come with Service Oriented Architecture (SOA). The ideas behind SOA are straightforward.

Each business process is implemented using business logic services. Some services invoke other services. But each service is assumed to be potentially re-usable. This implies that first-order service re-use may occur — at the touching points between business processes.

For example, CRM processes are designed to manage customers and every aspect of their relationships with an enterprise. ERP processes are designed to manage products, orders, financing and warehousing. When an order is placed for a customer there is interaction between the many services which deliver the CRM and ERP processes (and, subsequently, the SCM processes also).

This interaction (Figure 4.4) is the essence

Figure 4.2: Abstraction and re-use



of re-use at the service level. Its attraction is that it provides significant value to enterprises as they avoid the duplicate cost of business logic development. It is, in addition, the essence of integration between business processes and improved time-to-market — in the broadest sense — to give an enterprise degrees of agility.

There is second-order service re-use — as business processes are implemented using services that invoke other services. In general, lower level services in an invocation hierarchy are designed to manage the state of critical business entities such as [customer], [order] and [product]. These are sometimes called the 'create, read, update and delete' services — because they provide basic operations of the business entity. It is important that this state be managed consistently and, ideally, by a single unit of business logic.

Higher level services in the invocation hierarchy are designed to manage the state of relationships between business entities such as [customer] promises-to-buy [order], [order] commits-delivery-of [product] and so on. It is less likely that services at the higher levels of the invocation hierarchy can be re-used — but some may be applicable across business processes.

The often stated objection to re-use at the service level is that the interface defined is not rich enough in capability or it is too specifically aligned with the requirements of a particular business process. The answer to this objection is to align as closely as possible the service capability with business entities and the relationships between them.

In the example of a low-level service that maintains the state of [customer] entities it is easy to see that, when the interface is maximized to contain all the operations needed to handle customer related information, the service becomes more generally re-usable. It is when the operations are incomplete or when information related to other business entities is mixed into the interface that inherent re-usability is diminished.

When a strongly cohesive interface has been designed with these principles in mind it is natural to co-locate the business logic needed to implement the service operations for deployment (Figure 4.5). This also enhances re-use as the business logic may well be implemented in several different places in current systems. The business logic implementation is often able to share resources (such as Java object components and databases) and share design patterns as an added bonus.

The proposition that services can be shared between

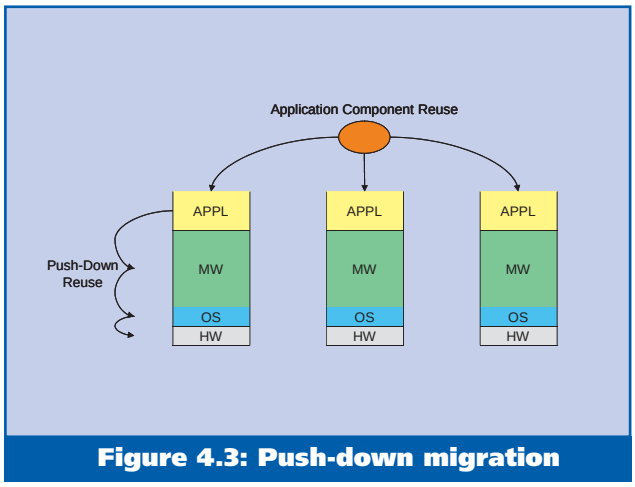


Figure 4.3: Push-down migration

Figure 4.4: Service re-use

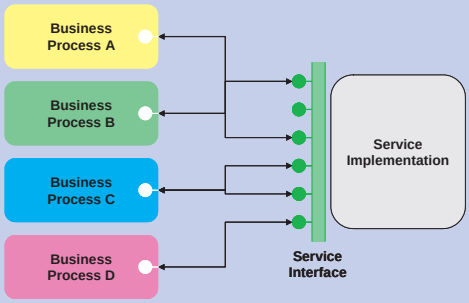
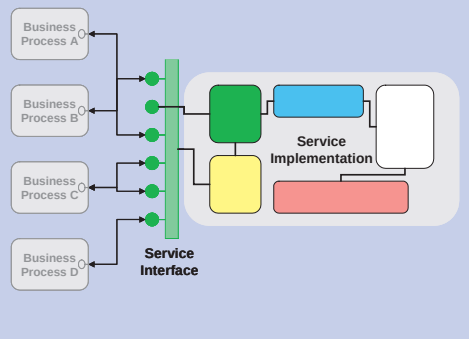


Figure 4.5: Interface cohesion



business processes is significantly enhanced when modeling has been used to demonstrate:

- **sharing of business entities**
- **the relationships between them.**

This aspect of the top-down approach to Service Oriented Architecture is an important enabler for service re-use. Unfortunately this value does not emerge until several business processes have been modeled, designed and implemented in a Service Oriented Architecture.

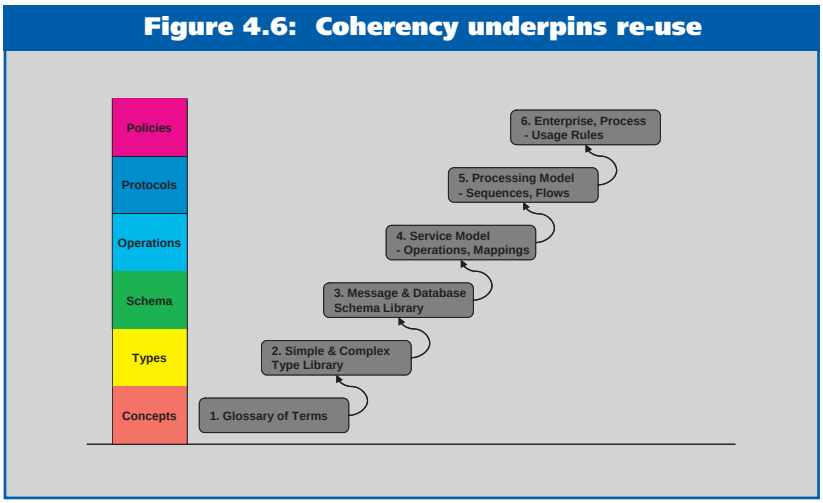
The bottom-up exposure of services in a Service Oriented Integration (SOI) scenario is another opportunity for re-use that may deliver great value to an enterprise. Analysis of interfaces between components in an SOI scenario can lead to the definition of a coherent set of services that become the platform for future business capability.

Many enterprises are already taking this route to implementing service architectures based on confidence that existing interfaces are proof of business value. However, care must be taken to ensure that the new service interfaces exposed are strongly cohesive, as described above. This may involve implementing a layer of infrastructure that maps service interfaces to one or many existing application systems in the integration scenario.

At the core

Whether the approach taken to re-use with services is top-down or bottom-up there are some fundamental technical elements which must be considered in order to be successful. There is a useful 'stack' (Figure 4.6) of such considerations for architects who are designing re-usable services.

Figure 4.6: Coherency underpins re-use



Each layer suggests an extra dimension of coherency that will tend to facilitate re-use.

Re-usable services must be based on a set of concepts and relationships at the most fundamental layer. Concepts that underlie business processes — such as [customer] — must have meanings that are shared by business and IT professionals involved in the design of re-usable services. Lack of this shared understanding is often the most significant impediment to re-use in an SOA.

A well developed glossary is the first step to achieving this level of coherency but process discipline is needed to ensure its consistent use in an SOA deployment. Only those concepts that are needed to implement active business goals at an enterprise should be included or this alone will become a mammoth undertaking.

At the next layer a shared definition of types is considered to be fundamental to achieving significant service re-use. The 44 fundamental XML Schema types are often used as the starter set but these must be composed into simple business types — such as [address] — and yet more complex types that can represent business entities — such as [customers], [products] and [orders]. The value of having a shared standard definition of types is that re-usable service interfaces can be more readily deployed in a heterogeneous service infrastructure.

The simple and complex types defined to represent the business concepts and their relationships are used to build a number of schema at the next layer in the stack. These schema are used to represent message payloads, object structures and database views in an SOA. Each message schema might contain business entity types together with relationship type information to be used in re-usable service operations.

At the next layer, the business operations that are incorporated into re-usable service interfaces are defined using the schema and types from the lower layers. Interfaces are inherently more re-usable when they are defined in this way — and when restricted to containing operations on single, or multiple, directly related underlying concepts.

Every operation in an SOA must have a protocol associated with its use. The next layer in the stack provides a number of defined protocols that detail how an operation is intended to be used. Most often

the synchronous request/response protocol is deployed — but increasingly service operations are being implemented using asynchronous protocols and with more complex flows. Without detailed knowledge of the protocol required, a service cannot easily be re-used.

At the highest layer in the stack a number of policies are defined. These provide the pre-conditions, invariants and post-conditions that must be satisfied in order for a service to be invoked successfully. Such policies include predicates for:

- **security**
- **transactional integrity**
- **response time**
- **reliability**
- **availability**
- **and other non-functional requirements that must be met when a service operation is invoked.**

By working 'up' the layers of this stack from concepts through operations to policies it should be possible to define a coherent set of re-usable service interfaces for deployment in an SOA. Failure to apply appropriate focus to the details outlined in the stack will almost certainly result in services that, even at their best, provide limited re-use.

Management conclusion

The search for re-usable software components has been underway for almost four decades. At each turn on the technology trail there is renewed hope that this time re-use will become a significant economic factor for enterprises who develop software for competitive advantage. SOA is the latest technology to promise the benefit of significant re-use potential.

During the same period of time the middleware layer has developed significantly. Much of the function that was at one time considered potentially re-usable at the application layer has now been absorbed into the middleware layer. Developer productivity has increased steadily whilst this migration of function occurred. Middleware must take some of the credit for this 're-use' advantage.

One of the most important benefits of SOA is the re-use that it promises. Service re-use occurs at the boundaries, between business processes and within those processes. Well designed service interfaces that are strongly typed and cohesive provide the basis for that re-use. Using an organized approach to design of service interfaces it is possible to maximize service re-use potential. As always, re-use is not just a matter of design and technology. It requires organizational governance, disciplined processes and a nurturing culture to establish re-use as a way of doing business rather than an accidental by-product of de-facto methodology.

If SOA is the answer, what is the question?

John Perry
Technical Director, Alphacourt

Management introduction

Ideas, techniques and paradigms come into fashion in the IT industry on a regular basis. Some have a profound effect on how one designs applications, runs operations or codes programs. Some slip into oblivion.

John Perry of Alphacourt started his computing career as an assembler programmer on IBM 360 mainframes. Since then he has seen many of these practices come (and go) — only to be superseded by the next. In this analysis he takes a practical, customer-focused view of whether SOA (Service Orientated Architecture) is the answer — using the WebSphere middleware product portfolio as a reference point.

The answer?

It would be arrogant to think that 'SOA is THE answer'. It will, without doubt, be supplanted by the next 'answer'. However, I believe that SOA is a necessary movement towards a better, more responsive IT world.

As such SOA is an answer not to problems within the IT industry, but an answer to the problems of the IT industry. What is the job function of an IT developer in 'Allied Widgets' — a medium sized manufacturing company? If you were to ask a person employed there, he or she would probably say something along the lines of:

- **"write good Java code"**
- **"ensure data integrity"**
- **"increase the throughput of such and such an application"**
- **etc.**

To me a better description would be that their job should be to help 'Allied Widgets' sell more widgets. The reality is that, in IT, this business focus has — to a larger extent than IT wishes to admit — been lost.

As IT systems and departments have grown over the last 40 years, they have evolved from being the solution to the problem to becoming the problem itself. IT departments now tell the business what is possible rather than implementing what the business wants. In fact, IT should be a service industry servicing the needs of the business that it supports (a position that IT ignores at its peril). This is the fundamental challenge that SOA addresses, and should address.

Building blocks

The basic building block of a Service Orientated Architecture is... the 'service' (this should not be a surprise). But what is a service?

There are many rigorous definitions of what constitutes a 'service'. As I do not want to add to these, let me say that, at it simplest, a service is a small atomic (in the sense of 'indivisible' rather than 'glow in the dark') piece of business logic. So, for example:

- **"get stock on hand" may be a service in a manufacturing/retail business**
- **"credit to account" is a service for banking.**

Such low level services can be aggregated to make composite services (which are, themselves, services). Thus the "Get Account Balance", "Debit Account" and "Credit Account" services may be aggregated into a (relatively simple) "Transfer Funds" service in the banking industry. This "Transfer Funds" service can be itself be aggregated into larger services ... and so on. (Just as an aside, these services should have no presentation logic within them — for this greatly hinders both re-use or aggregation of services).

As the lower level services are identified (and note that the identification is from a business perspective, not an IT one), the development of a new composite service should, to a large extent, be 'merely a matter of gluing' lower level services together — with some business logic throw in. In theory, after the initial pain of developing these services, the IT department will have become far more responsive to business needs and changes, as well as more agile.

Back to IT providing a service

If SOA is the answer, what is the question? The question is "How do I make my IT systems more responsive to the business?" Or, to put it another way: "how do I get back to the position where IT is the service industry providing IT services to the business for which I work?"

SOA is not new. A number of people have tried, with varying degrees of success, to implement their own home grown SOA using (for example) IBM WebSphere products, like the WebSphere Business Integration Message Broker" (WBIMB) and WebSphere MQ (WMQ). What were the main challenges?

When IBM first released its original Message Broker (MQ System Integrator — or MQSI 1.0), it was a relatively simple product that could:

- **read messages**
- **transform them according to graphically-created rules**
- **forward those messages to other queues.**

When IBM released MQSI 2.0, it was a completely new product with the concept of a 'compute node' with an accompanying programming language called ESQL. This was designed to facilitate more complex data transformations to be done programmatically. Database access was included to enable messages to be "enriched" by look-up in a database.

However, as ESQL is a complete programming language, it could be used for far more. At Alphacourt we have seen complete application systems written in ESQL, ones that only a few years ago would have been written in COBOL. Yet, ironically, this is not what WBIMB (or its antecedent

MQSI) were designed for (or were particularly good at). Many 'home grown' SOA implementations have used Message Broker to hold business rules, 'state' as well as to provide the glue for an SOA implementation. This has meant that some of the message flows are huge — and eye-wateringly complex.

At Alphacourt we have always maintained that the original Message Brokers should have always been used for their original purposes — essentially transformation and routing. This simple need, and the growth of Message Broker into other technologies (such as Web Services) has delivered the reality that now lies behind the concept of an ESB (Enterprise Service Bus). Indeed, even IBM itself sometimes refers to what is now WBIMB as an 'advanced ESB'.

But, where are the 'business rules' and 'state' to be held if not in a message broker? One solution, at least in the IBM world, lies with WebSphere Process Server. This is a product designed to provide glue and to mold (orchestrate) services into what becomes a business solution.

Correctly identifying services

One of the other problems, and not an insignificant one, that we have encountered with home grown SOA solutions concerns the correct identification of services — what constitutes a 'service'? As an SOA is predicated upon a paradigm where the IT solution models the business, the issue becomes — how can one implement an SOA without an understanding of how the business works?

This is the job of WebSphere Business Modeler. This is one of a variety of tools which help one understand the business and provide the bed-rock upon which to build an SOA. My point is — get this part right and the rest will flow naturally; get this wrong and the rest of a project will be all up-hill. In effect, the fundamental question is: how does the business work?

No one solution

The truth is that, within the WebSphere portfolio (and I believe that this is largely true of other vendors), there is no single product one can point to and say 'this is the implementation for an SOA'. Instead, there is a raft of product choices which, taken together, are able to enable an SOA solution to be introduced.

To flesh this out, there is the business being modeled using WebSphere Business Modeler. This identifies the services and how they interact in the business world. The processes identified by Modeler are then 'glued' together using WebSphere Process Server — where the business rules and state are encoded. Then an ESB (either WebSphere Enterprise Service Bus or WebSphere Message Broker) are used to provide a standard pipeline to these services (the pipeline provides mediation services).

Now you possess an SOA — yes? Well, no: not really.

There is a final step to take. Having modeled the business, identified the services, glued them together and provided connectivity, you also need to monitor how the business processes are working. I am not talking here about monitoring database usage or message queues (there are plenty of tools on the market that can monitor these low-level, technologies). Rather, what we need to monitor is how the end to end business processes are working. The WebSphere product that implements this is WebSphere Business Monitor.

Its importance is considerable. It may well be that, during monitoring, a bottleneck is identified in the business process that, if modeled slightly differently, would provide improved performance. WebSphere Business Monitor enables you to go back and make a slight change to the business model that that can then be exported to the process server for implementation. With this one now possesses an iterative, and improvable, process.

Governance

One aspect that I cannot be emphasized too much is the need for governance in the implementation of an SOA. One of the problems that I experienced in the object oriented programming world (C++) is that, rather than using inheritance to derive a new class, developers tend to want to start from scratch. This leads to a huge collection of very similar classes rather than an elegant OO hierarchical structure.

The same issue occurs in the SOA world. Rather than try to re-use an existing service, developers are prone to developing a new one. This leads to 'service bloat' — which is the death knell of a clean SOA implementation.

Of course, if the need for the service is based upon the business, then if the business has a new service, the SOA should have one. But the desire of architects and/or developers to take an expedient solution rather than the correct one needs to be curtailed. Someone needs to keep a view of what is being developed and enforce SOA compliance.

In this context, perhaps one of the most important facets to keep a track of is what services already exist. A library of

existing services should be maintained and referred to whenever a new service is suggested, for it may well be that this new putative service is one which has already been implemented elsewhere. Unless a governance body stamps quickly and effectively on the new development, the rot has started.

Web Services and SOA

Where do Web Services fit into an SOA?. There appears to be one opinion that Web Services are an integral and necessary part of any SOA — that Web Services are integral to implementing an SOA, that, in fact, Web Services are SOA.

I disagree. This is not true. A 'service' is, more than anything, a design concept or a business concept. It should be possible to implement this using multiple different methods and technologies. That said, Web Services are certainly a viable implementation of an SOA.

The key point is that no-one should suggest that, if you already have a large investment in other technologies, these should be scrapped and re-written as Web Services. That is wasteful in the extreme (as well as organizationally impractical). The trick is to:

- **make the design right**
- **then use the tools provided (Process Server, ESB, etc.) to help with the implementation and re-use of existing components.**

Big bang threats

Perhaps the greatest threat to successfully implementing an SOA is the size of the first implementation. A 'big bang' approach is a surefire way to reach a bad end.

But here we have a dichotomy. A service is a piece of functionality used by the business, not by a small project within the business. Therefore, when implementing an SOA, should one not try to implement things at the entire business level rather than at a small project level? Well, Yes and No.

An initial implementation should be done at a small project level but, when trying to identify the services that this project will use, a business view needs to be taken. This again is where governance must be applied.

It is all too easy to identify a service for a small project, implement that service but then find that it is unusable in a wider business context. As the whole point of an SOA is the re-use of business services, you have to ensure that any services identified will be re-usable (but not that they will necessarily be re-used).

Management conclusion

Is SOA the answer? Mr Perry believes the answer is partially Yes. Is SOA something that organizations should be looking to implement? Here Mr Perry definitely believes Yes. Yet he also makes the point that an SOA is 'something' that should not be taken lightly, 'something' that should be undertaken slowly and carefully — with buy-in from both business people and IT.

With the business correctly modeled, with services correctly identified, with strong governance in place, an SOA implementation can make a significant improvement in the responsiveness of an IT department to changing how it supports the underlying business. But, without these necessary steps, an SOA is likely to cause more problems than it was seeking to solve.

SOA — implications for change

Mike Beeston
Maven Associates

Management introduction

Working in IT these days you cannot miss the attention lavished on Service Oriented Architecture (SOA). Each day a fresh avalanche of news items and press releases concerning SOA initiatives from software and services vendors is likely to appear in your inbox or news reader. It would appear that the jury has finished and has returned its judgement — unanimously agreeing that SOA is the only way forward for IT and business solutions.

This is surely good news. If everyone agrees, then surely we can proceed with confidence as we move to create our own organization's SOA and reap the touted benefits. What can stand in the way of such self-ardent 'rightness'?

As Mike Beeston examines, there is more required than just the commitment of the IT industry to produce technologies that enable an SOA. Undoubtedly the availability of standards (such as Web Services) and technologies (like those found in the J2EE and .NET application frameworks) that help with a move towards service orientation are a start. But there is more to service orientation than technology alone.

SOA and application integration

With so much attention on SOA it is not surprising to find it well understood in customer organizations that SOA is about considerably more than technology — it can have much influence on business structure and strategy. Some of the excitement around SOA indeed stems from its apparent promise to provide an answer, in part at least, to a long standing problem — that of business and IT alignment.

In the last decade or so we have seen the rise of application integration that has partially broken down business process barriers between organizations, operating units, departments and teams. For the undoubted benefits this has provided it has also placed the systems that have been integrated under the microscope.

Many years ago it was common to build an application to address a specific collection of related requirements in isolation from the broader context of the business in which they existed. This provided short term benefits for sure, but ultimately this application silo approach — with its application solutions aligned to organizational areas — restricted change and growth.

Application integration has introduced a level of relief to these constraints by opening up otherwise closed applications to interoperate across both technological and business divides. In doing so it has helped uncork a latent demand for increased business agility — enabled though technology-led innovations.

Application integration initiatives, by coercing otherwise closed applications to share their joint resources of processing (business logic) and data, have made a major contribution to the development of the SOA vision. It is now taken as read that application solutions underpin agile business processes, a process which can alter and flex as business priorities change.

Of course, the reason we embarked on application integration in the first place was that 'the business' was demanding that the artificial process barriers which isolated applications and inhibited process improvements be eliminated. Application integration provides a short term measure to reduce these barriers. But it is not a final solution.

SOA and an ESB

Application integration is far from the only technology development that has led to the widespread acceptance of the SOA concept. However its significance applies to the evolution of the Enterprise Service Bus (ESB) which owes much to those pioneering efforts of organizations, along with technology and service providers, that set about building an agile infrastructure from otherwise proprietary technologies.

This work — completed in the pursuit of application integration objectives — has turned out to provide the necessary understanding for building a base infrastructure that delivers capabilities like:

- **location**
- **time independent communications**
- **mediation between disparate data formats**
- **levels of process orchestration**
- **techniques to interface with applications old and new.**

The legacy of the 1990s application integration initiatives (Figure 5.1) can, therefore, be considered as:

- **the evolution of the ESB**
- **the development of an understanding of what is needed to build agile integration solutions**
- **the appreciation and shaping of the demand for technology led change**
- **contributing to the SOA vision of processes that alter and flex in response to changing business needs.**

Aside from the concept of the ESB, application integration efforts have also provided needed insights into the challenges of interfacing with applications. Challenges that had to be overcome here included:

- **limited or no access to application processing via standard interfaces**
- **limitations in accessing data directly (thus bypassing application logic)**
- **myriad conflicting data representations and quality concerns**
- **security, management, auditing and problem resolution requirements.**

Discussion of the evolution of the ESB and application interfacing techniques is clearly incomplete without consideration of Web Services. Suffice it to say (here) that Web Services provide a set of standards that help address a long standing challenge — that of exposing application functionality and data in a consistent manner. Importantly, through the application of standards, it is practical to use both development and runtime tools and technology significantly to increase agility.

But SOA is not equal to ESB and Web Services

A standards based ESB plus use of a Web Services approach does not by itself deliver an SOA. This is a point that many organizations either do not understand or choose not to understand in their pursuit of the latest industry hyperbole.

An ESB represents a technology approach that emerged from efforts to reduce complexity when implementing communications between otherwise disparate application components. Web Services origins date as far back as 1997: in 1998 SOAP was still just an acronym; WSDL (Web Service Definition Language, used to describe service location and operations) only began to exert influence as a standard from 2000 onwards with UDDI (Universal Description, Discovery and Integration protocol) standards submissions also occurring at this time.

Yes, ESB and Web Services are key technologies that can be used to ease implementation of an SOA. But the basic concepts underlying an SOA long predate these technological developments.

To underline this point, my first practical experience applying SOA design principles dates back to the early 1990s working with CICS development teams designing mainframe applications that embraced the design principle of a 'separation of concerns' — between presentation, logic and data management. The business logic could be successfully encapsulated and, in essence by utilizing the CICS proprietary facilities prevailing at that time, was exposed as a re-usable service.

With this clear, SOA is:

- **typically built in a manner that exploits standards (such as Web Services) and enabling technologies (like an ESB); proprietary techniques can be used in conjunction with standards without undermining SOA principles — although this does require the introduction of some form of risk management activity for control of possible reductions in future agility**
- **about evolutionary change to the application portfolio; where previous integration efforts solved specific integration requirements, SOA seeks to make incremental change to the underlying application — with the result being that application processing and data are exposed consistently and regardless of how they will actually be used**
- **concerned with business activities and not only technology; services are implemented in some sort of business context and can, for example, encapsulate a specific function (like, 'create purchase order') or a process (as an orchestration of services)**
- **a framework and design approach which promotes agility though combinations of service composition and re-use — to address ever changing business goals.**

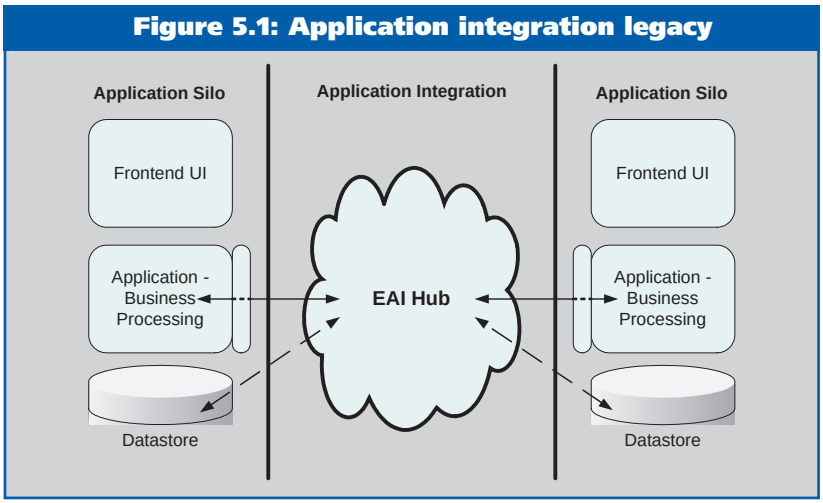
SOA — evolutionary change

SOA may not be entirely new. But it does present a new set of challenges to any organization that aims to benefit from it. It is in the detail, of how SOA is different from earlier methods, that the greatest new challenges are found.

Consider the ESB — already positioned as essentially an evolution of application integration efforts combined with industry standards and best practice. An ESB is capable of providing necessary functionality to an organization that needs to integrate applications.

Where possible application integration aims to implement straight through processing (STP) solutions between application components, with interfaces optimally operating at the business logic level and exploiting application vendor provided mechanisms to communicate between the ESB and an application. Increasingly vendors provide Web Service interfaces, which are a key part of the standards based language of any decent ESB.

Figure 5.1: Application integration legacy



What is achieved in this scenario is that a technology problem — application silos inhibiting cross business processes — is solved using contemporary techniques that have been delivered (embodied) in an ESB (Figure 5.2). But this is still application integration — not transformational service orientation — and business process change is only partially improved.

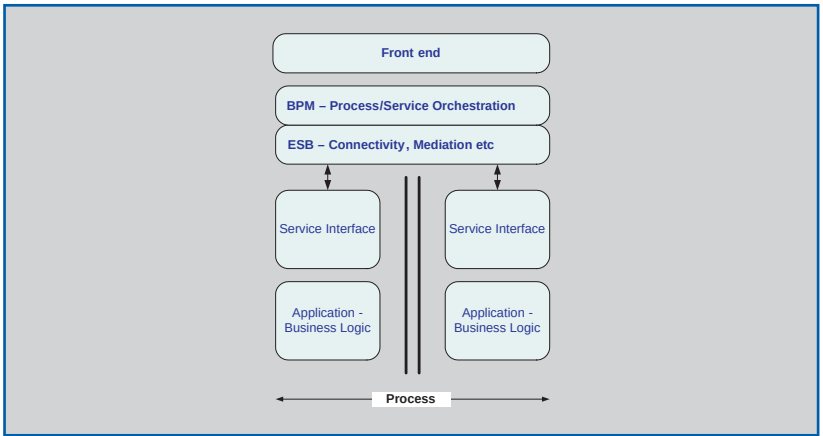


Figure 5.2: Traditional EAI silos

The underlying problem remains. The applications themselves are not sufficiently agile to participate fully in any complete SOA vision.

This aspect of the ongoing problem will take a long time to change. For example, consider that, despite the popularity of J2EE, CICS remains a successful proprietary application server due to the very large historical investment in solutions built with it.

To a large extent we are in the hands of the application providers to resolve this problem. An interesting example to explore is that of Oracle — the acquisitions of PeopleSoft and (thereby) JD Edwards combined with its existing E-Business suite gives Oracle a sizeable and ongoing re-engineering challenge. In order to protect its customers' investments in these solutions — and minimize disruption as changes are made to the underlying applications — Oracle embarked on Project Fusion (please note that this is quite distinct from Oracle's Fusion middleware products) which is designed to unify application solutions by aligning all to an SOA design.

The ESB technology stack can replace, to some extent, the application integration middleware that is its ancestor. Application vendor supplied Web Services, or perhaps even proprietary interfaces, provide at least a service oriented facade on top of legacy applications. This is progress towards an SOA vision (Figure 5.3).

What has not changed, and what needs to change

Yet, what has not changed is the way the applications are managed in an organization. A business operates many processes that cross departmental divides. Applications historically tend to have ownership within some organizational unit. Those owners tend to exhibit a protective view of their application asset.

Deconstructing application assets and then reconstituting them as 'business services' challenges this traditional organization structure. Adding a service facade to an otherwise monolithic application and publishing a set of service definitions for use introduces new considerations for the application owner.

The move to SOA must, therefore, be evolutionary rather than revolutionary. It is in part about introducing technology and standards to solve pressing concerns — much as application integration was. More significantly SOA is about changing how IT provides value. Initiatives such as integration were primarily about solving technical problems that hindered business change.

SOA, in its full glory, should be about changing the way IT solutions are implemented and consumed so that business activities can have the agility they require built into them.

SOA — new challenges

The objective of a SOA strategy is to ensure IT can optimally support business and in doing so drive process improvements as their requirements emerge. To do this requires adoption of SOA principles and seeing these embodied in appropriate enabling technologies — like in an ESB, in an existing application portfolio (for example, ERP suites) and in new technologies as these arise.

Many aspects of an SOA strategy are appropriately focused on technology, and will experience challenges similar to those faced in other technology initiatives — for example:

- **aligning new or updated SOA technologies to other IT initiatives (like server consolidation, rationalizing application portfolios, post merger and acquisition combinations, and so on); in this light SOA is an initiative that will so widely influence and ultimately contribute to**

the transformation of an organization that it must be managed at a strategic level

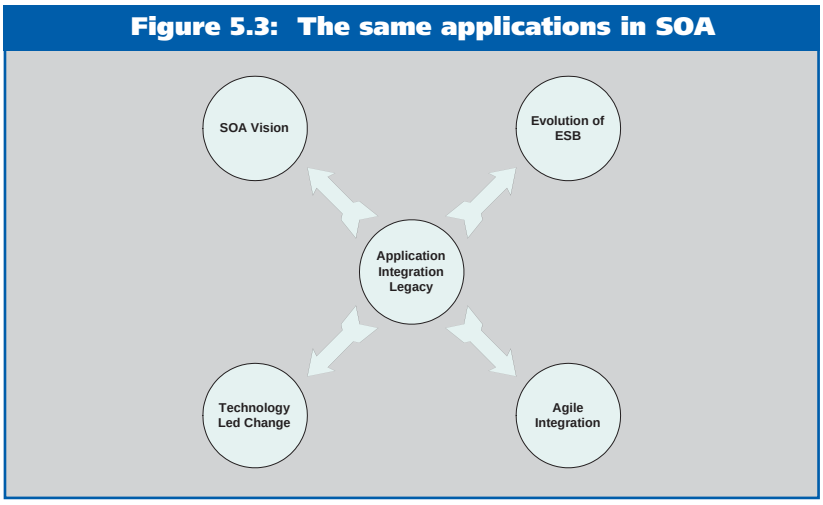
- **introducing and preserving new and updated skills and roles**
- **managing projects in portfolios, thereby achieving immediate term benefits as well as supporting the organization's long term vision.**

Those are the 'easy bits'. The more demanding aspects of SOA relate to delivering change through greater business alignment — where the 'business to IT divide' must be bridged.

Perhaps the major attraction of SOA is its promise to deliver an IT infrastructure that allows the business to respond to change and so gain some sort of advantage. Such change might be the opportunity to reduce cost by changing a supplier of some service (thus gaining a reduction in overall process cost); equally it might be to deliver a new service to the market ahead of the competition.

The implicit modifications to the traditional method of delivering of IT require considerable changes to the way in which organizations approach IT. Where, today, application solutions are aligned to departmental groups, in the world of SOA the vision demands cross departmental — or cross organization — processes built from services that satisfy the requirements of a business process step. Instead of focusing on optimization of the activities of a departmental unit — perhaps by exploiting the capabilities of a discrete application solution — efficiency is achieved through process refinement and service implementation. Rather than optimization based on organization structure, businesses should now be able to optimize based on process goals.

Figure 5.3: The same applications in SOA



Such a change requires more than alignment between the IT and business constituencies. Definition of business process can only be successful if the whole organization is fully engaged and conversant with the SOA principles regarding service design and orchestration. IT architects must develop greater understanding of the business environment in which their solutions will operate. Conversely, business people must become conversant with IT, service design and orchestration. In many respects it is likely to be the SOA architect who must bridges the gap between the two groups.

The ramifications of these considerations are substantial:

- **traditional approaches to IT delivery and management, through provision of applications alone, is insufficient: services become the primary delivery mchanism, albeit commonly implemented over an application solution**
- **when new solutions are required a more holistic business perspective (based on process and service use) should be applied, instead of considering deployment of another application to address the requirements (in line with the traditional organizational unit); ultimately the service implementation should not be a direct concern to the business constituency (of greater concern is its quality and service level attributes along with usage costs)**
- **services, being more granular than applications, are more easily measured in terms of usage and cost metrics; value, or not, can be more easily determined**
- **there is a huge opportunity for process simulation and refinement to be used during design; with more clearly defined service costs, it is viable to explore process changes and assess their impact using modeling tools**
- **the concept of out-sourcing services or perhaps dynamically selecting a service provider based on runtime requirements, adds new delivery and management possibilities.**

Services requireorchestration into processes

This discussion of SOA has, thus far, focused on the concept of a service and its access via an ESB. As a consequence the SOA has been viewed primarily from an

infrastructure perspective. But an SOA should be considerably more than an infrastructure capability alone.

Much of the value promised by an SOA is delivered though Business Process Management (BPM) capabilities. Without technology designed to manage the orchestration of services in support of business process objectives it is necessary to capture process management in custom code.

This is a constraint on agility. Instead of the problem being tied to applications used by organizational units, now it is associated with inflexible process implementations. Arguably this is an improvement of sorts as there is now a foundation — services — for more flexible change.

But, without BPM technology, the full value of the SOA vision will never fully be realized. Process implementations are limited in their ability to flex and respond to changing business goals — because their execution logic is captured in code somewhere.

A further consideration here is that, even with an SOA complete with BPM, the full promise of process agility may remain elusive for some time. This is due to the extent to which current process implementation are tied to application assets. Where parts of a business process are implemented in legacy systems application complexity can inhibit service enablement. The reality is that there is limited scope to deconstruct this process and reconstruct it using an SOA approach.

SOA — long-term change

Organizations face the continual challenge of achieving change through information technology innovation. As always, there are no silver bullets.

On the positive side, the prominence of SOA underlines growing maturity in:

- **technology — providing enterprise class foundations for IT solutions**
- **the use of wide spread architecture and design understanding**
- **an essential appreciation of how harmony between the seemingly conflicting worlds of business and IT provision might be realized.**

SOA is an important change enabler for development of agile enterprises that can deliver the vision of truly flexible business processes. The SOA 'nirvana' might be considered as being the state where business processes may be changed via BPM modifications without it being necessary to create or modify application based business logic.

Adopting SOA does not imply wholesale change to the application portfolio or IT infrastructure. SOA is not an unchanging state to be implemented in some manner. Rather it is an ongoing evolution of IT and business operating in growing accord — delivering increasingly cost effective IT and competitive advantage.

Management conclusion

SOA may not be entirely new. But, arguably, understanding it has now reached a maturity level that justifies the attention it receives.

Nevertheless it is important to be aware that introducing an organization-specific SOA is a long-term course of action. It is a course that requires changes across the organization in both IT and business constituencies and, ultimately, results in an overhaul of both application solutions and business processes.

To realize the full benefits of SOA requires an extensive commitment to change that will last for many years before SOA fully achieves the status of 'business as usual'. Fortunately wholesale change is not required in order to begin this change programme. Obtaining early benefits is possible — and such benefits will initially be found in areas such as improvements to integration along with efficiencies for in-house and bespoke application development solutions.

To achieve the full expected benefits of an SOA strategy requires a commitment to process improvement that extends beyond IT systems and into the heart of the organization. In addition application vendors must be pressured to continue to update their applications solutions and so improve their role as service providers. In the meantime hiding legacy, non-SOA solutions behind service facades offers a means to advance. In this sense, Oracle's Project Fusion offers an approach (yet to be validated in practice) which merits consideration by vendors and customers — alike.

Are IT procurement processes an obstacle to success?

Peter Bye
Consultant
Unisys Systems & Technology

Management introduction

The financial and other costs associated with IT project failure are well known and the subject of a vast literature (to which Peter Bye has made more than a modest contribution). As might be expected, much of the effort is spent in trying to identify the root causes of partial or complete failure. Factors such as unclear—and changing—requirements, lack of management commitment, selection of the wrong technology and plain incompetence are variously thought to be the prime culprits.

Nevertheless, it appears that there is no single dominant cause across the industry but rather a variety of factors, any one or more of which may be responsible in different cases. However, as Mr. Bye argues, there is one factor — IT procurement processes — which may lie behind a great number of apparently different causes of problems. Furthermore such instances are often associated with the implementation of middleware.

The argument

There are many forms of procurement processes. Amongst the most common are those where:

- **policies are defined to detail the procedures to be followed in procuring products and services, for example going to competitive tender for projects expected to be over a certain cost; these procedures are often formally documented, particularly in the public sector**
- **policies are created to prefer selected technologies, products or even suppliers; again, these may be formally documented and, in the case of the public sector, they may be imposed from outside or above**
- **a variety of other factors exist -— sometimes (even often) these are less than rational — and are as often to do with attitudes and ways of doing things; these are frequently not documented but nevertheless influence procurement processes.**

My argument is that — while procurement processes are almost always established with the best of intentions, primarily obtaining value for money for the customer — they may perversely achieve the opposite. In particular, I argue that procurement procedures often deliver negative results because they obstruct effective development processes.

I further suggest that, although this has always been a problem, it has become much more acute with the increasing incidence of composite systems, built on a middleware foundation. The kind of development processes required for these systems, especially at the inception of a project, are not compatible with many of today's procurement policies.

To explore these assertions, I:

- **begin by briefly discussing composite systems**
- **go on to sketch out the kind of development processes necessary in the context of these environments, using a typical project as an example**
- **consider how procurement processes may be obstacles to success**
- **finally offer some suggestions for improving the situation.**

What are composite systems?

A composite system is a distributed system comprising components implemented in different technologies and of varying ages. Such systems are becoming increasingly common, a trend that looks set to accelerate.

There are various reasons for the trend. First, the primary business driver in most organizations is cost, or — more precisely — cost/performance and risk reduction. Commercial organizations are under pressure to:

- **develop new application services as quickly and cheaply as possible, for competitive reasons**
- **achieve this with minimal risk.**

This is, however, not only a commercial imperative. While the public sector does not face precisely the same competitive pressures, there is an increasing emphasis by most governments on improved cost management and efficiency of delivery.

Secondly, most new applications are implemented within organizations that already have an IT infrastructure. Although new applications could be implemented as independent systems, there is a sensible trend to try to re-use existing applications, for example in mainframes.

New applications may therefore be developed in part as new logic and in part re-using existing logic. The concepts of service orientation and service orientated architectures are widely held to facilitate this approach, supporting the goals of competitiveness, cost and risk management and efficiency.

Finally, there is a need to make existing and new applications accessible through new channels, and to collaborate with other organizations, for example using Web Services. This external focus extends the distributed nature of today's systems.

In this context, a fundamental heterogeneity of composite environments — hardware platforms, operating systems and application infrastructures — has to be accommodated. Using middleware to integrate the component systems has become an integral part of the infrastructure. New logic may be custom developed or implemented in the form of a package, which then has to be configured and integrated with existing applications.

All these factors, and the expanding number of external connections, can-and do-make modern IT environments very complex. Indeed, they are becoming even more complex, rather than simpler.

Composite systems and development processes

Large scale, composite systems are not easy to implement, as the well documented failure rate suggests. To improve the chances of success it is, therefore, necessary to adopt the most effective development processes and methodologies. (It is also necessary to use the right people, a fact too often overlooked: development processes and methodologies make good people more effective; they cannot compensate for a lack of expertise.)

As an illustration of the development approach needed, consider a typical project. Suppose an organization operates some well-established, critical applications. Although the nature of these applications is not particularly relevant to this analysis, assume for the purposes of this example that they support the management of financial products. The organization's management considers that the systems are not sufficiently flexible: the current products are supported within what is regarded as a silo. Management sees that it is hard to create new products for customers.

One aspect of the silo view is not often mentioned. This is that, although it is usually relatively easy to discover the owner of a particular product, it is much harder to find out which products a customer owns.

In this example, the business vision is to create a new, flexible product management application. Part of this vision is to use a customer management application, which has been developed but is not yet widely used. In addition, the business would like to:

- **extend access to customers, using the Internet and other access channels**
- **enable customers to buy products supplied by partner organizations.**

The existing product management systems are implemented in mainframes. In contrast the customer management system is implemented in .NET.

There are many activities to be performed in converting this business vision into a working application. Although there is obviously some sequencing involved, there is also a need for a great deal of iteration, especially in the earlier stages.

It is here I wish to focus. It is what happens — or, rather, what often does not happen — that causes problems.

Early stage developments

Early stages include something like the following:

- **define functional requirements**
- **define non-functional, or environmental, requirements: performance, availability, reliability, security, manageability and testability**
- **define the architecture, in the sense of high-level design**
- **agree the implementation sequence**
- **begin detailed design.**

The business vision has to be developed into a set of functional requirements, which define what the system has to do. Of equal importance, and all too often not done, is the need to establish a set of non-functional requirements. Testability is included in the list above, as it should be considered at project inception and not tacked on as an afterthought.

Although every effort will have been made to ensure their completeness, the requirements — when first developed — should be considered as provisional. This is for various reasons, including that it may not be clear:

- **exactly what is required: this is not an admission of incompetence by anyone but rather a reality**
- **what can be done**
- **whether there will be new technologies involved, the possibilities of which need to be explored further as they enable still further new capabilities (not all of which, or even none of which, may be immediately useful for this project).**

Early developers of applications for the Web frequently encountered these sorts of problem. And they persist.

The process of requirements refinement begins with the development of models addressing different aspects, or views, of the architecture. Figure 6.1 is one possible view. It shows the new application, the existing product applications and possibly other business applications. It also shows a number of general services that may be required, for example:

- **systems management**
- **utilities, such as directories**
- **integration services**
- **access channel management.**

The whole structure is as interconnected by middleware, in the form of a service bus (or Enterprise Service Bus). Behind this is a hardware infrastructure on which the applications are implemented.

Figure 6.2 presents a different view. This represents the interconnections between the new and existing applications, both internal and external.

These, and other models, provide a framework for:

- **asking a number of questions, particularly 'what if?' questions**
- **exploring the consequences of specific decisions.**

Answering the questions, or more precisely obting reliable answers, requires much discussion and a range of skills, including a good understanding of:

- **the application domain and how the proposed application is intended to be used**
- **how you construct an application architecture**
- **the technical architecture, including hardware infrastructure, middleware and systems management**
- **the various technologies being considered, in particular application environments such as J2EE or .NET.**

From the starting point

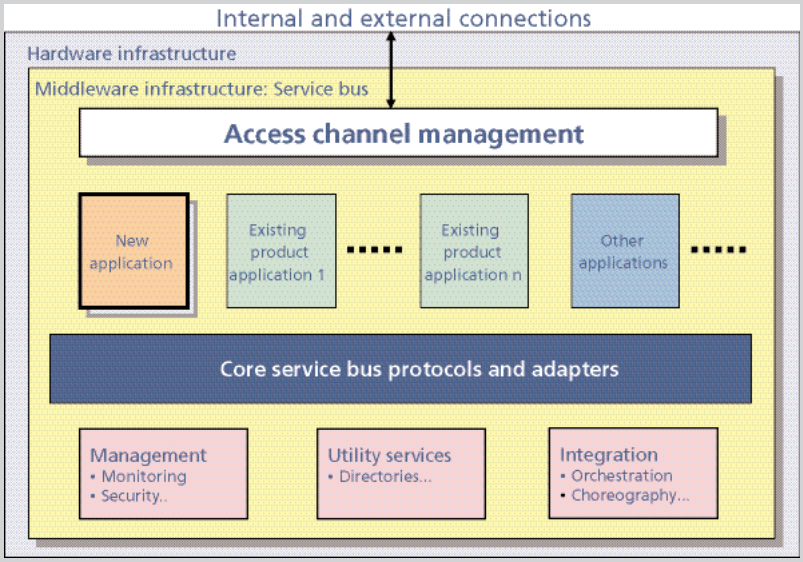
The starting point should be to seek to make reasonable choices about the technologies to be used, including application environments and databases as well as interconnection protocols. A high-level application architecture can then be defined. Some choices may be guided by what is already in place, because the organization may already have a well-defined middleware service bus architecture, and/or preferred application and database environments.

Questions can then be asked about the possible system:

- **are the functional requirements reasonably complete and realistic?**
- **do the technologies support the functional requirements, including the way the systems are expected to interact?**
- **where should the new application components be implemented (there may be a choice: if J2EE is the application environment, it could be installed in a new platform or in one of the existing mainframes)?**
- **will the technologies support the non-functional requirements?**
- **will the performance expectations be met and will the system be manageable?**
- **will it be testable?**
- **what are the consequences of failure at various points?**

Answering these and many more questions requires much discussion between the various parties involved. It is also

Figure 6.1: Architectural model showing middleware connecting components



likely to require short proof of concept projects, to explore and test various options.

The aim is to arrive at sets of functional requirements, non-functional requirements and a technology infrastructure that are consistent with each other and deliver what the users want. Indeed, much iteration may be required to reach this point, including making substantial changes to the requirements and even the business vision. Some ideas may not even be possible, at least as originally envisioned.

Following an agreement on requirements and technology, an implementation plan can be agreed, followed by detailed design. Although the discussions up to this point should have removed most of the existing inconsistencies, changes to requirements may still be necessary as the implementation unfolds. For these reasons, it is desirable to explore ways to implement the project in usable steps or phases, allowing as much user feedback as possible after each step.

The effect of procurement processes

The project discussed above could be implemented entirely within the organization concerned. In this instance, any procurement process is therefore internal, apart from possible hardware and software purchasees, plus some installation services.

How formal the internal process is depends on the organization. In some cases, it may be formal, with the IT providers being treated as a profit center. In other cases, it may be less formal. In any event, it ought to be relatively easy to hold the kinds of internal discussions that are needed. My concern, however, is with the more formal procurements, primarily those procurements that are using outside organizations to implement projects. Such formal arrangements are typical of the public sector, although they are not confined to it.

Obviously, it is not possible to operate without procurement procedures. It is not my intention to say that these should simply not exist. Rather, I am interested in the cumulative and systematic negative effects that are caused by wrong processes. I stress processes because I am not concerned with plain incompetence (which invariably can lead to undesirable results, but this will occur no matter how well designed any processes may be).

Policies for procurement procedures

The normal approach is that a request for proposal (RFP) is prepared and either sent to a preferred set of suppliers or openly advertised. Governments — and supra-national bodies such as the EU — follow these kinds of procedure, as do a number of commercial organizations. The RFP will have been prepared by the organization concerned, frequently with the assistance of consultants. The requirements will be determined by discussion with the business side of the organization, internal IT staff such as strategists and (with a bit of luck) with the representatives of the intended end users of the new system.

By far the most significant cause of difficulties is that the very processes to be followed inhibit real discussions. Many other problems flow from this one, single cause.

Policies are constructed with the intention of obtaining value for money and ensuring that competition is fair. Questions are typically confined to forums such as vendor conferences, where they may be verbally submitted and answered, with perhaps a subsequent written response. Questions may also be submitted in writing, with written responses being sent to all and every other candidate vendor.

It is very hard, under these sorts of rules, to engage in the depth of discussion necessary at project inception — as discussed earlier — to arrive at that optimum set of requirements and system architecture that is most likely to achieve success. Asking 'what if?' questions, trying different approaches — with short proof of concept projects — and even talking to the ultimate users are next to impossible.

Other factors can operate to compound the problem. First, there is, in many cases, a tendency to allow too little time for an effective response to be prepared. The effective time to deliver the response is usually shorter than the allowed time, as most bidders have to obtain internal agreement to proposal changes and risk. This takes a slice out of the available time.

A second factor is that the RFP may only cover part of the ultimate composite system. This is manageable if a number of vendors are bidding in a consortium and can therefore work with each other in preparation. However, I have seen cases where the RFP is for a component part of a composite system — without any adequate plan for integrating it into the overall end environment.

In this sort of instance, the result is likely to be a set of proposals that are not as good as they could be. Although the proposal may cover what has been asked for in the RFP, it is likely that this will not be what the users really want. But this only becomes apparent at a much later stage — and almost invariably results in:

- **user dissatisfaction**
- **cost overruns**
- **delays.**

In view of these uncertainties, vendors face a difficulty in quoting fixed prices. Their reaction is understandable: they inflate the price to cover the risks. Such factors also make it difficult for those who are assessing proposals to make a reasoned decision.

There is then the possibility of choices being made purely on the price, rather than price/performance, as a measure of value for money. Too often it is easy just to take the cheapest offer. This does not mean that the best is necessarily not the cheapest. Rather, it is to suggest that price alone does not and should not determine value.

Policies for technology and supplier selection

In addition to formally specified approaches to procurement, many organizations have policies defining technology and supplier selection. In the case of the public sector, these may:

- **be provided by some agency representing all public sector IT procurement**
- **take the form of lists of technologies, products and suppliers from which organizations either should or must make selections.**

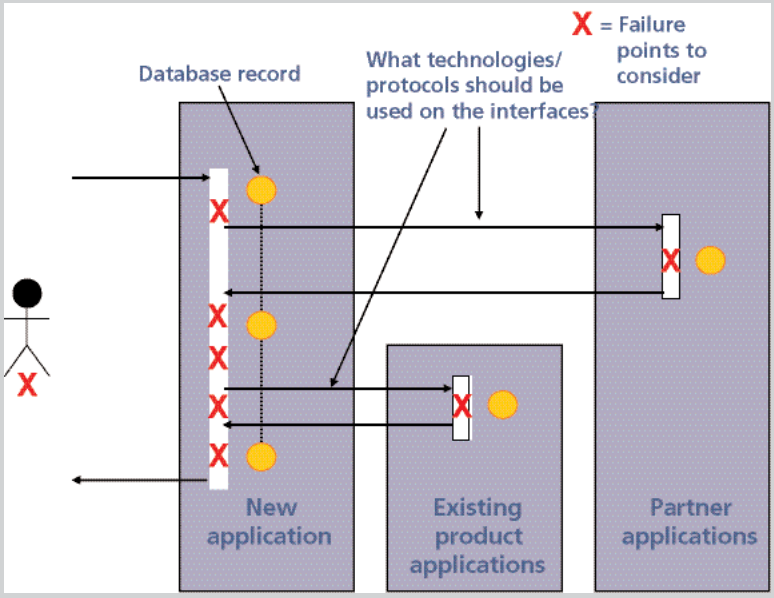
Most of these policies are established for the best of reasons. They can, however, cause problems. The following describe three examples.

Technology policies may result in inappropriate selections of product or, worse, a decision to rewrite an application simply because it is thought to be in the wrong technology. There are many cases of significant projects implemented just for the sake of changing technology (often because the original system is in a mainframe). Although there are reasons for doing such rewrites, for example where the business model has fundamentally changed, they are often a waste of money.

To compound the problems, the rewrite may fail. Although the organization would still have the existing application, it is likely that no upgrades would have been made, as the anticipated new system is anticipated to contain all the additional required features. In practice, it may be far better to add new features to an existing system — or to open it up as a set of services in a composite application.

Secondly, strong encouragement to use packages is, in itself, an apparently sensible idea. Choosing a package can, however, misfire as well as be wasteful. It may turn out that the cost of the package, plus the amount of implementation time required in adapting it to the specific application, is higher than the cost of writing exactly what is required from scratch.

Figure 6.2: Interconnections between new and existing applications



47

Further, if the application is mission critical and is key either to differentiating the business in a competitive market or to performing some highly critical role in the public sector, packages may not be the best choice. Packages are, by their nature, general purpose; they may need disproportionate customization.

In any event, it is always worth questioning whether a package or a new development is the best approach. The answer may, ultimately, be a package — but some essential analysis and thinking has been done in coming to this decision.

The final example concerns supplier policies, which specify lists of approved suppliers. In the interests of cost reduction, offshore suppliers may be recommended for new developments. While many offshore suppliers are extremely competent, they usually require extremely well-defined sets of requirements, which they will follow precisely in their implementation.

The difficulty with this approach is that the amount of discussion necessary in the early stages of many projects today is well nigh impossible when dealing with offshore suppliers — unless the commitment is made to spend sufficient time onshore, working directly with those who require the system. This of course increases the cost and, too often, defeats much of the purpose of going offshore.

Other factors in procurement

There are many other factors that can affect procurement, apart from the more or less formal policies. I do not intend to dwell on them, as they would be a factor even if all policies were perfectly defined. But they should not be ignored.

Three examples will suffice:

- **attitudes to vendors, whether negative or positive, are sometimes based on solid experience but are also sometimes irrational; 'I don't like vendor X' is a not uncommon attitude and, while the feeling may stem from bad personal experiences in the past, it may not accurately reflect on the capabilities of a given supplier**
- **ideology about technology and products is almost assumed; this sometimes takes on a religious fervor, causing undue pressure for (or against) particular technologies — and recognizable instances of this include debates on (say) the relative merits of .NET and J2EE or Windows and Linux (one consequence is that technologies and/or products are forced into an environment where they are too often inappropriate)**
- **specifying requirements with a preferred solution in mind is all too common; this is often also be related to the previous two factors (perhaps especially where technology preferences or biases are involved).**

A possible way forward

What can be done to reduce the likelihood of procurement processes adversely affecting IT projects, especially in the composite environments that are increasingly common? I offer three suggestions.

First, the need for the upfront iterative process of discussion and experiment is critical. It, therefore, has to be allowed for in procurement. It could at least in part be built into the actual process, before a contract is awarded. This would extend the procurement time and cost — but would, I believe, result in better systems.

One way of handling this would be to pay the costs of the losers in a bid. This is expensive but could ultimately save costs by increasing the success rates of projects. This approach is in fact adopted in some cases, for example large defense projects.

A second option is to allow greater flexibility at the start of a project, so that the necessary discussion and proof of concept projects can take place. This makes an initial fixed price impossible. But subsequent implementation projects could be done at a fixed price, following the early discussion time. Again, the likelihood of greater success rates, with fewer overruns of cost and time, should make this worthwhile.

Finally, establishing a standardized infrastructure — designed to support any kind of application an organization might wish to develop — can be effective. The number of variables is reduced, as the infrastructure provides a well-defined environment into which the new components can be placed. The effect is to reduce the amount of discussion required, particularly about the suitability of technology (although this does not eliminate such 'debates' altogether).

Building such an infrastructure can be combined with delivering new business functions — or be a project in its own right, with no immediate business driver in mind. The goal should be to make it easier and quicker to implement new business services.

Developing a pure infrastructure project should, of course, follow the iterative approach. Instead of discussions about specific functional and non-functional requirements for a set of business services, the need is to identify general classes of requirement and the kind of infrastructure that would be needed to support them.

Infrastructure projects are organization critical. If they are not correctly designed and implemented, they adversely affect all subsequent application development projects. The approach to procurement is therefore of particular importance in these cases.

Management conclusion

In this analysis Mr. Bye has shown that the number and complexity of composite systems is increasing. He explains why, if projects are to succeed, their implementation requires much early discussion and experiment to improve the chances of success.

He then argues that procurement processes — including formally defined procedures for procurement, selection of technologies and suppliers, and a variety of less rational factors — are obstacles to success. They 'get in the way' of the required discussion, either making it difficult or nearly impossible.

With today's ever increasing number of composite systems, it is imperative to identify ways of improving the selection, especially by enabling early discussions. In his analysis he suggests three approaches:

- ***building time into the procurement process for the necessary discussions***
- ***adopting a more flexible approach early on in a project, to ensure that sufficient discussions take place***
- ***constructing a standardized infrastructure designed to accommodate a range of projects of this sort, thus reducing the need for some of the upfront discussion.***

The first two of these could require some change in the way costs are managed, but should have an overall beneficial result. The third is, in fact, happening more and more already.

An important point to note is that it is critical to obtain the right standardized infrastructure. This is in itself a project, which itself requires the kind of implementation approach discussed here. A poorly defined and implemented infrastructure is a serious obstacle to the success of any implementation depending on it. Management must not permit or encourage such a result.

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

Subscription Center
19 St. Michael's Road
Winchester SO23 9JE
England
Telephone: +44 1962 878333
Fax: +44 1962 878333

Research and Editorial Office
19 St. Michael's Road
Winchester SO23 9JE
England
Telephone: +44 1962 878333
Fax: +44 1962 878333

Email and Internet
Email:
spectrum@middlewarespectra.com

World Wide Web:
www.middlewarespectra.com

ISSN 1356-9570

[incorporating FINANCIAL MIDDLEWARESPECTRA ISSN 1460-7220]