MIDDLEWARESPECTRA

incorporating FINANCIAL MIDDLEWARESPECTRA

Contents

August 2004

Volume 18 Report 3

Understanding why CIOs fail

Bernard Abramson
Consultant

Management introduction

Bernard Abramson is a consultant based in Princeton, New Jersey. He started his career in IT in the UK before moving to the US in 1985 to work for a major management/IT consultancy. In the late 1980s he was recruited to work as an IT Manager for one the top pharmaceutical manufacturers. He retired from this firm in 2004.

In this analysis, his goal is to:

- ***examine the symptoms of the challenges faced by IT management***
- ***present a diagnostic framework which reveals their underlying causes, and***
- ***recommend a treatment.***

As might be expected, this is — unapologetically — a view from inside IT looking out. The view from other perspectives can be examined in subsequent papers. Nevertheless, understanding why CIOs (Chief Information Officers) fail is instructive for anyone who has either to introduce infrastructure and middleware or to work with what already exists.

Frustrations

In the course of a relatively long career in IT, spanning several industries and countries, I have found certain aspects of IT management deeply frustrating. Among these have been:

- **dealing with managements who not only do not understand IT, which is forgivable, but do not want to understand it, which is not; as a result one is always simplifying IT issues in order that executives might understand them — but this leads to the unpleasantness of simultaneously trivializing complex issues and then appearing to whine about those trivialities**
- **sacrificing quality for speed: where life may be at risk, managements do understand the risks of cutting corners in systems development but where only profits are at risk an insistence on quality is easily construed as negativity (or its close relation, 'a lack of leadership')**
- **being relegated to the role of order-taker: this is partly a consequence of the above but also derives from user-driven technology innovation and the presence of technology cowboys among the 'client base' who seem to have plenty of money to spend and, therefore, believe in "I pay, I say".**

I have encountered these problems sufficiently often across a reasonable diversity of organizations on different continents to believe that they are an established part of IT's territory. If you look carefully enough, an explanation for many a CIO's departure can often be found here.

There must be some underlying pathology. It may not be curable but we should try to understand it. In this analysis I propose to explore this in the context of the pharmaceutical industry, simply because I know that industry best. Yet I believe the results apply widely.

Paradoxes

IT leadership in a global corporation is peculiarly difficult. It seems inevitable today that one must sustain three paradoxes simultaneously:

- **provide a robust, secure and worldwide infrastructure service plus support the latest IT innovations and still allow for rapid change and local differences**
- **create competitive advantage even though the strategic potential (or capability) of IT is not something which most business management personnel have any interest in or understanding of**
- **reduce IT costs continuously but continue to invest in order to enable the rest of the business carry on reducing its costs.**

These paradoxes — or, more properly, conflicts — lie at the heart of the frustrations I and many others have experienced as IT leaders. Arguably these conflicts make any CIO's job 'mission impossible'.

Before looking at the paradoxes more deeply, let me begin by recognizing a fundamental truth. Few senior business executives experience IT as a 'good thing'. The immediate consequence is that two handicaps exist which place IT at a real disadvantage in dealing with those senior executives.

The first is that computers are the often the one aspect of business life that an executive's children seem to understand better than he or she. IT makes them feel stupid in ways that science, HR, legal or even finance issues do not. Not understanding genomics or proteomics is not something to be embarrassed about (even in a pharmaceutical company) — one needs several years as a post-doc for that. But not understanding something that most 8-year olds appear to pick up as easily as a throat infection does not make business executives welcome the IT leader into the executive suite.

The second, more subtle but more dangerous, handicap is the irresponsible way in which IT professionals (or more accurately vendors, consultants, journalists and popularizers) promote new technologies as silver bullets. Most technology advances and improvements are beneficial. Nevertheless the benefits do not appear immediately, or without great effort. From the perspective of a non-comprehending, not to say antipathetic, executive management, IT is 'just one damned excuse after another' to spend money with little visible return. Just think of:

- **client/server technologies**
- **ERP**
- **Y2K**
- **CRM**
- **Web Services**
- **etc.**

Management's skepticism is understandable. The demand for funding, resulting from the overselling of immature technologies and combined with the frequent failure to ful-

fill promises, has hardly helped to demonstrate that IT is a mature discipline which has a real contribution to make to corporate strategic success.

Symptoms — infrastructure

Let us start with infrastructure. In this I include many of the layers of shared and global business process and supporting applications — as well as the traditional infrastructure layers of network and computing utilities (Figure 1.1). The tension that IT management feels comes from the expectation, almost never coherently articulated by executive management, that its (IT's) role is to provide a mature, robust infrastructure while also being agile and flexible in meeting whatever the customer needs.

The key words in conflict here are 'robust' and 'agile'. The pre-requisites for effective robustness in IT systems lie in detailed planning, exhaustive testing, continuous monitoring and constant iteration. When this infrastructure must serve a global enterprise, there is an overwhelming need to manage change with extreme care. On the other hand, agility means rapid change to components. There is no escaping from that fact that these two demands fundamentally conflict with each other.

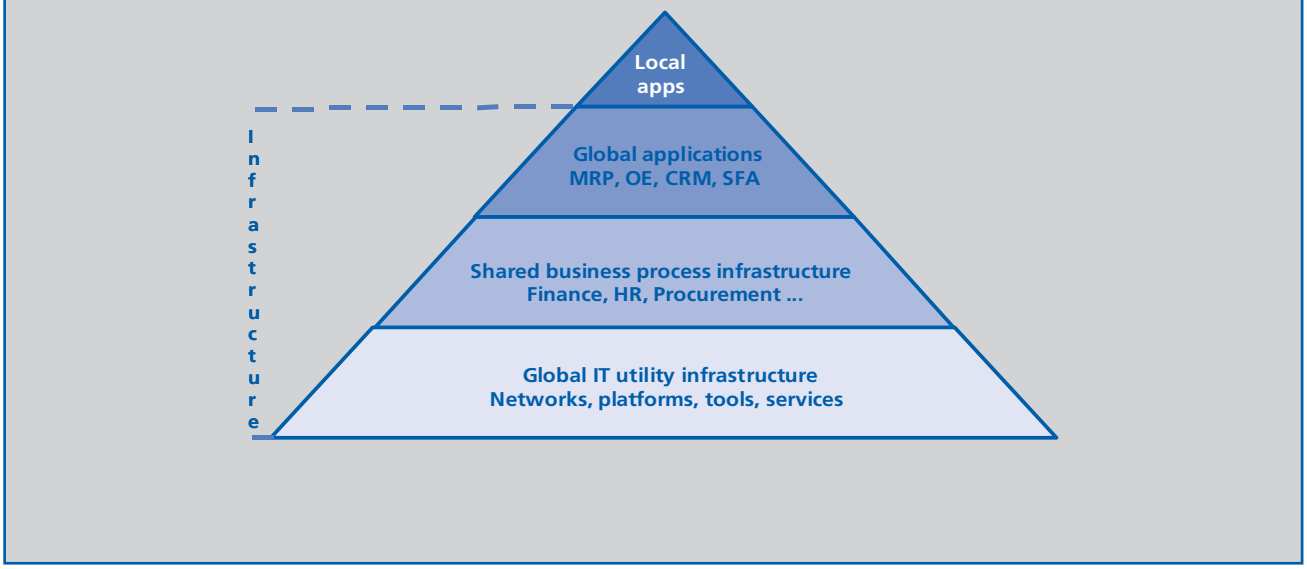
Furthermore, IT's customers often demand the latest and greatest in technology. The introduction of innovative technology requires that IT works with necessarily unproven concepts and combinations of ideas, systems and software. Inevitably such expectations conflict with maintaining the service levels expected for a working business infrastructure.

When IT decides to introduce new technologies it does at least retain most of the control levers, a luxury denied us when users take the initiative into their own hands. Most of us in IT have 'discovered' users who have implemented their own wireless networks or established collaborations with external partners giving them Intranet access. Almost always these are well-intended innovations. But equally often these initiatives produce network outages, security breaches or even the loss of proprietary information. But guess who the CEO will be calling first: the CIO (who may not even know what has been delivered by non-IT users in his own organization).

In this context, information security is an enormous topic in its own right. Suffice it to say, for here, that the best way to jeopardize security is to implement new technologies, products or services without thorough testing and change management — which only serves to re-emphasize the risk that demands for agility create.

Another force which exacerbates infrastructure conflicts is the global vs. local divide. This occurs in many guises:

Figure 1.1: Layers of networking and infrastructure



- **the geographic ('reimbursement rules for my country require that we provide different data to the authorities')**
- **the functional ('Basic Research must use the latest technology regardless of corporate standards')**
- **the structural ('joint ventures, partnerships, alliances or collaborations must set their own rules').**

In common with most business activities, IT applies standardization and shared services to reduce its infrastructure operating costs. Economies of scale and control of standards can be gained only from centralized activities. Accommodating 'local' differences produces a tension with these centralizing forces.

The misunderstanding and frustration between IT and its clients is mutual and profound. As I will discuss, this is a powerful clue to the fundamental challenge IT management (and CIOs in particular) face.

Symptoms — strategy and costs

Let us now move on to strategy. In IT most believe that we can make a strategic contribution to the business. After all everyone 'knows' that information is the essence of business and almost every modern business process is mediated by technology. Even so, IT strategic plans come and go — with each new CIO — and IT's strategic stature does not seem to grow.

Recall the debate about the impact of IT on productivity, the so-called productivity paradox. At that time it was famously said that "you can see the computer age everywhere but in the productivity statistics". This can readily be extended into the view that "you can see computers everywhere but in company strategy". It took some 15 years from the start of the debate over the productivity paradox until the impact of IT on productivity began to become visible. It may take just as long for the same to happen with IT strategy.

Consultants tell us, understandably if obviously, that IT strategy must be aligned with business strategy. From the mid-1980s to the mid-1990s consulting company CSC Index conducted an annual survey of CIO concerns. 'Aligning IT goals with business goals' was the number 1 or number 2 priority almost every time. The application of some powerful brains over so many years ought to have yielded some advances. Unfortunately, the evidence is not strong.

A recent Forrester Research paper summarized the problem as follows: "CIOs are being asked to align IT with business strategy without being told what the business strategy is". One of the reasons identified is that businesses do not have clear strategies. The suggestion is then made that the CIO should force the business to develop a business strategy. But this is about as helpful as attempting to prove Fermat's Last Theorem as a precursor to proving Pythagoras's theorem or asking that the soup be hot before cooking it.

There is a fundamental question that needs to be addressed. This is: can IT provide sustainable competitive advantage?

Take, for example, the pharmaceutical business. It has only two core competencies:

- **R&D**
- **marketing.**

It is not self-evident how strategic advantage can arise outside a core competence, which brings me to a key point. There is a real difference between being a business partner and being a strategic partner:

- **a business partner plays a vital role in the success of business operations**
- **a strategic partner goes well beyond this and plays a vital role in the core strategic leadership of the business.**

IT is clearly core to strategic success at Amazon.com, FedEx, and AT&T, for example. In each case the CIO reports to the CEO.

Then there are costs. The symptom here is the paradox that IT must keep its costs down while being the conduit through which all other parts of the business invest to reduce their costs.

The challenge here is substantial. Manufacturing, Finance and HR seek to reduce their costs by business process standardization and exploiting shared services. This entails investing in ERP (or similar) solutions plus a high-availability global infrastructure. All of these increase IT costs. The conflict is yet again clear.

Recall the Red Queen in Lewis Carroll's Through the Looking Glass. In crossing the chess-board she and Alice had to run as fast as they could just to stay in the same place. "If you want to get somewhere else, you must run at least twice as fast as that."

Suggested reading includes:

- **Bashein and Markus:**
  A Credibility Equation for IT Specialists
  (Sloan Management Review, Summer 1997)
- **Solow, Robert M.:**
  We'd Better Watch Out
  (New York Times Book Review, July 12, 1987, p. 36.)
- **Ross, Christine Ferrusi:**
  Aligning IT With The Business When The Business Strategy Is MIA
  (Forrester Research, December 8, 2003)
- **Carr, Nicholas G.:**
  IT Doesn't Matter
  (Harvard Business Review, May 1, 2003)
  (R0305B) [NB. Carr argues that as IT is a commodity if it cannot deliver competitive advantage.]
- **Carroll, Lewis:**
  Through the Looking-Glass
  (Macmillan, 1871)
- **Rockart, John, et al:**
  Eight Imperatives for the New IT Organization
  (Sloan Management Review, Fall 1996)
- **Dvorak, Robert, et al:**
  Six principles of high-performance IT
  (The McKinsey Quarterly, 1997 Number 3)
- **Feeny, David and Willcocks, Leslie:**
  Core IS Capabilities for Exploiting Information Technology
  (Sloan Management Review, Spring 1998)
- **Evans, Bob, Editor-in-Chief, Business Technology: What Tops Your To-Do List?)**
  (InformationWeek, May 27, 2002
- **Treacy, Michael and Wiersema, Fred: The Discipline of Market Leaders**
  (Addison Wesley, 1995)
- **Reality Hurts: CIO Turnover Increasing**
  (Meta Group, May 13, 2003)

Figure 1.2: References

Summarizing the symptoms and consequences

To summarize the symptoms:

- **IT must provide a mature, robust infrastructure yet be ready to implement new technologies and adapt responsively to 'local' demands; when (if) we can resolve these conflicting demands, IT will reach the first rung of the ladder and become a trusted service provider**
- **IT is racing the Red Queen: as IT costs are relentlessly slashed, business costs and the demands to do more with IT increase even faster; when IT can demonstrate that value delivered to the business can be continuously improved, IT will reach the next rung of the ladder and become a trusted solution provider**
- **IT is visible in every single activity across the entire pharmaceutical value chain except corporate strategy; when IT becomes a core competence for pharmaceuticals, IT will reach the top rung of the ladder and become a trusted strategic partner.**

Where can a harassed IT executive turn for help and guidance? There must be someone out there who does this right. There must be some successes.

Successes

Academic and business literature (Figure 1.2) provides a strong consensus that successful IT organizations do four things well:

- **leadership: trusted, strong relationship with company leaders**
- **business driven IT decision making**
- **technology adoption that is fit for its purpose**
- **frugal spending combined with wise investing.**

It is somewhat tautological to state that if you can exhibit these qualities then you have either avoided or resolved the paradoxes that concern me. The underlying question is: how does one to achieve this level of success?

It is useful to look at the market discipline theory proposed by Michael Treacy and Fred Wiersema (Figure 1.3). These authors identify three value disciplines or dimensions which drive company performance:

- **product or service leadership — products or services that expand existing performance boundaries; corporate examples include Sony, Intel and 3M**
- **customer intimacy — knowing and delivering what the customer wants; major examples are hard to come by here (although consider IBM — before the 1980s — Nordstrom and many consulting companies) for organizations which excel here tend to be smaller and concentrate on niche markets (plus success is hard to sustain)**

- **operational excellence — a combination of quality, price and ease of purchase that no one else can match; examples here include Wal-Mart, Dell, Amazon, Southwest Airlines and MBNA.**

Treacy and Wiersema further propose that successful companies provide the best offering in the marketplace by excelling in one chosen dimension of value. They maintain threshold standards on the other dimensions.

Returning to the characteristics of successful IT functions we can see that a leadership which has strong business relationships, and the successful adoption of business driven decision making, are aspects of a customer intimate IT function. Making technology work and doing so cost effectively are the essence of IT operational excellence. Product leadership is a threshold dimension.

My conclusion is, therefore, that a long term successful IT organization will concentrate on both customer intimacy and operational excellence when those dimensions are defined by:

- **leadership**
- **business-based IT decisions**
- **making IT work**
- **wise financial stewardship.**

Businesses, and their executives, need IT leaders who combine the ability to build strong business relationships with the ability to drive IT operational excellence. Yet it can be argued that this is the final paradox: these personal qualities are mutually exclusive.

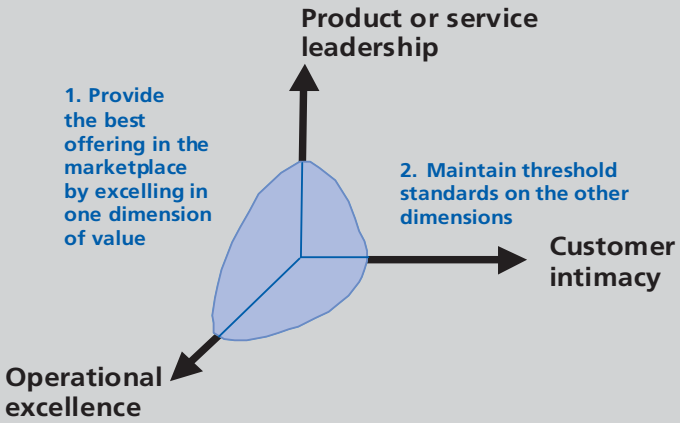
Successful business executives, those who excel at customer intimacy, are synthesizers who see the world in 60/40 terms:

- **they are adept at never saying 'never'**
- **they prefer shades of gray to all other color schemes**
- **as they tell it, everything is perfect and (almost) always getting better**
- **precision is less important than best guesses; they are comfortable with directional statements.**

On the other hand, IT professionals who achieve operationally excellent results are analytical and pessimistic:

- **they look for decision rules in any given situation so that they can determine a definite outcome**
- **they do not tolerate an 'if — then' with no 'else' clause**
- **Murphy's law is an axiom of existence**

Figure 1.2: Market discipline theory of Wiersema and Treacy



3. Dominate your market by improving value year after year
4. Build an operating model dedicated to delivering unmatched value

- **they are binary: zero/one, yes/no, works/does not work (or fails).**

Treatment — IT CEO plus IT COO

For an IT function to maximize the possibility of success it must be led by two people. At the top, there must be a business person who can build successful relationships with the organization's business leadership and embody customer intimacy. This person must be the CEO of the IT function looking outside, being the salesperson, and a great communicator. The IT CEO excels at managing perceptions. In short, IT needs a leader not a manager. These qualities are rare indeed.

This IT CEO needs to work alongside an IT COO (Chief Operating Officer). This COO role is focused internally. The individual must be a first rate planner, an architect and strong technologist. The IT COO excels at execution. This manager is a much more familiar individual. Many IT leaders do this job exceptionally well.

When these two exist together IT will have met the necessary conditions for success. This means that both have to excel within their respective spheres, possess a mutual respect and a shared philosophy.

That said, my experience indicates that this is 'necessary but not sufficient'. There is one more requirement: the CIO must be a true member of an organization's executive management team. A direct reporting line to the CEO of the company is a powerful and visible symbol. But it can often be just that, symbolic. More is needed. Where the CIO has the leadership and business qualities of a CEO, he or she will be able to earn the respect and trust of the top managers and become their peer. This is more important and productive than a line on an organization chart (although if the line is also there, so much the better).

Put another way, the key to success is that the IT function must have a leadership team which is part of the management inner circle and can balance the demands of customer intimacy and operational excellence. With these conditions in place one can return to the paradoxes and see how IT fares.

Back to the paradoxes

Let me start with the second paradox which was the need to create competitive advantage when IT is not on business management's radar. To a degree, with the model IT CIO, I have defined this paradox out of existence.

With the CIO at the enterprise's top table, IT is positioned to become a full partner rather than an order-taker. In a business where IT is at the strategic core, a CEO-type leader for IT will open the door to strategic partnership. In a business, like pharmaceuticals, in which IT is not a core competence, it is absolutely better to have such a leader, but business partnership is the goal and this will not wholly remove the frustrations described earlier.

In all cases the CIO must articulate a strategic direction for the role of information and technology within the enterprise. This strategy will have two core parts:

- **a transformational vision, a source of competitive advantage, expressed in language that will resonate viscerally with his (or her) peers; there is no formula for creating this and the talent to identify a vision is a distinguishing characteristic of the CEO-type of executive (expressing the vision is the CIO's 'elevator speech': it will change with time, or else the CIO will)**
- **a strategy for combining enterprise information, infrastructure and technology innovation: at the heart of the strategy is an operating model describing how the three characteristics of successful IT organizations will be achieved (the management of enterprise information, infrastructure and technology innovation will be constant threads that weave through this operating model).**

Now let me return to the first paradox: this encompassed the balancing of a robust, global infrastructure with the agility to meet rapidly changing business needs. The strategic approach I have outlined provides several levers to pull:

- **IT's positional power will derive from the CIO's membership of the top management team**
- **the IT COO will deliver the operational excellence now that he or she is freed from the demands of the role assumed by the IT CEO (this division of labor is vital)**
- **there now exists a business-based IT decision process**
- **the operating model explicitly addresses infrastructure and innovation.**

On this base an organization can benefit through building a strong demand management process for IT and integrating it with business planning. The 'shadow groups' and demand for local variations are unlikely to vanish; however,

once these are subject to the scrutiny provided by the demand management process — and when held up to the light of the infrastructure/innovation strategy — they become manageable and can be beneficial.

The third paradox — spend less but invest more — will fade in significance when tangible, visible results are obtained from taking the above actions. Value will become the driving force.

But value is very much in the eye of the beholder. The CIO's ability to manage perceptions will pay dividends here. Cost management will continue to be a top priority — but it need no longer be the first and last subject that comes to business management's mind when they hear the 'IT' acronym.

Management conclusion

Corporate IT managements invariably devote their time and energy to governance and to operational excellence. These are important and essential. They apply as much to the search for infrastructure and middleware delivery as application appropriateness.

However, in the absence of the type of leadership team described here by Mr Abramson, they — and the CIO — will fail. Their efforts will resemble the drunkard's search for his lost keys: that happens where the light is, rather than where the keys were lost.

Finally, this statement from an article on CIO turnover by Meta Group encapsulates it all rather neatly: "We believe CIOs will be increasingly successful based on their ability to complete constant, consistent, comprehensive communications about technology-supported business projects, but do so with the savvy and sophistication of any top-notch public relations or marketing agency."

Business Process Management

Tom Welsh
Consultant

Management introduction

Business Process Management (BPM) is one of the hottest subjects in IT right now, easily surpassing Web Services and Service Oriented Architecture (SOA) in terms of direct business relevance. Howard Smith and Peter Fingar — authors of the book Business Process Management: The Third Wave *— portray BPM as a radically new discipline based on the rigorous mathematics of Pi Calculus, which sets business processes apart from run of the mill software applications. They sum up BPM's attraction with the provocative slogan: "Don't bridge the Business-IT Divide: Obliterate it." Clive Finkelstein, who pioneered Information Engineering in the 1980s, calls BPM "an approach that uses a service oriented architecture (SOA) as a bridge between business integration and technology integration".*

From the technical point of view, most of the components necessary to bring about the BPM vision are already available. Nevertheless, as Tom Welsh analyzes, it seems unlikely to be fully realized for some time to come. The obstacles, though subtle, are many and formidable.

The attraction of BPM

The aims of BPM are undoubtedly worthy ones. Who could quibble with the promise of:

- **breaking down artificial barriers between departments and functions?**
- **automating business processes from end to end?**
- **giving business managers direct control of the processes for which they are responsible?**

The BPM vision is something like all the Enterprise Commerce Management packages (such as ERP, CRM and SCM) wrapped into one — but unshackled from the millstones of intensive programming effort, expensive consultancy and lengthy delays. Best of all, some BPM advocates paint a glowing picture of a world in which business managers no longer have to depend on IT departments to translate their requirements into software. Instead, they would control and adjust business processes themselves, either graphically or using a special language.

Business Process Management, modeling and automation

Delphi Group has described BPM software as a $500 million market, with up to $1.5 Billion being spent on related services. It expects the BPM market to grow by up to 30% annually through 2004-2006.

Other analysts offer contrasting estimates. Aberdeen Group sizes the BPM market at $3-4 Billion, says that this is shared by more than 60 vendors and is likely to reach $16.7 billion by 2005. Meta Group believes that as many as 85% of companies already have BPM projects on the drawing board.

In a survey of IT decision makers carried out by Vitria Technology, no fewer than 80% ranked BPM as one of the most important initiatives for executing their business strategy. A majority of respondents expected BPM to cut the time and costs of business processes by up to half. Some found that BPM projects paid for themselves within months due to decreased costs and reduced delays.

There are high hopes that BPM will turn out to be a more complete solution to real-world business problems than work flow — which is effective in some applications but limited in the types of process it can automate. Another glittering prize is the possibility of 'filling in the gaps' that ERP and CRM have failed to address. Companies like Intalio already have impressive tales to tell about their successes in helping customers surmount the limitations of their existing application packages.

But ask the key question, 'Exactly what is BPM?' and it becomes swiftly apparent that there is very little consensus about the answer. According to Forrester Research, today's BPM market embraces products that come from at least six different directions:

- **work flow**
- **EAI**
- **B2B**
- **business process integration**
- **business process modeling**
- **process automation.**

Aberdeen Group's Darcy Fowkes notes that "[BPM] has become the gleaming focal point in a number of technology suppliers' portfolios". As will become clear, this is an insightful observation.

The term 'BPM' has been in widespread use since the mid-1990s, but its meaning would seem to have shifted. It must have, because the specifications on which today's pure-play BPM products are based were not written until 2000 or later.

Work flow, however, has been around for much longer, together with business process modeling (the 'other' BPM) — evidence that at least some of today's BPM products have evolved from these earlier categories. Companies including Forte, Micrografx and even Netscape were proclaiming their BPM ambitions by 1997, which was also the year when Vitria claimed to have pioneered this area.

Surmounting paradigms

BPM (Figure 2.1) ostensibly surmounts the limitations of previous paradigms, such as work flow, by spanning network, departmental and even enterprise boundaries. In theory, it allows users easily to model, integrate, execute, manage and optimize all processes — in parallel, across organizational demarcation lines and even in real time. This wide-ranging functionality requires at least three essential components:

- **a distributed, parallel process execution engine: this already pushes the state of the art, if taken literally, but Milner's Pi Calculus at least provides a suitable notation for describing the working of such an engine (an alternative theoretical approach is based on Petri Nets)**

- **a process modeling workbench: this component corresponds to what is often called Business Process Modeling (but, unfortunately, this name has the same acronym as Business Process Management, a fact that has been responsible for increasing confusion)**
- **optimization and management tools: once a BPM engine is running it can add a great deal of value by revealing or uncovering delays and other sources of inefficiency (the related subject of BAM — whether expanded as Business Activity Management, Business Activity Monitoring or Business Analysis and Monitoring — yet another acronym that is causing yet more ambiguity).**

Is BPM feasible today?

With the 21st century under way, many factors contribute to the rising clamor for IT somehow to 'automate business'. There is the perennial, though misleadingly simple, precedent of factory automation. If a car assembly line can be run by computers, why cannot any other organization be run by IT?

Automation has other appeals. It promises to reduce the need for balky, troublesome, error-prone human employees. Unlike systems, people are liable to fall sick. In addition, numbers, pay and benefits are tempting targets for accountants seeking to cut costs. In this context, BPM in its purest form seems to offer executives the chance to run their businesses like skilled pianists playing their instruments — with a minimum of frustrating delays, obstructions or misunderstandings.

In a much-quoted research paper entitled *Ten Pillars of Business Process Management*, Tyler McDaniel, director of application strategies at Hurwitz Group, wrote: "Each business has unique characteristics embedded in the processes. In a commodity market, managed business processes define advantage. In an open market, managed business processes create opportunity. Too often, however, enterprises don't understand or control their processes. Management might have a model of an ideal process, but the reality of the execution may be strikingly different, leading toward redundancy, error, gaps, and inefficiency. Businesses without agile control over their processes often impede their own success."

But McDaniel's argument does not really stand up to close inspection. After all, if a business does not understand or control its own processes, surely automating them is the last thing it should think of doing? As a corrupted old saying goes: 'automating a bad process simply makes the garbage revolve faster'. What McDaniel does achieve, in the quoted passage above, is to advance an cogent argument in favor of business process modeling.

The biggest obstacles to successful implementation of BPM are, inevitably, the non-technical ones:

- **how well can an enterprise identify real — as opposed to theoretical or desired — business processes that can be analyzed and then fed into a BPM engine?**
- **when those processes are automated, will any small but vital steps have been missed out (like allowing time to countersign checks, for example) or incorrectly described?**
- **how safe is it to give managers the power to change business processes in real time?**
- **last but not least, how will the human components of the enterprise take to the new automated dispensation?**

IT experts can, no doubt, build and install BPM systems that do most of what enthusiasts like Smith and Fingar recommend. Indeed, vendors like Action Technologies, Fuego, Intalio, Lombardi, Metastorm, Savvion, Staffware and Ultimus already market such systems with some success.

It is also noteworthy, however, that few — if any — of these vendors' products are in any way interchangeable. Quite simply, the necessary common understanding and thereby the relevant standards are not fully mature yet.

The implications of BPM for middleware

Curiously enough for such an ambitious programme, BPM does not strain the limits of what present-day middleware can reasonably be asked to do. Within a single organization, there is a plethora of potential approaches ranging from CORBA to WebSphere MQ and their many competitors — although these might best be deployed in the guise of JMS providers. Between enterprises, Web Services stand out as the obvious solution: after all, much care has gone into ensuring that they work reliably across the Internet (and through firewalls).

The real difficulty does not lie with communication as such. Rather, it concerns the co-ordination of multiple interrelated activities — essentially, a kind of programmatic work flow.

The hardest problem is making sure that the scripts or work

flows at each end of a BPM link are wholly compatible with one another. If they are not, co-operative processes break down in confusion.

Relevant standards and specifications

Pi Calculus and Petri Nets are believed by many to provide an adequate mathematical basis for distributed BPM. Nevertheless, an elaborate infrastructure of vendor-neutral standards is required to put such theory into practice.

Where critical business processes are automated, security and integrity are obviously vital prerequisites. Support is also needed for transactions, including the so-called 'long-lasting' transactions that may take hours, days or even weeks to complete. The expectation today is that most of the Web Services 'stack' — initiated jointly by IBM and Microsoft and currently under construction at W3C, OASIS and elsewhere — will be drafted into the service of BPM. The two main server platforms, Microsoft's .NET and J2EE, already have comprehensive facilities for handling Web Services built into them. This is interpreted to mean that both are also 'BPM-ready', for today's generation of BPM products are mostly based on .NET or J2EE or both.

Web Services

Whatever BPM turns out to be, there is general agreement that Web services will be an important part of it. Dale Skeen, CTO and co-founder of Vitria Technology, Inc., is in no doubt: "I think we are at a historic moment: the Web has changed everything. At no time have we had such a ubiquitous infrastructure: TCP/IP, the Web, and now Web Services. Also open source (solutions exist). Those two trends go way beyond previous standards like OMG and CORBA". Skeen is well qualified to comment: he was previously a co-founder of Teknekron Software Systems (now TIBCO) and he is credited with the invention of distributed publish and subscribe communications. Some even regard him the father of BPM.

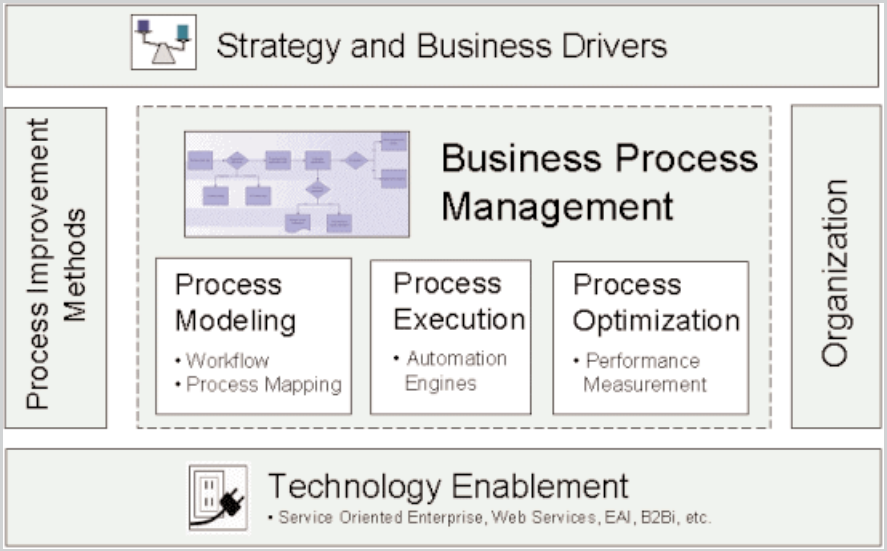
In practice Web Services are defined by the use of SOAP and WSDL, optionally with UDDI as a directory service. Security is provided by Web Services Security, now being developed at OASIS, and a number of other specifications layered on top of it. As for transactions, reliability and management, IBM, Microsoft and their partners have published proposals. These are, however, being challenged by rival draft specifications from other groups of vendors.

Java specifications

Although perfectly adequate APIs existed earlier, J2EE 1.4 is the first version of Enterprise Java to mandate support for them. Without going into excess detail, developers can:

- **create Web Services, and clients to invoke them**

Figure 2.1: Process Driven Organization



- **parse incoming XML payloads**
- **convert these to Java objects.**

So long as they comply with the Web Service Interoperability Organization's recommendations, Web Services based on J2EE can interoperate smoothly with those hosted on .NET or other platforms.

BPM specifications

The concepts of choreography and orchestration were introduced by BEA, IBM and Microsoft when they announced Business Process Execution Language for Web Services (BPEL4WS) in August 2002. The BPEL4WS 1.1 specification sums up the case for choreography in the following words: "the full potential of Web Services as an integration platform will be achieved only when applications and business processes are able to integrate their complex interactions by using a standard process integration model".

The pace of events quickened in April 2003, when OASIS set up a technical committee (TC) to discuss the standardization of Web Services Business Process Execution Language (WSBPEL). The BPEL4WS 1.1 specification was quickly submitted to this TC by:

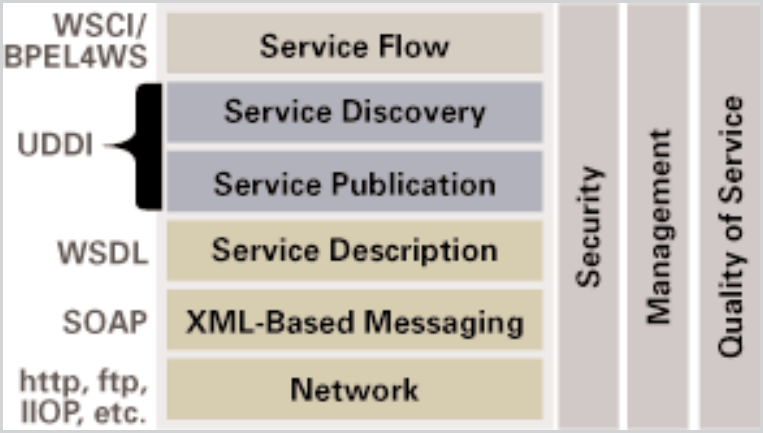
- **BEA**
- **IBM**
- **Microsoft**
- **SAP**
- **Siebel.**

Although the pioneers of Web Services must have foreseen the need for work flow, Microsoft's XLANG — which appeared in July 2000 — had the high ground to itself for the best part of a year. It was joined by IBM's Web Services Flow Language (WSFL) in May 2001, by which time insiders were already considering the potential of a merger with XLANG. On the other hand critics noted that XLANG and WSFL were distillations of Microsoft's BizTalk Server and IBM's MQ Workflow (which was previously Flowmark) respectively.

The fear was that a blend of the two might turn out to be both vendor-specific and unbalanced. BEA, IBM and Microsoft announced BPEL4WS in August 2002, together with WS-Coordination and WS-Transaction. BPEL4WS was positioned as the convergence of XLANG and WSFL, which effectively made it the only candidate to be the industry standard.

However the Business Process Management Initiative (BPMI) was quick to claim that its own Business Process Modeling Language (BPML) was 'a proper superset' of BPEL4WS — in other words, it had everything that was in BPEL4WS, and more. In an aggressive position paper dated August 2002, BPMI declared that "Microsoft pioneered the

Figure 2.2: Oracle view of BPML



adoption of the Pi-Calculus model with XLANG, IBM rejuvenated the use of Petri Nets with WSFL, and BPMI.org unified the two approaches with BPML 1.0".

IBM made the opposite claim, asserting that the Web Services Choreography Interface (WSCI) was a proper subset of BPEL4WS — and that because of this, there was nothing to gain from seeking to merge the two. Furthermore, IBM stated that BPEL4WS provides support for long-running transactions through its integration with WS-Coordination and WS-Transaction, whereas WSCI has no equivalent transaction processing support.

WSCI

BEA, Intalio, SAP and Sun had beaten IBM and Microsoft to the punch when they announced their Web Services Choreography Interface in June 2002. While XLANG and WSFL had already been in active use for two years by then, the big two did not get around to publishing BPEL4WS until two months later.

The WSCI authors stated that "[a]fter a public review period, WSCI will be submitted, on a royalty-free basis, to an industry standards body". The review period was minimal, as WSCI 1.0 was submitted to W3C before the end of June 2002. BPMI.org, Commerce One, Fujitsu, Iona, Oracle and SeeBeyond joined the WSCI authors in this submission.

BPML

The first public draft of Business Process Modeling Language (BPML) was released in July 2002. BPML is an XML-based meta language for modeling business processes Figure 2.2). It uses WSCI to define public interfaces to those processes. Yet, BPML has two practical points in its favor:

- **it provides a user-friendly graphical design interface**
- **it is also directly executable — obviating the need for a slow and error-prone application development phase.**

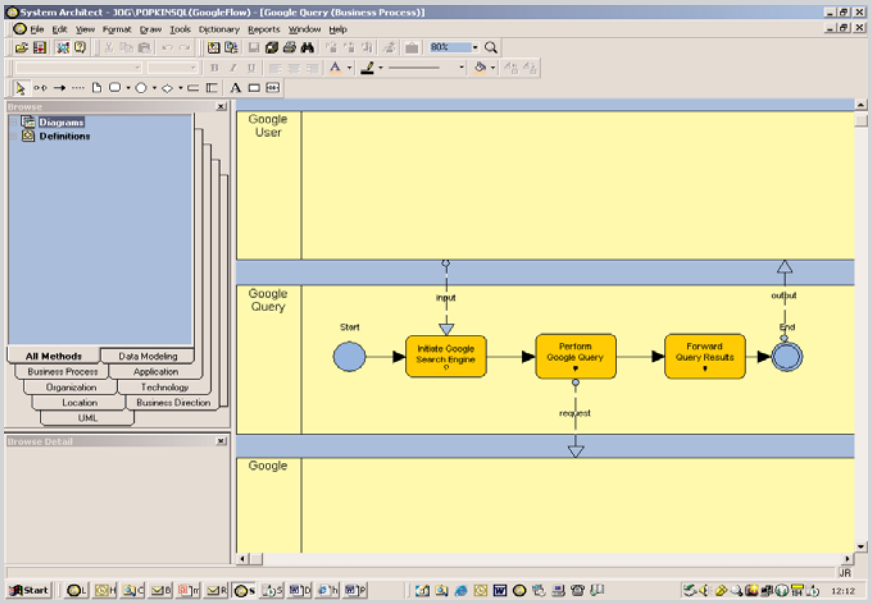
Some standards languish unimplemented, presumably because vendors do not see enough practical demand to justify the costs of development. This was not wholly true of BPML. Rather, BPML was quickly taken up by selected vendors — like Fujitsu (with I-Flow) and Popkin Software (with its System Architect modeling tool).

But the anticipated widespread adoption has simply not happened. Most reason that hard-headed product managers expect BPEL4WS to sweep the board in due course.

BPMN

Business Process Modeling Notation (BPMN) is aimed at providing a more user-friendly interface to languages such as BPML and BPEL4WS. As such, BPMN (Figure 2.3) is a

Figure 2.3: BPMN



graphical modeling notation with which business processes can be represented visually.

Its supporters believe that it is more appropriate for modeling business processes than UML, to which it can be mapped. The first draft of BPMN was published by BPMI in November 2002. The current draft is dated August 2003. It states that:

"The primary goal of BPMN is to provide a notation that is readily understandable by all business users, from the business analysts that create the initial drafts of the processes, to the technical developers responsible for implementing the technology that will perform those processes, and finally, to the business people who will manage and monitor those processes. Thus, BPMN creates a standardized bridge for the gap between the business process design and process implementation.

"Another goal, but no less important, is to ensure that XML languages designed for the execution of business processes, such as BPEL4WS (Business Process Execution Language for Web Services), can be visualized with a common notation."

Prospects for the near future

A whole spectrum of products and services is being marketed under the label of BPM. Some of these, like the products based on tried-and-tested work flow packages, can definitely deliver some part of the BPM vision. On the other hand, their vendors do not pretend to offer the full BPM 'solution' — as conceptualized, for instance, by Smith and Fingar.

Smith and Fingar's vision of the BPMS is a very different kettle of fish. They explicitly state that it is intended to 'relieve' business users of the whole burden of programming. But the only way this can happen is if the BPMS — like the RDBMSs to which Smith and Fingar approvingly compare — is completely self-contained. But in that case, any organization that purchases a BPMS will have absolutely no control over its internal workings. The BPMS, in other words, will be a 'black box' like a TV set or a proprietary operating system with large yellow stickers saying: 'Do not open. No user-maintainable parts'.

Committing a company's vital business processes to such a BPMS looks like it may prove to be a Faustian bargain. True, the organization gains the ability to control its business processes without having to be concerned about the underlying mechanisms of the BPMS. But, by the same token, it is thoroughly and completely locked in to the BPMS vendor. Two milestones must, therefore, be passed before BPMS becomes a truly viable option for the majority of enterprises.

The first is that a complete set of vendor-neutral industry standards must be agreed, published, implemented and debugged. At a minimum, this set will include the Web Services stack promoted by IBM and Microsoft (including security, integrity and transactions) plus choreography standards such as BPEL and BPMN. Vendors must then adopt and implement these standards, thus making sure that their products are interchangeable. Enabling customers to arrange for second sources is vital for credibility.

The second involves the feasibility of identifying, analyzing and specifying a wide range of business processes. These must be demonstrable. This is hardly, if at all, an IT issue. Additionally, it would be helpful if the ability to change business processes 'on the fly, yet in a controlled manner' could be shown to be practical.

For any new technology to get off the ground, lashings of enthusiasm and optimism are indispensable. So it would be self-defeating to criticize today's BPMS vendors for the imperfections of their products. However, it will probably take two or three years for the standards to be ready, and another year or two to shake out the bugs in the first generation of products that implement them.

Early adopters and innovators should sink their teeth into BPM, by all means — without them there would be no progress. But they should do so with their eyes open, having conducted appropriate and thorough risk assessments.

Management conclusion

There can be no doubt that BPM offers executives and business managers an attractive vision: the closer alignment of business and IT; direct, real time control over business processes; and, last but not least, freedom from the irritating shackles of software development. All of these are calculated to appeal and among the logical benefits would be:

- ***greater flexibility***
- ***faster response to opportunities and threats***
- ***greater effectiveness***
- ***lower costs.***

The big questions are: can it be done and, if so, how? The arguments most often deployed in favor of BPM tend to dance around these issues, often relying on tricky, almost paradoxical reasoning.

The key point to be aware of is that BPM presentations, books and reports are mostly addressed to (presumably non-technical) business managers and executives. The benefits promised are such as to appeal to that audience, and the discussion usually remains on a higher plane — well above mere technical IT concerns (see also Bernard Abramson on page 2). Thus the argument, boiled down to its essentials, amounts to 'Trust us. We will rescue you from dependence on your own IT department'.

The feasibility of this undertaking, however, can only be properly assessed by that self-same IT department. Some prominent BPM advocates have evolved a sales pitch that automatically excludes criticism by the only people who are qualified to criticize it.

BPM will eventually deliver important benefits to business. The question is: when? Each organization must come to its own conclusions. This can be done safely and effectively only if IT experts are involved — which must mean ignoring some of the hype about liberating business from IT.

Make no mistake. BPM is just one more IT specialty. It needs IT expertise to make it work. If decision makers turn their backs on their own IT staff, they will merely end up delivering themselves into the hands of some vendor's sales staff.

Client middleware reinvents the office

Amy Wohl
Principal
Wohl Associates

Management introduction

Middleware is constantly evolving. Once upon a time it was a narrow specialty for systems programmers only to think about. That is changing and fast.

In this analysis, Amy Wohl explores the latest evolution — the arrival of 'client middleware in an office context'. Although many have seen this as a frontal attack by IBM on Microsoft's Office Suite franchise, she demonstrates that it is not. Instead it is a logical and much more subtle evolution than expected.

What exists and what existed

Architecturally speaking, office work has been fairly static for a long time, content to occur mainly on desktop personal computers. For more than 90% of North American users and substantial numbers of business users worldwide, this desktop was outfitted with the Microsoft Windows operating system and the Microsoft Office Suite application, a set of personal productivity tools — including word processing, spreadsheets and presentations, and sometimes including personal data bases, web site design and other tools. This duo has such a pervasive presence that an entire generation of office workers may not realize that it is not the only possibility that exists.

Before personal computers, we used computers to assist us with office work. If that assistance was less widely employed, that was more for the higher economic cost (and lower perceived value) of putting previous generations of office or general purpose computers on office workers' desks. That was especially true when one added the fact that these earlier systems were harder to use, required more support and provided assistance across a much narrower set of tasks.

Nevertheless, in almost every era of computing — from hardwired machines (with their tasks built right into their mechanisms), to expensive but more flexible software-programmable word processors to general purpose mini-computers equipped with office software — we supported office work in the 1960s, 1970s and 1980s in some form. But this was before the personal computer became common.

Break points

We now seem to be coming to several break points at once:

- **the Internet has accustomed millions of users to the idea that data and processing can occur on a server located 'somewhere' and that this will, nevertheless, be safe and easily accessible**
- **the pervasive use of cell phones, PDAs and other small, wireless devices has begun to drive a demand for Internet access that ranges from the device in a worker's hands through to the information in the office**
- **management has become impatient with the notion that desktop PCs must be repurchased at regular (and diminishing) intervals in order to be able to run the current generation of infrastructure software and modern bloated applications.**

Perhaps it is nearing the time to 'reinvent the office'. A reinvented office would take the issues above into account. But it would also consider a number of others as well:

- **office workers do not necessarily work in offices any more — they may work in delivery trucks, cars, clients' offices, schools, hospitals, factories or farms and everyone needs information to do their job while nearly everyone needs to communicate with suppliers, partners or customers**
- **a project team may include workers in many physical locations and time zones while a vice president may be supported by an assistant in another city; connected by email, instant messaging and other collaborative tools, they may scarcely notice the distance**
- **there is a change in emphasis from the kinds of desktop productivity tools we used in the eighties (word processing, spreadsheets) to the kinds of office tools we focus on today (electronic mail, line of business applications, Internet-based applications)**
- **an increased interest in alternative (to Microsoft Office) solutions exists — especially those that come as 'Open Standards' or in 'Open Source/Linux'; this is especially prevalent outside the United States and particularly with governments and public institutions.**

Perhaps it is worthwhile to start with history

It is common for users to think that whatever they are doing now is the way things have always been (and, by extension, the way things always will be). Our memory of the past is short. Our vision of the future is cloudy. Our children, for example, cannot imagine a world without personal computers — just as we can barely remember a world without television. I suspect our grandchildren will not be able to imagine a world without ubiquitous wireless connectivity.

In other words, we have come a long way in a relatively short time. That is the effect on users of dynamic technological change.

Word processing, perhaps the first office technology, is only 40 years old. In that time we have changed its enabling technology at least four times and its leading vendor many more times (Figure 3.1). Yet Figure 3.1 does not, however, include some of the more interesting minor notes:

- **Apple's Macintosh had a certain glamour in the corporate market early on (1984-1988), based on its ability to produce print-like output; it still shows up in graphics departments and on some 'right-brand' executive desks**
- **'thin client' solutions have appeared from time to time, sometimes based on internally supported solutions and sometimes based on external service providers' offerings (like ASPs); in the reinvented office, we will see a much more vigorous thin client appear**
- **as Linux server solutions have increased in popularity and have moved into large companies and mission critical applications it has become obvious that Linux desktops will also have a significant market; Wohl Associates believes, based on research house findings and a survey we are currently conducting, that the actual number of Linux desktops far exceeds the 2% shipped with desktop hardware and that it is likely to be in the realm of 5-10%, worldwide before the end of 2007.**

What must be done for the future

But customers are also insisting that any new office solution will have to fit within what customers are currently doing in their offices. Requiring massive refitting, retraining or software rewriting is not acceptable.

This means that any important contender will have to meet a formidable set of requirements. It must:

- **provide ways to support office workers that take advantage of the cost savings in human capital available when administrative support is centralized and automated; portal-based offerings that provide software and services to users based on their organizational role are becoming popular, not least because they also are easier to support in a highly centralized and more efficient way**
- **ensure that any new approach accommodates the continuing use of popular Windows-based products (like the Microsoft Office Suite); there are too many users of this product (and its familiarity is too deeply embedded into corporate culture) for it readily to be replaced**
- **offer an environment where a variety of office workers, with a broad variety of office tasks and needs, can be economically supported — without paying too much to support casual worker usage or limiting intense workers to an unsatisfactory interface or feature set.**

Changing the paradigm

IBM's recently announced (May 10) its Workplace 2 product offering attempts to answer these requirements, especially for enterprise customers who use portals to distribute software and information to their employees. Its server-centric approach, using client middleware, is a quite distinct departure from the current architecture which is PC- and desktop-centric (Figure 3.2).

Function located on a central server is managed by IBM middleware and then distributed, via client middleware (a new concept — Figure 3.3) to whatever device the user employs. This could include anything and everything from:

- **a desktop**
- **a laptop**
- **mobile devices such as PDAs, tablets, and smart phones**
- **less conventional notions such as smart cars.**

Workplace is architected to distribute software between its server(s) and any enabled client. The key attractor is that Workplace operates in a centrally managed, server-centric world. Function is:

- **located on central servers**
- **managed by middleware**
- **distributed, via client middleware**

to whatever device the user happens to be employing. As such Workplace is designed

Figure 3.1: Evolution of word processing

Year	1964	1974	1976-1978	1982-2004
Technology	Hard-wired, Dedicated HW	SW Programmable Dedicated HW	SW Programmable Multi-User/ Mini Computer	SW Programmable General Purpose PC
Leading Vendors	IBM	Vydec	Wang	IBM with WordPro
			Digital Equipment	IBM with WordPerfect
				Wintel with Microsoft

to distribute function and information between its server and any enabled client.

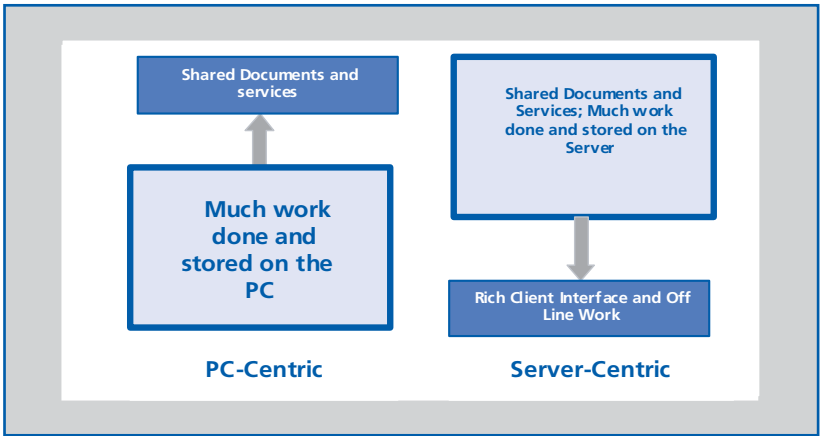


Figure 3.2: Different 'centricities'

Architecture first; then function

Let us now talk about 'client middleware'. Client middleware is an extension of server middleware. Generally speaking, middleware is that layer of software that provides functionality above the operating system and below and between applications.

We already sometimes install a kind of middleware on personal computers (think of Virus and Spam filters, for example, or Media Players). But we have not tended to think of this as middleware before.

IBM's Workplace client middleware places a layer of middleware onto desktops and other devices to enable a rich client experience (by that I mean richer than what is possible with a browser). This client middleware provides:

- **a local database**
- **Web application server**
- **provisioning agent**
- **replication support.**

Everything a user creates or modifies on his desktop is automatically stored on the server (even if it is created in Microsoft Office, for example). So everything exists in a managed environment which is backed-up and made available for search and sharing.

A micro version of the client middleware is available for lightweight or mobile devices. Footprints for this form of client middleware depend on the actual device and the functions supported — and can range from as little as 300K in some PDAs to around 20MB in a desktop device with a rich mix of functions.

Workplace Messaging and Workplace Documents

Workplace currently runs two software offerings, Workplace Messaging and Workplace Documents. These exploit the client middleware.

Workplace Messaging includes:

- **group calendaring and scheduling**
- **document and attachment viewers and search**
- **mailbox/folder archive and restore**
- **LDAP directory certification**
- **off-line mail and calendar viewing**
- **dynamic client provisioning, configuration and upgrading**
- **instant messaging (and chat)**
- **extended productivity features including self check, full text search on local mail and built-in editors.**

Workplace Documents includes:

- **key document management features — including check in/check out, version control and document level security**
- **integration with desktop applications; documents may be stored both locally and (automatically) at the server, with secure local content store**
- **server provisioning**
- **a builder tool to create new applications**
- **support for the DB2 Content Manager**
- **built-in editors.**

Workplace buyers may choose either or both of these applications. IBM and its partners will, of course, offer other applications over time. Partner solutions have already been announced by:

- **PeopleSoft**
- **Siebel**
- **Adobe**
- **Intellisync**
- **Rim**
- **and others.**

Architectural choices

There are many architectural choices for those interested in how to implement partner offerings. They may be on additional servers, or they may be provided in the form of portlets — small plug-ins, allowing existing applications to be used within the Workplace environment without alteration.

IBM has also built a portlet for the Microsoft Office Suite. This enables Office users to take immediate advantage of Document Management without changing anything about their Office application, adding automatic replication of files to the server and document management. This is an important idea because it means an organization with many Microsoft Office users can choose to implement Workplace without the need to displace its Office users. It can continue to use Microsoft Office, adding the additional functionality of Workplace Documents.

But IBM is also offering users who do not currently use Microsoft Office (or a competing product) another path. Both the Messaging and Documents applications each contain a set of embedded personal productivity components. These are available to anyone using the messaging or document management application.

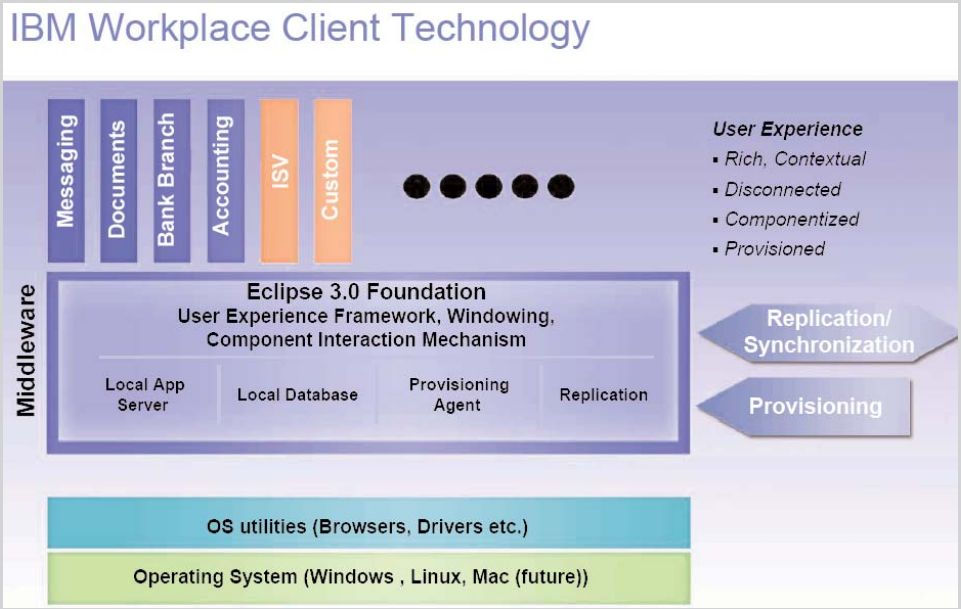
It is important to note that they are not a separate offering. They cannot be purchased for use in an environment where server support is not available. This, in itself, makes them very different from current office suite offerings from software vendors — such as Microsoft, Corel and StarOffice/OpenOffice.org. It is essential to remember this differentiation. Many of the early press reports — and even more of the comments about them — gloss over this difference and seem to think that IBM is now shipping a Microsoft-competitive office suite. That simply is not so.

Componentization and middleware

The set of componentized office applications that IBM is offering as part of its Messaging and Document Management applications is derived from the OpenOffice.org open source code. This means that the level of functionality and interoperability provided by the IBM text editor, the spreadsheet editor and the graphics (presentation) editor are equivalent to that found in OpenOffice (or Sun's StarOffice). We, at Wohl Associates, estimate that this is (today) at about 75% of the functionality of Microsoft Office for the equivalent applications, albeit increasing with each new release of OpenOffice. (On the other hand, Microsoft's Office Suite, in its various versions, includes other applications — such as a personal data base as well as note and web editors — which are not included in the IBM
offering.)

That, of course, is the point. IBM's Workplace is not intended to be a Microsoft Office Suite replacement.

Figure 3.3: Client middleware from IBM (Source: IBM)



Instead, it offers a server-centric approach to providing office function which includes componentized editors which exist only as part of the server software offering (and not as a shrink-wrapped or downloadable 'office suite'). Workplace functions are not available to an individual user, working in a PC without a server environment.

On the other hand, once these functions are downloaded to a user's computer (desktop, laptop or mobile), he or she may use them (and other elements of Workplace) in disconnected mode without continuous access to the server connection. Whenever the Workplace user is working in connected mode, additional server-based functions become available and any work (emails, new documents to be stored on the server) will be synchronized and processed. Likewise, new email will be downloaded, new information will become available and new software (or upgrades and patches) will be automatically applied.

The enterprise dimension

IBM is emphatic that Workplace is a product defined by its enterprise (large) customers who have been successfully moving to portal-based solutions to deliver applications and information access to their users. These customers have asked for improvements in two areas:

- **a richer client than the browser**
- **improved productivity applications (than those provided with a Portal).**

IBM Workplace 2.0 and the new messaging and document management applications are intended to provide that answer:

- **a richer client, provisioned and managed at the server, promises a combination of lower TCO and richer function wherein this rich client can integrate with desktop applications, offer drag-n-drop and has UI control; the customer can then choose between rich client, browser and mobile client solutions — without having to alter applications or routes to information**
- **the micro edition of the Workplace client provides micro middleware components for all types of mobile devices — supporting transactions, a Web container, messaging and a database; this offers embedded Linux support and portable JVM based graphics as well as a new set of enterprise development tools**
- **managed clients function in connected, intermittently connected as well as disconnected modes — enabled by Lotus-provided bi-directional synchronization and replication.**

As such, Workplace provides a middle ground, sitting between:

- **clients, for whom it provides collaboration services**
- **interaction and access services**
- **managed client services**
- **business context and activities**
- **back end integration to business processes (monitoring, work flows, and access to business applications)**
- **information (data, content, integration and tools for search and analysis).**

All of this occurs within:

- **a single architectural model**
- **a single programming model**
- **a single consistent tool set.**

Eclipse is the basic development environment. Its tools include:

- **a Portlet Builder**
- **a Workplace Builder**
- **designers for Domino and Workplace**
- **WebSphere Studio.**

Security is also well provided for. There is a separate client-side repository. Only applications provisioned from the server can access this database. This belongs to the client-side middleware — not to the desktop file system (and is, therefore, separate from the operating system). It is this separation — of the middleware from the client operating system — which provides the security and should significantly avoid unwanted invaders.

Starting out and where to go

Adoption of Workplace is likely to start out with generic, horizontal applications. But do not expect them to stay that way. IBM will mesh them with its solution strategies, with specific Workplace offerings in the future aimed at specific industry segments.

For example, expect a branch banking or insurance branch Workplace. IBM's Global Services has practices focused on these type of segments. It will use its expertise to provide the relevant feature set. Look for more announcements for

specific verticals in the second half of 2004 and well into 2005. Wohl Associates also expects partner offerings — built on top of Workplace — for a variety of other vertical markets.

Pricing for IBM Workplace is expected to be aggressive and competitive. IBM has mentioned that — for a large enterprise user (we guess that means multiple thousands of seats) — each server (Messaging and Documents) will be $1 per user per month plus $2 per user per month for support. That is guessing. But we assume that must mean that some multiple in the 2 to 3 times range would be the 'list' price. If those guesses are correct, one consequence of 'client middleware' may yet prove to be considerable repricing of 'office offerings'.

IBM's roadmap makes some Workplace functionality — and its associated client middleware — available now. Early customers are participating in a Beta program. Workplace Messaging and Workplace Documents — as well as Workplace Builder and the Microsoft Office Plug-In — are scheduled to be available from the second quarter of 2004. The Office components (the text, spreadsheet and graphic editors) are delivered with Messaging and Documents. Micro Builder 5.6 is available for download now; 5.7 will be available in the 3rd quarter of 2004.

From a pervasive computing point of view, customers are pushing IBM at about as fast a pace as it (IBM) can go. The problem is sorting out the pilots from real projects.

There is also a new focus on ISVs, especially helping them to enable software on new devices and new applications to broaden demand for their applications. Indeed, a good source for micro middleware information is www.ibm.com/developerworks.

Figure 3.4:

Year	2004	2007
Technology 1	SW Programmable PC plus Server	Internet/Server/ User Device
Leading Vendors 1	• Wintel with Microsoft Office System • Wintel with Linux Sun JDS	Linux or Wintel Server/ IBM or Microsoft Server SW/ Variety of Device SW
Leading Vendors 2	IBM Office Workplace 2	
Technology 2	Internet/Server/ SW Programmable PC	
Technology 3	Software as a Service	Software as a Service
Leading Vendors 3	SalesForce.com SmartOnLine	IBM

Microsoft

Because much of the change occasioned by Workplace will mean pressure on the current market leader, Microsoft, we thought it would be interesting to try to compare:

- **the IBM Workplace offering (as described above)**
- **Microsoft's Office System (the current standard).**

In order to do this, you must throw your net much more broadly than just the ubiquitous Microsoft Office Suite and consider its surrounding environment. Microsoft calls this the Microsoft Office System. By this it means the combination of its Office Suite with functionality enabled when Office Suite is used with other Microsoft products such as the SharePoint function (in Server 2003) and the SharePoint Portal. (Microsoft would also like customers to consider the functionality available in other related products such as InfoPath and BizTalk, but that may be more of a stretch.) In this context, Windows SharePoint Services (WSS) provides tools for teams to create and work within collaborative spaces using a Web browser or a rich client. Its capabilities include:

- **self-service site creation**
- **browser and office interface**
- **document libraries**
- **collaboration**
- **customizable lists**
- **document and meeting workspaces**
- **personalized Web part pages**
- **FrontPage customization.**

SharePoint Portal Server builds on top of WSS to provide:

- **My Sites and people finding**
- **targeting information at audiences (by their roles)**
- **site registry and connections**
- **enterprise search and alerts**
- **portal publishing and navigation**
- **single sign-on**
- **BizTalk support**
- **cross-portal shared services.**

Microsoft has more than 30 million SharePoint Portal users (our guess is that this includes registered visitors to Portals — it is nearly impossible to count SharePoint Service users since anyone who has the 2003 Server plus Office can use it). Most Share-

Point users use it for the functionality described above, some using their Office interface, others via a browser (Microsoft supports both).

Microsoft partners (and internal developers) also use the Office Suite, SharePoint Services and SharePoint Portal as platforms for creating additional applications and for integrating existing software. This means that, for users who want a PC on every desk, many of the functions available in Workplace are also available within the Microsoft Office System. All of this, of course, is on the Windows platform (with the Office Suite also supported on the Macintosh platform). This is a major difference with IBM and other alternative office vendors, who tend to support a broader variety of operating systems, including Linux.

Do not expect Microsoft to give up market share easily. The software giant has shown great resiliency in the fact of competition and could choose to do something substantial in answer to a significant challenge, such as porting its product line to Linux (not likely yet), or moving to a 'software as service model', which it has the cash to afford (the transition from a pay-in-advance to a pay-as-you-go model is daunting to an existing vendor). But do expect IBM, with a new product designed to fit exactly what customers want, as well as other vendors to present challenges. The office market is moving (possibly) from the PC and the desktop to a networked model in which any device, anywhere becomes the locus of office function.

Management conclusion

IBM will focus Workplace at the enterprise and the high-end of the mid-market (IBM's definition of mid-market is companies with 100 to 1,000 employees), at least for a while. But Wohl Associates expects this approach will be equally appealing to smaller organizations.

That will require new routes to market, more vertical solutions, and perhaps a hosted offering for the SMB market. That could be run on IBM's hosting infrastructure and perhaps connected to IBM's recent announcement of a hosted systems management offering for SMB PC buyers, adding office services to system management.

In any event, Wohl Associates expects IBM's Workplace offering — with its novel client middleware —to be part of a larger trend, over the next three to five years. During this time Wohl Associates expects (Figure 3.4):

- ***customers to express an interest in more diverse products***
- ***the concerns of emerging markets to be considered (where PC penetration rates are necessarily still quite low and impose less of a limit on future architectures)***
- ***additional vendors, lured by the economics, to choose to enter a newly competitive office marketplace that has been enabled by new forms of middleware.***

Coupled or decoupled plus heavyweight and lightweight delivery considerations in an Enterprise Service Bus context

Andy Stanford-Clark
Master Inventor and Senior Technical Staff Member
IBM Hursley Park Laboratory

Management introduction

The notion of an Enterprise Service Bus is being widely promoted by several vendors. Some seem to envision it as a new technology while others prefer to characterize it as the logical successor to message brokers and the like.

In this analysis, Andy Stanford-Clark — of IBM's Hursley Laboratory — looks at:

- ***what an ESB represents in both conceptual and practical terms***
- ***examines how existing products — like WebSphere MQ and WebSphere Message Broker — deliver ESB function today***
- ***explores the implications for tightly and loosely coupled processes in an ESB***
- ***considers what needs to be assessed when choosing between heavy-weight and lightweight messaging within an ESB.***

What is an Enterprise Service Bus?

An Enterprise Service Bus (ESB) describes a utilitarian concept which is applicable to most organizations which have different 'islands of information automation' that need to work together but which do not. The reason they do not work readily together is most usually because of their genesis over many years of development and production.

An ESB provides a logical mechanism which enables separate areas of information automation to work together in flexible and adjustable ways that, traditionally, were:

- **not possible without major financial investment**
- **often delivered in ways (to keep the original costs down) that subsequently proved to be inflexible when further changes or improvements or connections were needed.**

If this seems somewhat abstract — and the ESB concept can seem to be this — consider an analogy. Think of (say) a red double-decker bus in London. Its purpose is to take you from one place to another (in information terms it is to convey information from one place to another). But, whereas a red London bus has only one entrance, an ESB will have a number of different doors on it — with different entry and exit points. Indeed, each door may be of a different style and even have different sized porches — to accommodate different sizes of people (information) and the different ways they may choose to board or exit the bus.

These porches are as important as the doors. The porches 'ask' what language you prefer to speak ('what protocol you are using', for example); when this is understood, it (each porch) converts everything boarding the bus to a common form. The same applies for exiting. Thus something arriving through one porch and door in one form can leave through any other porch and door in another form.

In doing this, whatever can board can also disembark — even though the departure format and protocol may be quite different to the arrival format and protocol. In effect, an ESB enables an application which produces one form of output to submit this to an ESB — and know that other applications will be able to receive what the original had created as input as an output, without the originating application having to understand the destination application.

But that is not all an ESB can do. Again, return to the red London bus. It has two decks (floors). The lower deck is where all the various ESB doorways and porches are. It looks after boarding and the rendering of information arriving into a common format as well as subsequent disembarkations. But more may be needed. By paying a little extra and going upstairs to the upper deck, additional degrees of processing can be applied — from persistence to reliability to transformation to routing to encryption to rules and/or content-based processing and other such value-adding services. You may not want all of these all of the time. But certain passengers (or their patrons) may choose to exploit these when necessary.

An ESB in practical terms

Ideally, an ESB is a modest sized piece of software which can be deployed in multiple instances around an organization, or even between organizations. Each ESB instance is relatively lightweight but with the inherent capability to self-organize so that two or more ESBs can link together to form a logical interconnection bus along which information can flow as required.

In contrast to a red London bus, which proceeds along fixed routes and at set times (traffic permitting), an ESB's value increases as it provides ever greater flexibility. By being able (in an ideal world) to install an ESB instance at any point convenient to an application (that needs to talk to another application), the combination of self-organizing ESB instances delivers the capability for anything boarding at one ESB instance to be delivered via another ESB instance; the underlying communication connection between all the ESB instances is delivered via a LAN or WAN or combinations of these. Furthermore, ESB instances can be sized and located to suit specific requirements — at functional or departmental or enterprise or inter-enterprise levels.

The key factor that an ESB must deliver is the linking of multiple ESB instances. The reason that an ESB is a logical concept (that is implemented in software) is that it delivers its value by being able to be operate across a network cloud — which provides the interconnections between applications and systems. Too often in the past, these connections either had only been able to work together with expensive and rigid point to point connections or had remained in isolation — with people as acting as the 'middleware', the connecting mechanism between applications. The attraction of ESBs is that they can provide not only automated entrance and exit doors — across a logical network — but also the additional upper deck added-value.

Another way of illustrating the attraction is where there might be one particularly 'strange or rare' entrance/exit door which need only exist once in an organization even

though its outputs are used by many. This can be accommodated once in an ESB cloud but is still available to all other possible users.

Similarly, for the upper deck's additional value-added services, it is not necessary to locate every transformation or routing instruction on every ESB instance. Instead high volume value-added services might be replicated in many places for performance or throughput reasons while other value-added services might only be located in one or two places which could then be accessed via the self-organizing ESB instances.

The inherent advantage of an ESB is, therefore, that the location of all ESB instances can be arranged in a topology that is optimized to particular requirements. This can be at the functional, departmental, enterprise or inter-enterprise level — or combinations, as needed. Furthermore, this provides the capability for delivering work-load balancing, scalability and resilience. By implementing enough ESB instances, you introduce operational as well as logical flexibility where it is needed. In so doing, the rigidities of (say) dedicated point to point connection (for integration) are subsumed into the broader ESB concept without losing the capabilities that traditional point to point solutions had delivered.

ESBs and the physical dimension

To talk of 'an ESB' is correct, but can (unintentionally) be misleading. As described above, an ESB is a collection of physical ESB instances which self-organize to work together, to know about each other.

What many find confusing is the not well described contrast between:

- **'a logical ESB'**
- **'many physical ESB instances'.**

You do not buy 'an ESB'. You buy the software for each 'ESB instance'. It is the combination of many software instances which are working together which delivers the ESB concept.

Conceptually, the purchase of one ESB instance can provide a logical ESB. But the practical nature of a successful ESB is that it will become mission critical. Although one can implement one huge, single ESB instance — running, say, on one large centralized system — and possess thereby a logical ESB, this may not make operational sense. If that instance fails, so do all the connections that flow through it. It is just as if there is only one red London bus on a route and it breaks down; that route ceases to carry passengers until the bus is repaired or replaced.

In practice, it is likely that operational implementations of the ESB concept will be made up of several or many individual ESB instances. These will be located around a network. Each will possess:

- **different entry and exit combinations of doors and porches**
- **upper deck value-added services**
- **the ability to communicate (between the various ESB instances).**

The many ESB instances combine to become a flexible, connected intermediary. For some organizations it may be most appropriate to accomplish this by incorporating ESB instances near centralized facilities. Others will prefer distribution of instances and facilities while some will opt for that particular combination of these extremes which best suits their business requirements.

For this is the beauty of the ESB concept. Physically it can be implemented to deliver what you need. Logically, it should be a self-organizing group of physical instances which deliver the ESB concept. For example, in broad messaging terms, it can support:

- **for boarding, the push model ('I want to board'), and the pull model ('the driver will stop and tell me to board, now')**
- **for exiting, the pull model ('I want to get something off the bus') and the push model ('the driver to tell me when, where and how to disembark)**
- **many variations in between — from publish and subscribe to point to point to Web Services support as well as synchronous attachment and disconnection.**

The 'Service' in an ESB

The term Enterprise is well understood. Similarly, the notion of a 'network bus' has been described above. Often one other problem emerges when people try to understand what the term ESB means — namely what does 'Service' encompass? Possibly the simplest way forward is to convey that there are three complementary dimensions to 'Service' in an ESB context

The first refers to the notion of a 'service' or well defined, stable loosely coupled communications interface provided by an application or software component to other applica-

tions running within a business process. This aspect of 'service' is elaborated with the notion of a Service Oriented Architecture (SOA) as a means of building flexible business systems (see also www.ibm.com/software/solutions/webservices for additional information).

The second involves the 'translation service' between different protocols. In a sense, this translation service is an elementary (but sophisticated) part of the 'first hop onto, and last hop off' an ESB. Once something arrives on the ESB (whether via HTTP or MQTT or WMQ or JMS or ...) it then needs to be translated into the selected 'esperanto' which operates on that ESB — perhaps XML or eSQL or whatever else might be appropriate.

The important point is that an ESB should not force upon you the need to do translation in a certain way, that (within reasonable limits) you are free to do more or less what you want. Choices are not imposed upon you. One dimension to 'Service' is that an ESB offers the choice as to what form a 'message' should arrive in; the ESB's inter-translation service provides the conversion into your chosen generic format as well as the translation necessary to go out to your chosen destination.

The third dimension to 'Service' is that associated with the upper deck added-value possibilities. These services are relevant when information proceeding through an ESB needs additional work performed — like:

- **persistence**
- **reliability**
- **replay (in terms of archiving)**
- **transformation (in the programmatic sense — like Fahrenheit to Centigrade, Kilos to Pounds, etc.)**
- **doing joins with databases (as you can do in IBM's WebSphere Message Broker) for looking up name and addresses from, say, a customer number before adding this data to the original).**

The essence of this third dimension to 'Service' is that activities can be automated on your behalf. Rather than have a person receive an email order and then have either to print it several times or create and send several copy emails in order to distribute an order to many required destinations, an ESB automates the repetitive and mundane.

Similarly, added-value services can automate persistence or even 'filing'. An ESB service can provide either — whether in a database or working with an email server or whatever.

In essence, an upper deck 'Service' in an ESB undertakes activities than an individual could do (or has been doing). Instead an ESB can perform where it makes little sense to have a person involved — with the caveat that there is a distinction that should be drawn between the processing that:

- **an ESB broker delivers, where the logic to be applied is intentionally lightweight and repetitive**
- **an application server (like the WebSphere Application Server) offers, where the logic is heavier duty.**

As ESB is not a substitute or an alternative for the heavier duty application logic that an application server is designed to deliver. An ESB should not try to do the work of an application server. Equally, an application server may be overkill for ESB-type activities.

An ESB should facilitate the supply of data to and from the applications running on an application server. Another way to characterize what should be on the ESB is to work out if the processing is 'on the fly' or not: if it is, then it is probably appropriate for an ESB. If it needs more work, like interacting with a customer, then that element should be handed off to an application.

Tightly or loosely coupled considerations in an ESB environment

The key distinction is summarized by the differences between, for example:

- **WMQTT**
- **WMQ or WMQe.**

In a tightly coupled, or a synchronous program, model, the application has to be aware of whether it has a connection to the far end — or not. Such awareness may be good or bad. Indeed the people who think it is bad are usually adherents of the loosely coupled or asynchronous model (or vice versa).

However, what IBM has found with customers is that the key distinguishing element boils down to the difference between attended and unattended operations. If you have an unattended device — like a telemetry device — it is valuable (and often essential) to know if connectivity exists. The application needs to do more than simply send or receive data which will one day arrive. For example, in a factory, if a temperature reading goes over a certain level, then it may be more than desirable that someone is informed so that

actions can be taken. It is of little use if the warning is sent but only received at some point in the future after the plant has caught fire or in a way that does not facilitate an appropriate action.

In attended operation, the need is to try the existing network connection. If this is not working, then alternatives need to be tried until either success or disaster occurs.

Part of the price paid, for synchronous or tightly coupled, communication is that more pain is placed upon the application developer — who has to be aware of all the possible different scenarios and then create solutions which address the selected ones. This makes applications larger and more complex, as different combinations and retries are attempted via the sequence coded into the application. The first connection might be via the normal internal LAN, the second via a dial up connection, the third by satellite and the last via GPRS/G3 or some form of other final resort communication.

The issue here is that handling all these alternatives produces much extra work for the application writer. But, if you are an engineer working with production plant and machinery, this is exactly the kind of thought process that is necessary. After all you do not want a refinery to leak, a chemical plant to blow up, a food processing system to produce contaminated products, etc.

In these situations you do not want to hand 'messages' to an opaque middleware layer where you do not know when information will arrive. Instead you want to process the information through a transparent middleware layer where there is sufficient control to know for certain that delivery has been made.

In this context, one service that an ESB must offer in some form is the ability for those who want to take direct responsibility for their data to be able to do this. Any ESB must ensure that it is able to provide tightly coupled or synchronous capabilities to address such requirements, even if that means that the responsibility for the action to be taken is passed to an external (to the ESB) application which is being monitored by a person or is prompting an action to be taken by a person.

Having said all this, one issue that does require thought is whether the status of the connection needs to be known 'across' the ESB. For example, a source application might deliver a message of some form in a synchronous or tightly couple manner to an ESB like the WebSphere Message Broker (WMB). However, once inside the WMB, the tightly coupled link (even if it is from a synchronous protocol like MQTT) is effectively lost once WMB has accepted the input. Should there need to be a continuous connection across the ESB/WMB from source through to destination(s), then additional considerations will apply like the destination application sending back a confirmation message along the 'the reverse route' saying I got your message.

Alternatively, a different solution might apply (for example using the WebSphere Application Server to behave like an ESB).

Loosely coupled or asynchronous considerations in an ESB environment

The attraction of decoupled, or asynchronous, solutions is that they reduce the amount of work which application developers have to perform to make their solutions behave as intended. Rather than having to consider every circumstance, and then coding for the relevant ones, decoupling enables different parts of an overall process to be created by different activities. Indeed, the application that is the source need not know anything about the nature of the destination application (and vice versa). This has always been true with WMQ, which is one reason WMQ is so popular in large enterprises.

Adding an ESB introduces still greater flexibility. A source need only know about an ESB (like WMB) — and not all of an ESB's destinations. Similarly a destination does not need to know about individual details of every source. The ESB is the destination as far as the source is concerned (or the source where a destination is concerned).

The ESB, once it has received a submission, can then look after routing to the one or many destinations to which the source information should be sent. The ESB's processing capability can exploit the 'upper deck' services for enhancement, encryption, persistence, etc. The destination applications then receive what they are sent by the ESB (through whichever outbound doors and porches are applicable).

Another of the decoupled services that an ESB can offer is confirmation of delivery. The source can receive a response or message back indicating that delivery to the ultimate end point has occurred (although, to be fair, for pub/sub this can become a non-trivial exercise). This is important. Applications do exist which need an explicit confirmation that the message has made it through from source to ultimate destination(s) — in other words that the middleware has worked as intended.

As such, decoupled environments are ideal for solutions that do not want or have to handle large amounts of

confirmation. The advantage is that the pain of worrying about the sending of messages is removed.

The ESB and its related middleware do this. This applies particularly to the unattended application space. Instead of spending time writing programs about what a user will do (as is the case with attended applications), the focus can be spent on sending the data. This is applicable to applications talking to each other, for example a PDA transmitting the details of an insurance claim or a laptop placing an order.

In such instances, the key issue is usually not whether the claim or order has been submitted 'yet'. Rather it is that there is confidence that it will occur and reliably. As far as the user is concerned, the successful act of pressing the 'submission button' means that person has completed his or her part; now it is up to the middleware and ESB to ensure the rest happens.

In effect the application gives the message to the middleware. The middleware hands it to the ESB. The ESB determines what should happen next — using, for instance, the content routing logic in WMB — and then assures delivery to the appropriate destinations. From the source application viewpoint, no more needs to be known than the verb to 'put' to the middleware — from which all else happens as the message finds its way to its destination.

In contrast to the tightly coupled case, users in a loosely coupled example must take more responsibility. The infrastructure is there to support the will of the programmer in tightly coupled instances. In the decoupled case the programmer expects to hand over transmission and delivery responsibility to the infrastructure which, via something like an ESB, is capable of handling this.

Lightweight and heavyweight messaging considerations in an ESB

A first step here is to understand the 'light-weighted-ness' or 'heavy-weighted-ness' of a message. With this complete one can determine what one should do.

Different communication environments require different solutions. On a modern enterprise LAN you will likely have gigabit Ethernet. If the task is to send a message about (say) a $500M transfer, it is really, really important that the message arrives and arrives once and once only. To achieve this you would use 'heavyweight' messaging because it has lots of facilities associated (from persistence to once only delivery to logging to ...). Even though this message might take only an infinitesimal proportion of the gigabit Ethernet, and even though there might be a huge overhead associated with relevant protocol exchanges, the value of the transaction is so great that assured end to end delivery in the correct manner is paramount. To lose or duplicate such a message could be hugely expensive. Yet the network cost is trivial, the delivery is high speed and there is low latency.

In this instance, therefore, enterprise strength messaging is practical and desirable. It can be heavyweight even if the actual requirements are lightweight because the network cost is negligible within the context of the enterprise WAN/LAN. Furthermore, any messaging features that 'increase the weight' — longer fields, more informative date stamps, space for future enhancements, etc. — will not materially add to the cost per message. This cost may be entirely justified by the operational savings achieved through a consistent approach to messaging.

Indeed, in retrospect, most messaging has been heavyweight in the past — because it operated within an enterprise environment where the networking cost was not significant. On a per message basis this has meant, perhaps inevitably, that the characteristics and requirements associated with heavyweight messaging, including a degree of over-engineering because the additional cost was so modest — have been accepted as the norm. Such habits can prove expensive as well as inappropriate if the real need is for lightweight messaging.

This leads to a parallel conclusion. The need for lightweight messaging may, in practice, be more common than has generally been appreciated in the past. Why is this? It is because lightweight messaging is applicable to a wide range of application scenarios — even if it can be more difficult or expensive (through simple unfamiliarity) to achieve.

Lighter-weight messaging

The advent of the Internet plus the arrival of pervasive messaging changed the overall picture, in favor of lighter-weight (and lightweight) messaging. Across the Internet, bandwidth tends to be limited at the end points. In addition, the basic nature of the Internet involves multi-hop connections from source to destination (you only have to do a 'trace route' to understand just many hops are involved).

This means that if you are sending small messages there is a large overhead per message associated with submission through to delivery. This is both expensive and is not acceptable or desirable — not least because you have to wait for so long: a 'verbose protocol' means that the

latency is much higher as the verbosity has to be handled at every one of the Internet's many hops.

Limited bandwidth and minimizing latency mean that using a heavyweight messaging protocol may not be appropriate. This is not to say that heavyweight messaging is not possible over the Internet; but it is frequently less than optimal. Put another way, the assumptions that continue to underpin WMQ in an enterprise WAN/LAN environment — such as exactly once delivery in all circumstances — are not necessarily appropriate to many Internet connections.

So, if there are circumstances where the full rigor of heavyweight messaging is not needed, then lighter-weight alternatives, with less capability (and overhead) are desirable. This is often the case when a human operator is able to deal with communication errors. For example, the whole Web experience — like buying on Amazon or buying and selling on eBay — has shown that losing a connection is not the end of the world: you just reconnect and start again. Similarly, in a securities trading environment, losing a price tick does not matter if another one is coming along shortly after. 'Just enough is good enough.'

The Internet creates opportunities to communicate with tens or hundreds of thousands of applets running with Web browsers. For such applications there is a need for lighter-weight protocols that can be scalable as well as use less resource. (Heavyweight messaging tends to be less scalable because of the size of messages plus all the protocol handling plus logging and enabling of persistence and automated recovery, etc.) Instead, on the Internet, scale counts. For example, 450,000 people may want to watch live Wimbledon scores. They want the latest score as soon as it happens, as soon as possible. Yet, if they miss a point, it does not matter that much because the next point scored will supersede the previous score — moving from 40-15 to 40-30 (say).

To obtain rapid delivery, in as near real time as the inherent latency of the Internet permits, the need is for each piece of information being transmitted to have as little overhead as possible. The lighter-weight the message (if possible within one packet), the less time it will take to transmit across the Internet, plus any bandwidth constraints at the receiving end (like a a 33Kbs or even a 9600 baud modem) will matter less. Yes, there is a chance that a score or a stock tick will be delivered twice (or not at all) but the next update will supersede the duplicate or the message that did not arrive.

This is why the WMQ Real Time (WMQRT) protocol was created to enhance the capabilities of an ESB. It enables lighter-weight messages to be created and received — without the impositions required by its heavier-weight brother WMQ. Now it is possible to publish one message to the Internet which might then be seen by 10,000 or 450,000 or 10,000,000 people. Not only is this more efficient for the publisher (who does not need to create 10,000 or 450,000 or 10,000,000 individual addressed messages for transmission) but it is better for the Internet as a whole. Wimbledon, the Super Bowl, the Masters, the World Cup or even the Olympics do not have to swamp the Internet.

Lightweight messaging

If that described 'lighter-weight' messaging (compared to traditional heavyweight messaging), there is then lightweight messaging. This is driven primarily by the existence of pervasive devices that are installed in anything from plant and machinery, cars and trucks, sensors on transmission lines or pumps and even in domestic products (refrigerators, air conditioning, oil tanks, etc.). The key difference here concerns the different qualities of service and scaling — with the additional problem that the available bandwidth can be both low and very expensive. Typically, a 9600 baud modem is pretty much the industry average if satellites are being used (from remote or inaccessible locations where telephone lines are not available). The cost may be as much as 5c/byte transmitted or received.

The good news here is that the data to be sent or received tends to be much smaller — not a person's banking profile of several thousand bytes but a simple command to turn something on or off or a report of a temperature reading or an alert that a known condition has changed. This means that the data to be sent is small. It also means that the overhead associated with sending it needs to be as small as is practical. There is little point in sending 2 bytes of data if this has 10Kbytes of routing and other overhead.

As already discussed, one of the attractions of an ESB is that it can support the capability to receive from and send out many different forms of messaging via its differing 'doors and porches'. In this context of lightweight to heavyweight messages, the following represent the minimum message header sizes:

- **for WMQ Telemetry Transport (WMQTT): 2 bytes**
- **for WMQe: 18 bytes**
- **for WMQRT: 24 bytes**
- **for WMQ: 850 bytes.**

If a satellite or similar connection is going to cost US$0.05 a

byte, the attractions of WMQTT or WMQe message are all too apparent — if the other associated constraints are acceptable. In essence what an ESB enables is an enterprise to choose what to use where and to obtain the best of all relevant options. The actual choice between lightweight, lighter-weight or heavyweight messaging is going to emerge from a consideration of function as well as the underlying residence and bandwidth of the various communications options. But, with an ESB, combinations of options are possible — which is why flexibility is improved.

Management conclusion

It is true that, with lightweight messaging, you cannot do everything: you cannot readily (say) bracket transactions across multiple messages — although it is possible to program ways to address such a requirement when using lightweight messaging. Equally, heavyweight messaging for delivering tennis or golf scores or for carrying traffic from remote sensors via satellite is too expensive in practice (although again it is possible). The point about an ESB is that the best of each form of messaging can be selected and combined to produce solutions which do not demand that expensive, custom development or rigid infrastructure is always required.

Indeed, using an ESB you could even have most of your messages being lightweight but, when the occasional 'important' or 'valuable' one appears, it is the ESB that takes the decision to communicate it — because of its content — via a heavyweight mechanism like WMQ rather than WMQTT. For that is the beauty of an ESB: it enables flexibility.

Similarly, the loosely and tight coupled can be brought together as required. ESBs enable different styles and approaches so that what each organization immediately requires can be satisfied in the first instance, and then modified or improved as business circumstances evolve or change. Equally, the notion that 'an ESB' is an overall concept made up of one or many physical ESB instances which work together to automate what previously required human intervention is attractive. While an ESB does not try to do everything, and should not (there are levels of business processing above an ESB — like work flow and process choreography, as found in WebSphere Business Integrator), an ESB can automate the routine integration of applications and systems and sensors that was not possible before.

This is why many argue that an ESB is the full expression of the true value proposition of middleware — it is the intermediary which delivers flexibility between applications. An ESB enables integration (between applications); this is what most organizations are looking for because they can no longer tolerate their existing and new investments in IT being unable to work together unless people are involved. If systems and applications can work together to do the repetitive and regular, that frees up people resources to concentrate on higher skilled activities with higher returns.

In more technical terms, introducing an ESB removes the need to worry about the nitty gritty detail of how connections are made between applications. Just as TCP/IP and its protocols are the unsung hero of the Internet, which routes all your requests (whether from a browser or email or instant messaging) to the selected destination, an ESB moves this to a higher and more sophisticated plane. The broad concept is the same although what is produced should be that much more valuable.

With an ESB, organizations start to eliminate the common scenario today — of static links which have to be administered (from definition through configuration and maintenance) physically by people. This is too expensive as well as being too inflexible to meet the business needs of today.

Instead an ESB is the realization of the value proposition where places function above the wire connectivity (TCP/IP) in order to provide increased and automated dependability and flexibility between applications.

Semantic middleware

Dr. Keith Jones
IBM Software Solutions Worldwide

Management introduction

Middleware is often said to capture the best of the IT industry's thought leadership and to be the lever by which application development productivity is improved over time. The increasing numbers of infrastructures now being deployed using middleware that supports standard XML data exchange and standard WSDL services is evidence of one of the more recent evolutionary phases.

*In recent months the World Wide Web Consortium (*www.w3c.org*) — under the direction of Tim Berners-Lee — has recommended two new standards in support of a 'Semantic Web'. Such a Web will provide a common framework for allowing data to be shared and re-used across application, enterprise and community boundaries.*

The first standard, RDF, will be used to represent information and to exchange knowledge in the Internet. The second, OWL, will be used to publish and share sets of terms called ontologies — supporting advanced Web search, software agents and knowledge management.

In this analysis, Keith Jones reviews the growing interest in semantics and asks what it might mean to build infrastructures in future using 'semantic middleware'. Such middleware would support applications on the Semantic Web. But it might also enable a new class of enterprise business applications that automatically understand the meaning of data.

Middleware is everywhere

There are several popular perspectives on what constitutes middleware software — the most frequently encountered are shown in Figure 5.1. The traditional perspective (bottom center) shows middleware to be the layer of software between application logic and underlying operating system logic. This probably dates from the days when most commercial IT systems were largely monolithic and standalone.

The client/server perspective (top left) shows middleware to be the logic that exists between presentation logic on client machines and data access logic on server machines. Much of this middleware was actually running on both client and server machines but later became concentrated on LAN-connected application server machines shared by clients.

The more recent distributed computing perspective (top right) shows middleware software to be the logic that not only supports applications in the traditional manner but also supports sophisticated interactions between application components connected in a network. This 'vertical' middleware has been exemplified first by DCE, then by CORBA and more recently by open standard Web Service-oriented software products such as the application server with J2EE or .NET.

Evolution

The richness of function now provided by middleware products has been accumulated over many years as new paradigms, architectures and technologies have emerged and been deployed. It is often said that middleware software in its many forms has captured the wisdom generated by thousands of academics, research workers, developers (open source, government, corporate and vendor) and thought leaders across many industries. This accumulation continues today at a seemingly never ending pace.

Throughout an evolutionary period of about 40 years, the principle objective for middleware designers has been to simplify the work of application developers by:

- **systematically abstracting (out of the business logic) all configuration, security, transaction and other generic concerns**
- **providing libraries of re-usable function that can be invoked easily using standard APIs.**

In the latter case, this function would otherwise have been re-coded time after time at tremendous cost both in time and money to industry at large. The general principle has been to 'push down' into middleware any logic that is not directly concerned with implementing specific business purpose. This principle remains as relevant today as at any time.

Whilst not a formal process, there is a well-beaten path taken for many middleware capabilities as they are developed and incorporated into vendor packages. That path often starts in government or academic institutions, with research work on challenging problems and with early solution prototypes (DCE was a good example).

Experience with early vendor products, open source projects and pioneering enterprise deployments of closely related solutions now often fuels an open standards process. Middleware software later emerges with standard function incorporated that can be re-used by application developers.

Semantic buzz

For many years computer scientists have worked to apply computers to challenging linguistic and AI problems involving semantics. Recent events in the IT industry have propelled recognition of this topic to the point where it has become relevant to discuss the possibility of semantic middleware in support of future commercial applications.

The standardization of XML and its incorporation into current generation middleware products has provided one major motivation. The vision of a 'Semantic Web' as described by W3C Director Tim Berners-Lee has been another. But just what are semantics? And what might semantic middleware be?

By the dictionary, semantics is the study or analysis of relationships between linguistic symbols and their meanings. Figure 5.2 illustrates some of the possible relationships between a '*customer*' at an '*address*' and his '*order*' for an '*item*' in a simple business language scenario (although many other details have been omitted for simplicity). But a similar diagram could have been drawn 25 years ago by any application developer writing COBOL programs for an order processing system.

In those days of procedural COBOL applications — and more recently in object Java applications — an understanding of the linguistic concepts named '*customer*', '*address*', '*order*' and '*item*', their attributes and their relationships were implicitly built into the program logic. So, for example, when a raw file record or networking message was received (from middleware), the application logic automatically associated values received in a certain sequence with specified data types and the concepts needed. The logic applied was then based on the relationships between the

concepts, their properties and the actions required to progress the business process.

The application program in this case assumes the meaning of the values received in the context set by the business process. But the semantics have effectively been hard-coded into the application logic and, to some extent, into the infrastructure configuration supporting it. But these semantics cannot be re-used without recoding them into another application.

The promise of semantic middleware is that interpretation of the language used in a business process can be 'pushed down' from application development tools through run-time application logic into supporting middleware. In so doing a new, higher level, lower cost solution is achieved.

The evolution of ideas that leads to this promise is extremely interesting:

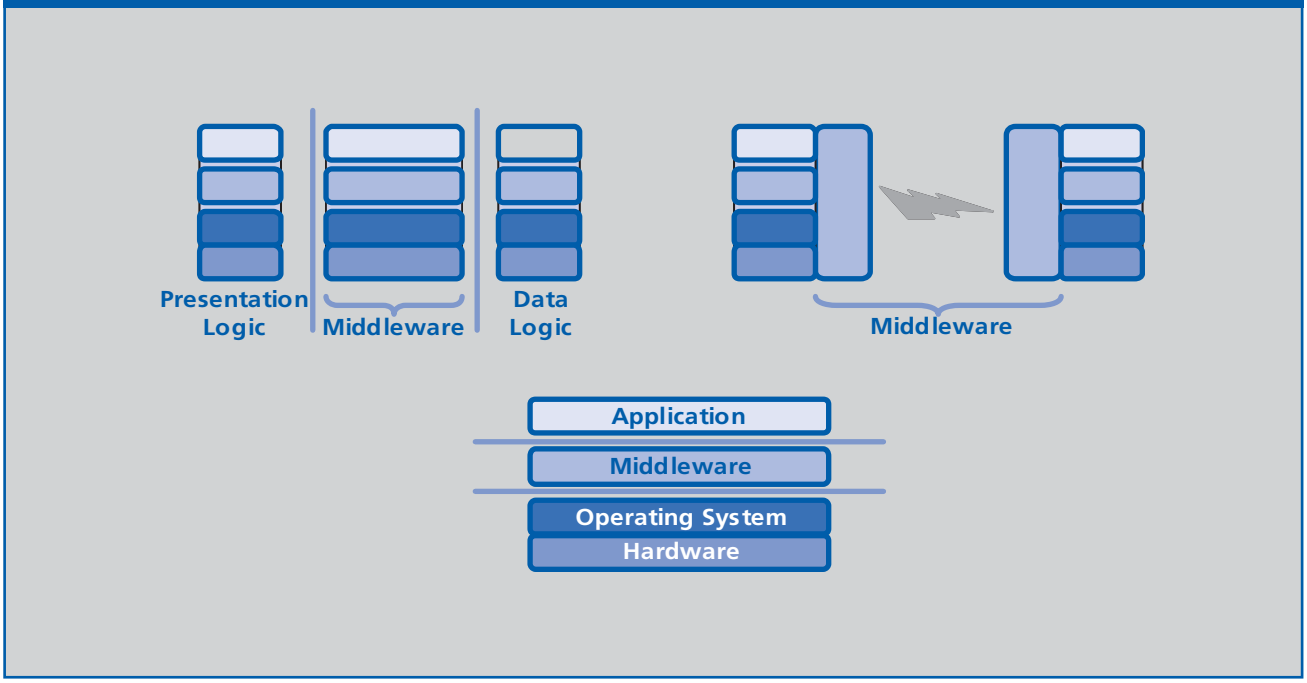
- **development tools have evolved from simple editors and compilers into sophisticated graphical modeling systems that generate application code based on business process concepts, attributes and relationships modeled**
- **runtime capabilities have evolved from simple data processing into sophisticated virtual machines supporting dynamic interpretation and introspection of program data**
- **database tools have evolved from simple record management systems into sophisticated data warehousing and query systems based on assigned and derived meanings for values stored in a wide range of managed data sources.**

Data about data

Most of the data flowing in middleware infrastructures today is described — by the modeling and semantic communities — as 'instance data'. At best, this is data that is strongly typed and precisely defined in specifications held within development file systems, network messaging repositories or database dictionaries and schemas. Semantic middleware promises dramatically to improve on this over time.

Many middleware products have recently introduced support for XML. Some also support XML Namespaces, Schema and Style-sheets. IBM's WebSphere middleware products, for example, support the use of XML in end to end business scenarios for tagging instance data coming from back end databases or legacy systems on its processing path to users' Web Browsers. Tagging of instance data using a language like XML is often seen as the first step on the road toward semantic systems. It adds descriptive information about the data flowing to that data as it is processed.

Figure 5.1: Middleware perspectives



The W3C XML standards provide clear rules for syntax and structure of instance data documents and for building vocabularies of descriptive terms used in tags. It is, therefore, possible to ensure that instance data tagged with XML:

- **is well-formed (obeys the rules for structure)**
- **uses recognizable tag names taken from vocabularies defined for specific enterprise or industry usage.**

Note, however, descriptive tags do not reveal the semantics of the instance data.

The next step is to ensure that such documents also conform to schemas that prescribe primitive data types and rules for sequencing and composing data elements into higher level reusable types. The use of XML Schema Documents (XSD instances) to complement tagged instance data (XML instances) is a standard mechanism to validate such conformance. Leading middleware tools and runtimes already provide this capability.

XML schema instances contain data-about-data (meta-data) about XML instance data (Figure 5.3). These schemas are used by middleware to ensure that instance data is properly typed and conforms to rules specifying appropriate combinations of data elements. For example, in the earlier simple business language scenario (Figure 5.2), an '*order*' might have one or more line '*items*' and be associated with exactly one '*customer*'.

At the highest level, a meta-model specifies language rules for the meta-data. In the case of XML schema instances this model is the W3C recommended XML Schema Language. But in other scenarios it might be a different language that defines the meta-data, such as ISO/IEC SQL to describe the contents of JDBC row set instances in J2EE systems or WSDL for Web Service descriptions.

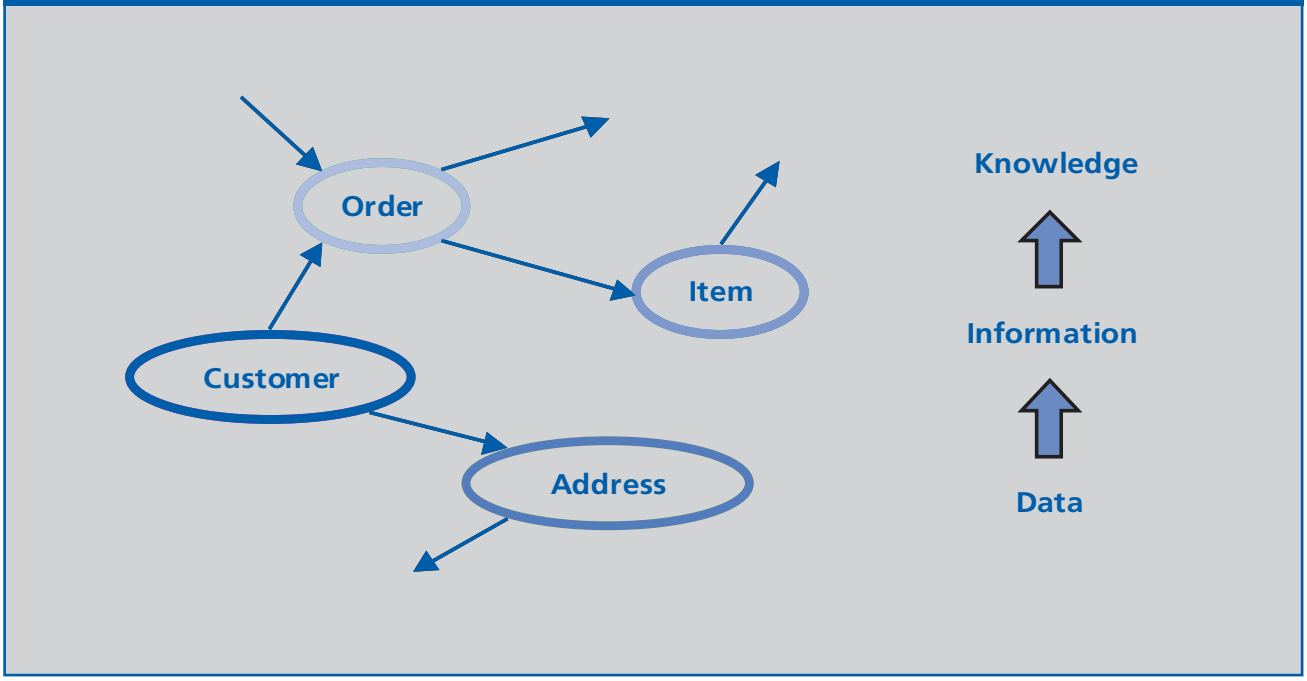
Semantic layers

The descriptive tagging of instance data and conformance to meta-data schema, supported by current middleware products, is the lowest layer in a 'stack' of semantics envisioned by the W3C for future Semantic Webs (Figure 5.4). Much of the data currently available to World Wide Web users is already tagged using HTML or XML.

The problem is that the meaning of the tagged instance data cannot be inferred from the tags. To solve this problem a new language has been proposed to capture the meaning and to open up the possibility of intelligent search capabilities and sophisticated Web Services.

The Resource Description Framework (RDF) and its corresponding Schema Language (RDFSchema) is the solution recently recommended by the W3C. Using RDF it is possible

Figure 5.2: Semantic relationships



to provide simple semantic information about XML instance data — in a standard interoperable way. RDF is a practical and mathematically precise foundation upon which the Semantic Web will be built.

In essence, RDF provides a simple but powerful data model for expressing knowledge about instance data elements that are exchanged between applications. The data model allows knowledge to be expressed as triples relating:

- **rdf:subject resources**
- **rdf:object resources**
- **rdf:predicate resources.**

A resource, in this context, is any object that can be identified by a URI. This, by IETF definition, is anything at all — whether accessible via the Internet or not.

Each RDF resource is described by its properties (rdf:predicate) and property (rdf:object) values. Each property is also a resource and so may be described, in turn, by additional named properties and property values. The result is that an arbitrarily complex graph of nodes and arcs representing resources, properties and values may be constructed to represent knowledge about any given instance data — a straightforward and powerful framework for exchanging semantics between applications.

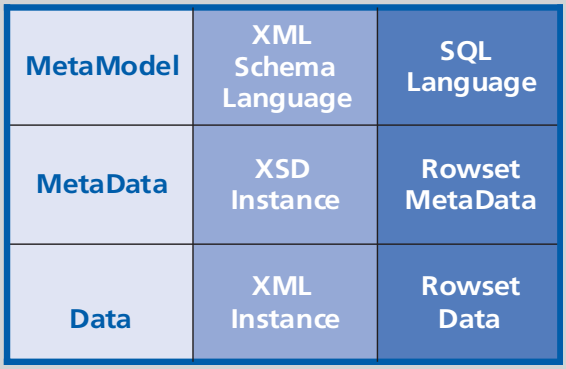
In the simple business scenario (Figure 5.2) each of the '*customer*', '*address*', '*order*' and '*item*' concepts could be identified as RDF resources. The meaning of the arrows connecting them are identified as RDF properties. Note that '*address*' is an example of a property (of '*customer*' in this case) which may also have meaning as an independent resource with its own properties. An RDF instance containing these semantics could then be associated with XML documents containing business order values.

Property values in RDF are identified by their type and position within a corresponding well-defined value space. RDF types may be simple XML schema data types (for example, xsd:integer) or higher order types with well-defined value spaces. Semantic middleware and tools will be able to:

- **recognize these types**
- **automatically interpret their precise meaning in terms of allowable range checking and operations.**

RDF properties and their expected values are also extremely important as it is expected that common vocabularies of meaning will be defined for a wide range of business, scientific and governmental purposes. The RDF Vocabulary Description Language (RDFSchema) has been specified to make this a relatively easy task. The RDF Schema language provides a type system for RDF instances which is similar to the type systems of some object oriented programming languages such as Java.

Figure 5.3: Data, meta-data and meta-models



In this type system, RDF resources are considered to be instances of RDF classes with properties that may be inherited and sub-classed. Semantic middleware and tools will be able to recognize the specialized vocabulary of terms defined in the RDFSchema language as well as the basic RDF resources, properties and property values introduced earlier.

An RDFSchema could then be defined to specify the semantic use of terms such as '*customer*', '*address*', '*order*' and '*item*' in the simple business scenario. Such a schema would precisely define the semantics of allowable resources in the meta-data for this scenario.

Semantic systems

Several additional layers above the RDF layer in the semantic stack have been defined by the W3C. These represent the additional language capabilities that are needed to support the development of 'ontologies' and the additional logic capabilities needed to reason with knowledge encapsulated in RDF meta-data. The significance of these layers is that they enable new classes of application and middleware that exploit useful vocabularies and rule-based reasoning.

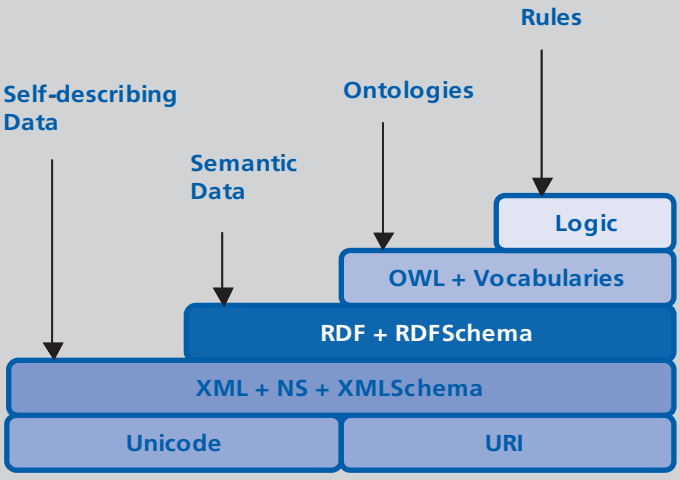
An ontology defines the terms used to describe and represent an area of knowledge. Ontologies may be used by people, applications and databases that participate in sharing of domain-specific information. The recently recommended W3C Ontology Web Language (OWL) extends RDF and RDFSchema — by adding more vocabulary for describing:

- **classes and properties**
- **relations between classes (for example, disjointness)**
- **cardinality (for example exactly-one)**
- **equality**
- **enumeration of classes.**

The definition of ontologies using OWL is currently an active field. Ontologies for commercial enterprises in many industries are emerging together with pilot projects to explore their use. The Tourism Harmonization Network (THN) is sponsored by the European Commission IST program and supported by a large number of enterprises in that industry is but one such example. THN explores the use of ontologies for reliable sharing of information between business partners in the tourism industry.

To maximize their value ontologies must be well-formed, easily accessible as sharable resources and comprehensive in their scope. Several long running projects — such as the Dublin Core Metadata Initiative (*www.dublincore.org*) and the Global WordNet Association (*www.globalwordnet.org*) — have set ambitious goals for defining the semantics of large numbers of linguistic concepts in common usage.

Figure 5.4: The semantics stack



Semantic middleware in future may be capable of combining a variety of readily accessible ontologies on demand to support business applications — see also the IBM Ontology Management System (*alphaworks.ibm.com/tech/snobase*).

The introduction of logic into the semantic stack has not yet been standardized by the W3C. However, there is a consensus of opinion that future semantic middleware will be able to translate RDF and OWL meta-data into simple logical assertions and, by introducing inferencing and other reasoning technologies, higher level business logic will be supported. This could be another example of the middleware 'push down' principle at work, resulting in further improved programmer productivity.

However, there is much to be done before the promise of semantic middleware can be fully realized:

- **knowledge from the many different commercial domains must be assembled into standard interoperable ontologies**
- **instance data must be annotated with semantic meta-data that refers to such ontologies**
- **middleware infrastructures must be adapted to recognize standard semantic constructs.**

This work has, however, already started. Examples of semantic business applications are emerging at a growing pace. In addition, many new scientific, geographic, industry-specific, legal, medical and governmental ontologies are already being developed (see NASA's Semantic Web for Earth & Environmental Terminology at *sweet.jpl.nasa.gov/ontology* for another example). Many Web pages and enterprise resources are already annotated with RDF and many more are emerging.

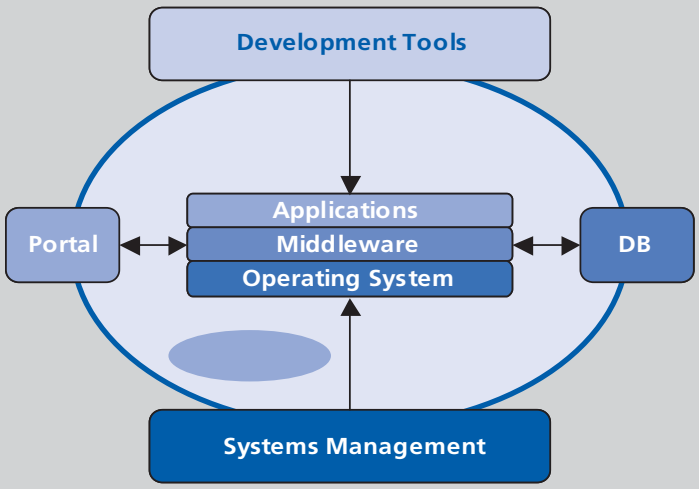
Furthermore, many prototype semantic middleware components are also emerging, including:

- **open-source RDF databases**
- **reasoning engines**
- **annotated UDDI registries**
- **Web Service components (see Semantically-Adapted Service Data Objects at *alphaworks.ibm.com/tech/ettk*).**

Future semantic systems (Figure 5.5) will be based upon an evolving set of ontologies and logic capabilities not only built into middleware infrastructure but also into development tools, database systems, management systems and enterprise portals. In the long term, the result could be much higher end to end:

- **coherency (from common semantics)**
- **reliability (from fewer mistaken meanings)**
- **productivity (from more intelligent tools).**

Figure 5.5: Semantic systems



Management conclusion

It would be easy to predict that at some time we will all be using some kind of 'thinking' middleware that automatically understands the meaning of data as it flows through supported business processes. However unlikely as that may seem, there is an undeniably growing interest in semantics following recent announcements by the W3C of standards for the Semantic Web.

Many of the middleware infrastructures deployed today already support the XML foundation needed for semantic metadata. Indeed several different kinds of metadata are also already supported (see Eclipse Modeling Framework at www.eclipse.org/emf/*, for example). To this extent it might be said that many of today's middleware tools and runtime products are already somewhat semantic in nature.*

The promise of future semantic middleware is that the same language will be usable by:

- ***enterprise business analysts***
- ***IT architects and modelers***
- ***application programmers***
- ***operations staff***
- ***business users***

— with agreement and coherency in the common meaning of their language.

As Dr Jones argues, before that promise can be realized for even the simplest of enterprise scenarios, there must be an aggregation of ontologies covering relevant knowledge domains that define the business processes to be deployed. Data must be tagged with appropriate semantic metadata.

The first step is a startup cost and may be seen by some as a barrier to entry. The second step will be manual effort for early adopters. It must then become an automated task as tools become more sophisticated.

Once the language has been developed and the data tagged, semantic middleware will provide the infrastructure needed to access ontological meanings, manage versioning and conflict resolution when meta-data are interpreted and provide reasoning in support of business applications. Realizing the promise of semantic middleware is very much a journey ... but major first steps have already been taken.

Designing for performance

Peter Bye and Andy Roles
Unisys Systems & Technology

Management introduction

Performance problems have been a concern since the origin of computer systems, and are not specifically related to the use of middleware. However, the use of middleware to build increasingly distributed systems, integrating existing application packages and new developments poses interesting problems.

The aim of middleware is to abstract the application developer from concerns about the underlying hardware and operating systems. While there are advantages in this kind of abstraction, in that developers can concentrate on business functions, there are also dangers. Complex layers of software sit underneath the application. The behavior of this software is often ill understood. The result may be a functionally sound architecture and implementation that is operationally unusable because of performance problems.

In this analysis, Peter Bye and Andy Roles argue — from their practical experiences — that performance must be considered at the start of any project, when the architecture is defined. Architects must have an understanding of the capabilities of the technologies and products proposed for the implementation, or recruit someone who does understand.

Once this is complete a performance model of the proposed system can then be built, based on the technical understanding of how the middleware should be used. If the middleware is new, or there is any uncertainty about its behavior, they suggest that early (and short) proof of technology projects are essential to explore the performance implications of possible solutions. The results can then be used to refine the performance model. Continuous measurement and tuning during a project's implementation are essential.

The problem

Too often, performance is not considered until late on in a project. Developers will spend months working towards a solution, only to find that it runs poorly in production. This is because:

- **they have overlooked the performance question completely (this still happens)**
- **the application has not been in a fit state to be able to test under load until the final few weeks of the project**
- **they have not sized the platform correctly for the expected (or unexpected) workload**
- **there is a belief that using more hardware will solve any performance problems**
- **they have been lulled into a false sense of security by believing the hype attached to middleware, frameworks or packaged solutions.**

We are seeing more and more examples of this last in the field. It is not that the middleware manufacturer has deliberately misled the client. It is often a case that the client has simply 'mis-used' the middleware and produced an ill performing sow's ear from a series of well performing silk purses. And, in some cases, the wrong middleware may have been chosen, which can lead to the introduction of an inappropriate technology for the problem at hand.

Performance modeling

A performance model is a useful thing to develop for any application. Much work has been done to try to automate the generation of performance models.

In their simplest form they work on the principle that, if one knows that one interaction with the system consumes X amount of resource (utilization) and takes Y seconds to complete (throughput) then it is possible to compute the resource utilization and throughput for any given arrival rate for interactions with the system.

Some of the most accurate performance models blend queuing theory, modeling packages and complex mathematics. Ironically, some of the best performance models use a ballpoint pen and the back of an envelope. The candid reality is that predictive modeling of system performance is an art, not a science.

One of the fundamental problems with predictive models is that people insist on modeling the performance of imbalanced systems. Computer systems behave much like mechanical systems:

- **work them within the boundaries of their design and they 'purr' like well oiled machines**
- **overload them and small imperfections can be magnified and cause severe problems.**

Many of the problems experienced by computer operating systems in the past have been ironed out by better software quality and considerably cheaper hardware. For example, the restriction on 'memory quantity' now seems to be a factor of the past — you can buy 1GB of memory for less than US$200. But most people will have seen systems experiencing 'memory thrashing' — the computer equivalent of the Tacoma Narrows bridge that so spectacularly disintegrated in 1940.

The point of all this is that the best advice is to tune an application before attempting to model it, or to use it as a starting point for any prediction. The counter to this is that, early on, most applications are far from stable, far from tuned and may not even be representative of what eventually goes into production.

Given these dichotomies it is not unreasonable to ask: why bother with performance at this stage? The intention of the rest of this analysis is to try to convince everyone of the need to examine system performance early on in the life of the development project and that the preceding question will be answered in the following sections.

The role of the 'proof of concept'

It is advisable to try to measure end to end performance and resource utilization early on in any development project, especially when new middleware technology is being deployed. When new technology is involved a proof of concept (PoC) development is always advisable, as the lessons learned from such an exercise will pay dividends in the long term. A PoC implementation enables developers to find out whether the architecture devised, and middleware chosen, have a good chance of working in practice.

In addition, a PoC has another, bigger, role to play. It can be used to estimate likely path-lengths and system resource utilization levels.

A PoC also permits development of correct instrumentation and debugging methods so that performance of the final application can be measured. All too often applications go into production, perform badly — and their developers are hamstrung because there is no way of finding out where the resources are being consumed. Ideally no application should go into production without instrumentation points being built into it at key locations.

A PoC that has been prepared well will help to determine where these instrumentation points should be, and ultimately to see where time is being spent before any major development is undertaken. Not only does the PoC enable developers to check out functionality and develop an end to end solution, it also provides the opportunity to develop a model of production performance based on expected throughput and the measurements obtained from the PoC.

Looking at the engine

Twenty years ago, one of the biggest criticisms of software development was the lack of re-use of components. If hardware development had worked in the same way as software development at that time, computer engineers would still be soldering basic NAND gates together to build computers and automobile manufacturers would still be winding copper wire around magnets instead of buying in ready made starter-motors.

To an extent this has been addressed over the last 10 years by the growth in Object Oriented development and concepts such as COM+ and more recently Web Services and Portals. Unfortunately this greater and greater abstraction has made it harder to deal with the performance of any system. An application which was once written in assembly language, and bound to a single CPU and a few disks, now has to contend with:

- **networks**
- **sophisticated database management systems**
- **generic parameter driven software**
- **layer upon layer of middleware.**

It is broadly fair to say that programmer productivity has improved through the use of re-usable objects (this may also be questioned — some measures have productivity going backwards). But the cost has been the introduction of less control over the performance of the whole application.

Consider the following (somewhat extreme) example. A programmer might want to write a program to query the balance of a bank account. He (or she) discovers that there is already an object written to do this, which accepts an account code as input and returns a real number as output. Delighted, s/he goes ahead and uses this object. But what s/he may not appreciate is that the object retrieves its information on the account by performing a brute-force search on an account balance table with 6 million entries.

This is obviously not that realistic. But it makes the point. Sometimes it is important to lift the hood on your new car to see if there is a decent engine underneath or just a large rubber band.

Typical components of performance

If one looks at a typical transaction processing system, its end to end response time can be seen to be the sum of a number of other response times. This is regardless of what middleware is being used, or whether the system is 'green screen' or a GUI based client/server system (Figure 6.1). End to end response times involve multiple components

Component times consist of:

- **network time — the time taken for a message to reach its destination in the system; this can apply to data being sent into a transaction program, or results being sent out to the display device**
- **CPU time — the time spent by the activity performing instructions**
- **queuing time — the time spent by the activity waiting for resources to become available**
- **I/O time — the time spent by the activity sending data to, or receiving data, from peripheral devices such as disk and tapes (typically, this is bound up in database activity)**
- **voluntary delay — the time spent by an activity doing nothing except waiting: this is different from queuing because the activity is not specifically waiting for a resource, it is just 'waiting' (as will be discussed in more detail below).**

Each interaction with a transaction or a service will have a response time. This will be made up of all, or at least the majority of, these component times — in no particular order.

The transaction will most likely comprise multiple functional services, which when grouped together form the whole transaction. Each one of these individual services may experience delays due to the component times listed above.

Even if one considers a batch process, one typically scheduled to process bulk data without human interaction, all of these times may need to be considered as components of response time for that process. This will also include network time if, for example, a customer information database that is being referenced by a batch program actually runs on a separate platform.

The goal of performance analysis is firstly to try to measure the time spent in each of these areas. Then it is to minimize each area to deliver the best end to end performance. But also note that if you cannot measure performance, then you will not be able improve it — because you will not be able to tell whether the efforts have had a positive or negative effect.

A key conclusion is that you should not be afraid to add instrumentation points into a system. If coded correctly their impact is likely to be minimal. Indeed, basic instrumentation should be available from any system all the time with the more detailed instrumentation being written so that it can be switched on and off dynamically.

Finally there is an adage amongst old performance analysts that says: 'If you cannot afford the overhead of instrumentation, then you really have got performance problems'. (We really do lead exciting lives, do we not?)

That said, it is often the case that tuning these component times can involve much effort for minimal benefit. That is why it is better to concentrate on tuning the areas that will deliver the most benefit to overall response time — the 80/20 rule where 80% of the problems are caused by 20% of the components. In most cases, practical experience shows the most often offending components are I/O time and queuing time.

Some words about voluntary delay time

Before looking at each of these two areas, consider the slightly enigmatic 'voluntary delay time' (VDT). This is time spent by an application just waiting for time to pass.

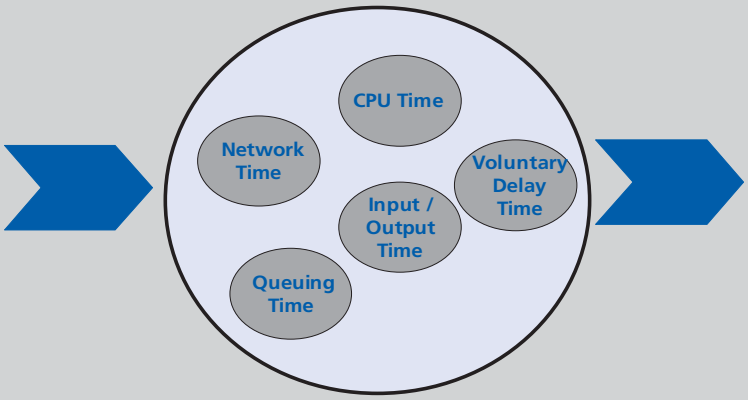
VDT is built into processes when programmers do not really have a way of knowing when a resource has become available. For example, a process may go into voluntary delay whilst waiting for a message to appear on a queue. The process will have been coded as shown in Figure 6.2. This is highly inefficient.

A far better approach would be for the waiting activity to go to into a deactivated or sleep state, to be woken up every time a new message is added to the queue. However these kinds of programming constructs are still being produced and if you do see VDT (if you are able to measure it) when monitoring your application, our recommendation is that this should be dealt with first.

System instrumentation vs. application instrumentation

Of course nothing is ever easy, and measuring the time spent in each area may not be possible simply because the instrumentation does not exist in the application. In these cases you may well have to use 'system level'

Figure 6.1: Typical transaction processing



instrumentation. This presents a view of application activity as seen by the operating system, for example:

- **the CPU time consumed by a process**
- **the number of I/Os performed by the process and the time taken to perform the I/O**
- **the time spent queuing on a resource (waiting for a record lock to clear or activities queuing at the CPU level).**

This shows what a process was doing — but not why it was doing it. Where possible this kind of instrumentation should be used alongside application-related instrumentation that details what an application does in terms that the developer will understand. It should also be noted that middleware often has its own instrumentation, frequently in the guise of 'log' files which, providing entries are time-stamped, can be jointly used for debugging, tracing and performance analysis.

Optimizing I/O performance — reducing database queuing time

It would be fair to say that most tuning activities achieve their biggest improvements by optimizing the I/O environment associated with any application:

- **in the mid 1980s, a typical disk I/O took between 20 and 30 milliseconds**
- **by the mid 1990s this was reduced to about 10ms**
- **presently a good cache disk subsystem should be able to deliver an I/O response time in the sub 5ms range.**

System attached caches — such as the Unisys XPC — can even accelerate I/O to the sub millisecond range. While the hardware moves on apace, it is worth examining the software to ensure that the application is making the most of the hardware.

Disk caching has hidden a multitude of the sins that plagued performance analysts over the past 20 years. File placement on disk packs has more or less become a lost art as issues of physical placement have become less of a concern. Physical I/O issues have been abstracted up to the software environment, and I/O optimization has been replaced with database optimization.

But databases are not perfect either. Common problems observed with databases include:

- **thread management**
- **caching**
- **record locking and queuing**
- **deadlock resolution**
- **selection of indexes**
- **batch processing.**

Thread management

Database management systems support multiple concurrent program activities accessing a shared resource (data). Each activity is typically allocated a database thread of activity which is:

- **instantiated via a BEGIN THREAD command**
- **terminated at the end of the program, or sooner if the program decides to do so via an END THREAD command.**

The database thread is used to maintain database integrity, and all database updates performed within the thread can be abandoned or rolled-back should the program err or encounter a problem. The thread is also used to allocate internal database resources and the typical DBMS will be sized to support a maximum number of database threads.

Several issues relate to thread management and applications should be designed to avoid these issues:

- **typically threads should not be kept open across client server links (for example while waiting for a user to choose from a list of items previously extracted from the database and displayed at the workstation); the thread can be instantiated and control passed back to the user and then the user will go for lunch, or his workstation goes down leaving the thread orphaned**
- **instantiating a thread can be expensive in terms of resource usage (allocating memory, assigning and opening files etc.); if every single interaction from a client program has to instantiate a thread then a huge bill arises in terms of resource usage; it is better to use a thread pooling technique where incoming client interactions select from a pool of already open threads related to a given service or, alternatively, underlying services should keep threads open indefinitely, performing a database COMMIT to secure updates and provide a known rollback point**
- **if insufficient threads are configured, the RDBMS will cause the accessing program to queue on the release of a thread's activity.**

Caching

Most database managers have their own store-through cache memory so that database pages read in and updated by one activity are maintained in some form of memory cache in order to speed up subsequent requests. Depending on how this memory is managed it might take longer to find a page than reading it from disk.

Record locking and queuing

Record locking and queuing matter. An incorrectly coded application can lock records for long periods of time. When a second transaction tries to update or even read the record, it finds it locked and then has to queue to gain access to the record.

An example of this has been seen at a site that for legal reasons has to record the details of who initiated every database enquiry (and update) as an audit record in a central log file. The log file started out as a single table but, since every transaction was hitting that file, it became a single queuing point with the result that transaction response times were horrendous. Once the log file was redesigned as 10 separate tables, selected via the last digit of the current time (0-9) expressed in milliseconds, queuing reduced drastically.

Nevertheless, the system was still not free of queuing after doing this. This indicated that further tuning was still required. A better solution would be to have one table per thread with selection based on a hash of the thread number (provided by the system).

Deadlock resolution

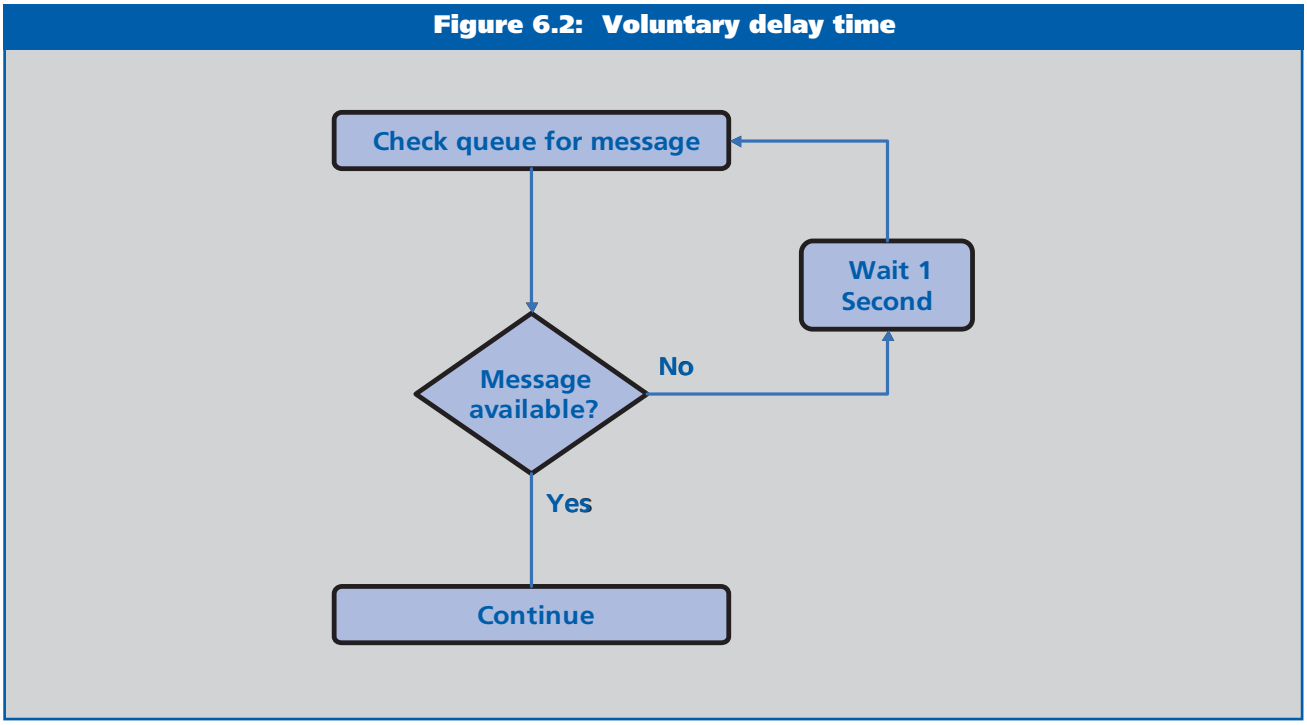
Deadlock resolution is worse than queuing on a lock; deadlocks result in transactions being terminated prematurely. A deadlock occurs when:

- **program A locks record X**
- **program B locks record Y**
- **program A tries to reference record Y but finds it locked by B**
- **program B tries to reference record X but finds it locked by A.**

This deadly embrace will be detected by the RDBMS, which will try to resolve the deadlock by killing off one of the offending activities. How the system decides on which program to kill is called the deadlock resolution strategy and is typically based on:

- **least wasted effort — the killing off of the program that has used the least system resources**
- **updates — the killing off of the program that has performed the fewest updates to the database**

Figure 6.2: Voluntary delay time



- **priority — the killing off of the program with the lowest priority.**

All of this results in wasted effort and wasted resources. From a performance perspective deadlocks should be avoided at all costs.

Selection of indexes

Much has been written about choosing the right indexes to ensure the best database I/O performance. The fact is that in many situations choice of index fields may be something that is again abstracted into the middleware layer. They may, therefore, be out of the developers' control.

However this an area to be aware of as, in the earlier example, selection of records via a non-indexed field will result in the RDBMS using brute-force or undertaking full-table scans. These impose unnecessarily high CPU overheads.

Batch processing

In today's online/real-time world, it is often forgotten that there may still be a requirement to perform a repeated operation on every single item within a database, or a requirement to add multiple records to a database without human interaction. Examples in the financial world are interest calculations or processing the payment of regular bills — both of which are typically done as batch processes.

In batch programs, it is good practice to reduce the number of commit points so that record inserts and updates are less frequently committed to the database than in a transaction environment. This results in a significant reduction in resource utilization. At one site overnight batch programs keep track of the number of records inserted and only commit the updated records to the database after every 500 updates.

Obviously how this is done depends upon multiple factors, which include:

- **the size of the records**
- **the scalability of the RDBMS**
- **the ability to restart the batch process should it fail just before writing its 499th record.**

Over componentization

We have already commented on the problems created in software development by the industry's tendency to keep on re-inventing the wheel. One phenomenon that we have observed at some sites is the tendency to subdivide applications into smaller and smaller functionally re-usable service components.

This is fine in a tightly linked application where each component is just a subroutine jump away. But defining each component as a separately callable service — that has to be scheduled and executed — places an enormous overhead on the whole system.

As an example of what can happen, one site we know of initially coded its entire replacement OLTP system as a series of independently callable services, each one handling a single low-level function (such as reading a single account record). After initial transaction performance tests revealed an incredibly long transaction response time and low throughput, the designers went back to the drawing board and redesigned the infrastructure to group these low level functions into common services.

In effect they increased the granularity of the items that had to be scheduled by the transaction monitor software. The result was a far shorter response time, with far lower system resource utilization levels.

Management conclusion

There are a number of conclusions Mr. Bye and Mr. Roles draw from this analysis. It may appear obvious, but the number of failed projects is so alarming that it should be clear that placing more emphasis on performance is necessary. While this has always been true to some extent, the use of middleware to build distributed systems adds new complexity. This is frequently not understood.

Their major recommendations are:

- ***performance must be considered at the start of a project, when laying out the architecture — it cannot be left to the end for that is too late; fixing an ill-performing system is much harder, and more costly, than getting it right in the first place***
- ***specifications for systems should contain performance requirements — which should sound blindingly obvious; but, as experience shows, performance requirements are often not stated and there is 'just an assumption' that the developers will 'make everything perform'***
- ***if there is any uncertainty about the behavior of middleware products, short PoC developments should be used to explore the parameters; the results can then be fed back into the model***

- ***architects must either understand the ramifications of the middleware technology and products they choose or work with people who have a detailed knowledge of the chosen middleware and other underlying technologies, such as operating systems and database managers***
- ***performance models should be constructed early on and maintained and iterated throughout a project***
- ***performance testing and tuning should be started as early as possible and continued throughout the rest of the project; key metrics need to be defined and monitored, in order to see the effects of tuning on them***
- ***performance problems may be solved by throwing hardware at them, but apart from increasing cost, hardware will not solve all problems; the increasing power of computers has been a mixed blessing for the software industry.***

This analysis has focused on performance. But it is just one aspect of non-functional requirements. Security, stability and manageability are equally important and should, likewise, be considered at the earliest possible stage, when the architecture is being defined.

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

USA and Canada:
Spectrum Reports, Inc.

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

UK and Rest of the World:
Spectrum Reports Limited

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

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

Email and Internet

Email:
spectrum@middlewarespectra.com

World Wide Web:
www.middlewarespectra.com

ISSN 1356-9570

[incorporating FINANCIAL MIDDLEWARESPECTRA ISSN 1460-7220]