MIDDLEWARESPECTRA

incorporating FINANCIAL MIDDLEWARESPECTRA

Contents

November 2007

Volume 21 Report 4

"Middleware is software that nobody wants to pay for"

Tom Welsh
Director
Tom Welsh Research Limited

Management Introduction

Chris Stone, co-founder and first president of the Object Management Group (OMG), used to remark that "Middleware is software that nobody wants to pay for". As Tom Welsh argues, far from being merely the complaint of a disappointed middleware enthusiast, this is a rather deep observation which has serious implications for the middleware market.

Stone's point was that making money out of middleware is an uphill struggle, because end users (including many IT executives) do not see it as a desirable asset in itself. In most organizations, middleware is not seen to boost the bottom line — or even the top line. Although it can save large sums of money, and may make the difference between the success or failure of large IT investments, this is obvious only to those who understand how distributed systems work.

Like bitter medicine that is palatable only when sugar-coated, middleware is most likely to be accepted when it is part of a complete application package. That is why systems integrators and major IT systems vendors have always been among the leading buyers of advanced middleware. Their purchases, while often substantial, are rarely publicized.

As a result, middleware is generally perceived as complicated, expensive, boring, or even irrelevant. Worst of all, its very existence may be overlooked.

Making middleware profitable: an elusive goal

In the past 40 years dozens of distinct types of middleware, and hundreds or thousands of middleware products, have emerged. Yet only a fraction of these have survived long — and only a handful have been serious commercial successes. Why is this?

One reason is that, for most people outside the computer industry, software is an impenetrable mystery. Even the media fail, more often than not, to give software its due. How often have we read, or heard on TV or radio, of a new 'computer', when what was really meant was a computer system, or a distributed system made up of numerous computers?

Worse still, the computer is widely seen as a 'black box' whose unpredictable actions are not attributable to any human agency. Even intelligent, well-read individuals do not always understand how all the information held by the Web can be crammed into a computer. This is because they do not know about software, still less about networks, and how these work.

As a wise cynic once observed, the perceived value of any piece of software tends to be inversely proportional to its distance from the end user's eyeball. (This is the principle behind the runaway successes of Windows, the Apple Macintosh and the Web). An obvious corollary is that software will be deemed unimportant, or completely ignored, if it lacks any obvious relevance to attaining desired business objectives. Every experienced salesperson knows that 'perception is all'. Nowhere does this rule apply more strongly than in IT procurement.

Another familiar sales maxim is that 'people buy holes, not drills'. In this sense, unfortunately, middleware is the quintessential drill. Qualified architects and experienced craftsmen understand its value — indeed, its necessity — but they comprise a small fraction of the overall market. Moreover, precisely because they are so well-informed (and because they must absorb their own costs), they are unlikely to pay premium prices for even the best of products.

Making an impact

On most occasions when middleware has been sold profitably, it has been because it yielded unmistakable end user benefits. For example Teknekron, the forerunner of TIBCO — one of the great specialist middleware companies — found that its Information Bus sold very well in the world's financial markets because of its ability to 'get rid of' many of the terminals that cluttered dealers' desks, replacing them with just one. It also used techniques — such as failover — to provide reliable information feeds. So useful were these improvements to the dealers that corporations were happy to pay for the new technology.

Similarly, the Web was the first form of middleware that allowed a single GUI to be used for accessing local and remote computers, whether on a LAN or around the world. Web Services, for their part, automate the labor-intensive activity of Web browsing — saving much time and effort. Today, SOA and BPM — the most recently fashionable types of middleware — owe much of their popularity to the alluring prospect of automating business processes.

If, then, middleware is so hard to market profitably, why do so many companies go on trying to do so? There are several possible reasons. Middleware enjoys high prestige in the technical community; many software vendors are keen to position themselves as 'technology leaders'. More pragmatically, middleware is an invaluable 'fifth column' which when cunningly marketed can help a vendor to infiltrate its customers' IT architectures, establish lock-in and shut competing suppliers out. Big players such as HP, IBM, Microsoft, Oracle and Sun are particularly apt to pursue this kind of middleware strategy.

A copybook success story: Microsoft

A study of the software market — and middleware is no exception here — shows that many companies fail because they take their eyes off the ball. The ultimate goal of any company is to make a profit, the bigger the better. In the long term, a company should aim to maximize its overall profit, and this must entail expanding its market share as much as possible.

Yet many software suppliers allow themselves to become side-tracked by:

- **creating the biggest, best and purest architectures**
- **embarking on eye-catching but expensive projects**
- **outdoing their most obvious competitors.**

All too often they run into delivery and/or cash flow problems, back the wrong horse or become over-ambitious and fail to deliver. Yet one company has done an outstanding job of keeping its eye on the ball for over 30 years. It is, of course, Microsoft.

Rarely accused of excessive technical perfectionism, it has

assiduously entered new markets, increased its market share and, wherever possible, expanded its markets and market reach. While carefully recruiting the best employees and developing hundreds of software products, it has always put financial stability and growth first. With the help of a certain amount of luck — which, we should remember, favors the well-prepared — Microsoft has enjoyed unprecedented growth.

Ten years ago it overtook IBM to become the world's biggest software vendor. Now it has no serious rivals in the pure software business.

Microsoft has evolved a marketing strategy that gives it many advantages. First of all, its product line encompasses almost everything needed by consumers or businesses. It offers:

- **an operating system (Windows)**
- **an office automation suite (Office)**
- **a Web server (IIS)**
- **a Web browser (IE)**
- **a database (SQL Server)**
- **a comprehensive software development suite (Visual Studio).**

The .NET Framework, Microsoft's answer to Java EE and Java SE, is either shipped with Windows or comes as an easy add-on option. But whereas Java EE constrains developers to code in Java, while allowing them to deploy on almost any operating system, the .NET Framework constrains them to deploy on Windows, while giving them a choice of programming languages.

Even more important than Microsoft's 'full spectrum' software marketing strategy is the sheer size of its customer base. There will be 1 billion PCs in use worldwide by the end of 2008, according to Forrester Research, and 2 billion by 2015. Of these, some 95% or more run Windows.

As of 2006, IDC reported that Windows had almost half of the global server market as well. With hundreds of millions of customers, any product that achieves reasonable penetration of that base is more or less guaranteed to sell tens of millions of copies, bringing in revenue on the order of $1 billion.

For these reasons, Microsoft has far more freedom in its marketing strategy than the average software vendor. If it wishes, it can introduce a new product completely free of charge, simply by 'bundling' it into Windows — as was notoriously done with IE. Microsoft has consistently pursued such a strategy, no doubt reasoning that it stands to gain most in the long run by making Windows as attractive and cost-effective a platform as possible (arguably the price of Windows Vista today is little more than Windows 3.1 some 15 years ago, yet Vista is vastly more capable). This has entailed 'giving away' some middleware such as COM and COM+, and offering others at low prices.

Today almost all of Microsoft's middleware — in the shape of Windows Communication Foundation (WCF) — has been folded into the .NET Framework which comes free with Windows. Instead of making money directly through middleware sales, Microsoft uses middleware as a lever to increase its share of the mobile, desktop and server markets — the best way to maximize its profits in the long term.

By the end of the 1990s, Microsoft had internalized the lesson of the Web: that the whole world would henceforth be linked by a single 'network of networks', in which everyone could participate freely. That was when it gave up its efforts to overcome the Internet, the Web, Java, CORBA, and other industry standards — and decided "if you can't beat 'em, join 'em".

The new approach, roughly speaking, was to support the most popular standards but to aim at influencing them from inside rather than defeating them from outside. One way in which this played out was calculated to astonish seasoned Microsoft watchers: the decision to work closely with IBM on a whole new framework of specifications governing Web Services (and, in due course, SOA).

Standards: cannot live with them — cannot live without them

Back in the 1950s, 1960s and even 1970s vendors — mostly IBM — made their own standards. Among the best middleware examples are CICS (1968) and MQSeries (1993), both of which are still going strong. System Object Model (SOM) sank like a stone, though, as did OpenDoc and Taligent (both collaborative efforts with other companies).

Microsoft made COM ubiquitous by bundling it with Windows, but was not above imitating successful IBM products like MQSeries (MSMQ) and CICS (MTS/COM+). Its latest effort is .NET, which exhibits a considerable degree of originality.

Other classic vendor-created de-facto middleware standards include AT&T's Tuxedo and Sun's Java. The latter, however, is now managed by the Java Community Process (JCP) rather than Sun.

From a software vendor's point of view, the desirability of vendor-neutral industry standards is a complex question. Users gain from them, if only because it becomes easier to abandon a poor product and migrate to a similar one that complies with the same standard.

Rather less obviously, standards also favor most vendors — with one exception, those of the established leader (whose products, having been designed years ago, are almost always incompatible with the standard). Last but not least, standards help everyone to work more quickly and efficiently, and it is risky to oppose them too obviously.

The problem, then, becomes one of controlling standards — rather than being controlled by them.

Language and database standards, published from the late 1950s until the present day, were instrumental in the rapid expansion of those markets. Networking standards allowed all types of computer to interoperate, which initially annoyed and frustrated vendors but was later seen to be broadly advantageous. The 1980s saw the first important vendor-neutral middleware standards — including NFS, DCE, CORBA and XA. The Web (1990) was arguably the biggest breakthrough of all — beating even TCP/IP, which it required.

By the early 1990s, just as Stone was coming to the conclusion that "middleware is software that nobody wants to pay for", free (or at least very inexpensive) middleware was already on the way to taking over the world. From the point of view of anyone looking to make big profits by selling network software, TCP/IP was the kiss of death. As early as 1982, the US Department of Defense mandated the new protocol suite as the standard for all military computer networking. Driven by energetic UNIX start-ups like Sun Microsystems, TCP/IP grew more and more popular throughout the 1980s, and swept to world domination with the popularization of the Internet in the early 1990s (both DECnet and SNA were among the more obvious victims).

Once every Windows PC had a TCP/IP stack — as well as those built-in to UNIX and other workstations and servers — neither proprietary networking systems nor the OSI formal standard had a chance. Why pay for a minority networking system, with an uncertain future and expensive support, when you could join everyone else in what was essentially free networking?

TCP/IP's advantage was further accentuated by the advent of the Web. Its transport protocol — HTTP — runs on top of TCP/IP.

By now vendors hardly knew which way to turn, or which way was up. How could they best recapture the revenue streams that had once flowed so freely back in the days when each customer was locked in to buying those products written specifically for his or her computer's hardware architecture?

On the one hand, it looked as if TCP/IP and its host of transport protocols had squeezed all the profitability out of the networking business. Instead of vendors having the upper hand in setting standards, that function had been usurped by self-appointed groups of technical experts — such as the IETF and W3C. Microsoft, for one, spent a great deal of time and money trying to persuade consumers to forsake the Web and use its own closed, proprietary network instead. Less flexible companies like AOL and Compuserve persisted in that futile attempt for many years, with ever-dwindling customer bases.

The new dispensation: IBM and Microsoft take over

When looking back on the 1990s, one can see it as an era characterized by competing middleware standards and would-be standards. On the whole, neither proprietary de-facto standards — such as CICS, MQSeries or COM — nor vendor-neutral standards — like CORBA and the Web — could gain a decisive advantage. By the end of the decade, the two big winners looked like being the Web and Java: this was not good news for Microsoft.

Realizing that it was outgunned, Microsoft 'upset the board' — and began a new game with completely different rules (which is almost always a reliable strategy for the biggest player in any marketplace). In future, the word came down from Redmond, distributed objects like CORBA and Java were out, and XML Web Services were in. This majestic volte-face was meticulously timed to coincide with the launch of .NET in summer 2000.

To everyone's astonishment, IBM joined Microsoft in advocating and steering Web Services. Suddenly it was Sun's turn to become the outsider (and, in its behavior, it duly obliged).

What was the thinking behind this new development, whereby the world's two biggest software vendors began assiduously working through standards consortia? IBM's Bob Sutor, speaking at the first Interoperability Summit in December 2001, summed up IBM's view of the matter. As reported by Alan Kotok[1], Sutor put his cards on the table.

"To achieve this vision [enterprise Web Services] will require

an unprecedented degree of co-operation and co-ordination among standards groups, and it will also require many groups to change some of their long-standing operations." IBM's experience with developing the early Web Services specifications showed that it could shortcut the usually lengthy standards processes without sacrificing technical quality. Sutor said he still expected to involve the traditional standards groups in the development process, but IBM would not be tied to the extended timetables that many groups have used in the past.

Sutor also said IBM still expected high quality results — 'weeding out the junk' — but he wished to see standards groups working with Open Source communities as well as co-ordinating their activities to obtain architectural consistency. In addition he expected the work of standards groups to implement reality checks — by having as many of the technical decisions as possible made by industry representatives.

IBM and Microsoft submitted SOAP 1.1, the fundamental Web Services specification, to W3C in 2000. WSDL followed in 2001.

But then the two vendors abruptly changed course and handed UDDI to OASIS instead, where the groups working on ebXML and UBL began to show interest in Web Services. In 2002 W3C set up a Web Services Activity with several subsidiary Working Groups. Nevertheless, IBM and Microsoft continued to favor OASIS — handing over to it WS-Security, WS-Transactions, WS-Coordination and BPEL4WS.

By mid-2004 there were already over 100 specifications and standards closely related to Web Services, and still the flow continued. In a 2004 paper entitled "Standards for Service-Oriented Architecture", Scott Dietzen, then CTO of BEA, listed some of the specifications, protocols and programming models that he considered important for SOA at that time. Starting with 'the WS-* family', he moved on to:

- **the Java APIs for Web Services**
- **W3C's XML family of specifications**
- **'metadata and JSR-175'**
- **BPEL, BPELJ and JPD**
- **xQuery**
- **Java Web Services and JSR-181**
- **EJBGen and EJB 3.0**
- **XMLBeans**
- **XScript**
- **page flow for Java and Struts**
- **portlets**
- **WSRP and content management**
- **server-side 'Controls' (JPF, JPD, BPELJ).**

This lengthy shopping list was — and is — just the sort of 'approach' that IBM and Microsoft (as well as other big vendors) love to see their customers asking to be delivered. It favors the 'rich'.

As early as 2002 confusion was growing, as it became obvious that Web Services were failing in their primary goal of guaranteeing simple interoperability between different platforms. Consequently IBM, Microsoft and seven other companies set up the Web Services Interoperability Organization (WS-I). Explicitly rejecting the role of a standards body, WS-I aims to provide profiles, sample implementations and guidelines. Rather than issuing standards, in fact, it is in the business of telling implementers and users how to make existing standards work properly together.

Today, W3C continues to refine its ideas and specifications for Web Services and SOA. Meanwhile, OASIS has:

- **no fewer than 16 committees focused on Web Services**
- **another ten working on SOA**
- **several dedicated to relevant security specifications.**

In August it (OASIS) announced the formation of yet another six committees to 'simplify SOA' by advancing what is now known as the Service Component Architecture (SCA). Interestingly enough, although SCA is being supported by IBM and over a dozen other major vendors, Microsoft is not involved. But this in no way diminishes its commitment to Web Services and SOA.

Meanwhile, the more vendor-independent consortia — such as IETF, W3C and OMG — continue their valuable work maintaining and improving the Internet, the Web and other forms of middleware. But Web Services and SOA — definitely the middleware of the 2000s so far — remain firmly in the control of IBM and Microsoft, even though all the specifications are passed through OASIS or W3C for approval. Perhaps this is the inevitable compromise between the vendor-imposed standards of old and the unduly controversial vendor-neutral standards of the 1990s.

Management conclusion

Trial and error is the software industry's modus operandi. The minds of the software industry seem incapable of taking in the full scope of the problems facing business (and

individuals), let alone coming up with ideal solutions first time.

Instead, as Mr. Welsh describes, the scene is one of hundreds of independent or partially-collaborating groups, each tackling some particularly challenging problem — 'scratching an itch', as it has been said. As time goes by, thousands of techniques and pieces of code pile up, providing a marvellous source of ideas for future pioneers. Thus it was that Tim Berners-Lee devised the Web, by gluing together the separate concepts of hypertext (55 years old at the time), markup language (30 years old), and the Internet (20 years old).

IT is now entering a period when a great deal has been learned about middleware. Many attempts have been made to design the 'ultimate middleware', leading to the unsurprising discovery that there is no such thing. You can have more performance, but only at the cost of less flexibility (and probably higher price). Tight security can be imposed, but the consequences include reduced performance and ease of use, and much higher prices.

Despite what some extremists have been saying, distributed objects are suitable for close-knit systems that frequently exchange small packets of data — especially when designed by a single agency. On the other hand, XML Web Services excel in open networks where many independently-designed systems must work together as best they can. And so on.

More than ever, middleware is the 'software that nobody wants to pay for'. But that is mostly because it is everywhere, and we have come to take it for granted.

1 http://www.xml.com/pub/a/2001/12/12/kotok.htm

Agility and events

Dr Keith Jones
IBM Software Solutions Worldwide

Management introduction

In the current business environment, where companies are under increasing pressure not only to increase revenues but also to respond more quickly to changing market conditions, business enterprises will be successful only if they transform themselves and become more agile. Increasingly there is a focus on integrating business and IT strategies so that both can respond more readily to opportunities and threats as these are recognized. Componentization of both business and IT domains, a converging interest in modeling before execution and service orientation, are just some of the ways in ,which this integration is being approached. But is this enough?

Many companies have now identified key business services and are beginning to realize the value of building a Service Oriented Architecture (SOA) that enables services to be deployed quickly and enables flexibility within business processes for creative change. But some have already gone further in their quest for agility.

In this analysis, Keith Jones reviews the ways in which business events are being added to business services to provide a more dynamic form of agility, one that enables organizations to:

- *change their business processes on-the-fly*
- *capitalize on market opportunities*
- *counter threats that might otherwise go undetected.*

Agility

Agility has become the center of focus for many in the IT industry. This focus is in direct response to increasing business demand for IT responsiveness plus the realization that many enterprise Information Systems have become extremely complex, difficult to extend and expensive to change. From the parochial IT point of view, becoming agile is fast becoming a matter of survival.

But becoming agile is not just about developing software more efficiently, using extreme programming methodologies and other concerns inside the IT-box. It is more fundamentally about the flexibility that enables business leaders to change the direction for an enterprise when opportunities arise or competitive threats are perceived.

Business strategy for many enterprises is increasingly dependent upon agility in the marketplace. In the face of competition from around the world many corporations are basing their strategies upon an ability to respond quickly to new growth opportunities as well as becoming more efficient at bringing products and services to market.

The pace of this change is quickening for most. Thus pressure is mounting upon IT departments to accelerate their response to demands for innovation and change. Yet technology has continued to evolve at a rate that many believe has been too fast for most enterprises to exploit.

The question for IT strategists is, therefore, how best to exploit available technology to achieve the agility required within the bounds of acceptable risk. In this climate of adjustment, SOA has become the de facto prime answer to this requirement in recent years. Many organizations are now benefiting from their first generation of SOA deployment projects. Some boast advantages in both time-to-market and cost-reduction for new business capability — delivered as services. Business services have become the focus for those most convinced of the power of SOA.

The basis for much of the agility achieved by adopting SOA is the identification and realization of business services that can be re-used in a number of different contexts. Any given business service might be used in several different business processes. The relative cost — in both time and effort — is reduced with each re-use.

At the same time there is an overall reduction in complexity associated with building systems using loosely coupled IT components that implement well encapsulated logic and information service providers (Figure 2.1). A service bus (also known and as an ESB or Enterprise Service Bus), with registry and standard service interfaces, provides much of the technology for loose coupling between service components.

The tools needed to provide the agility afforded by SOA must facilitate modeling based upon business semantics that are later incorporated into business logic (Figure 2.1) — as it is realized in a form that can be deployed into the supporting middleware infrastructure. The semantics should also provide the basis for meta-data. This is vital because this is used to implement mediation logic deployed in the service bus to automate transformation, routing and management of service interactions.

Much of the functional capability required to achieve business goals can, therefore, be implemented using the SOA approach outlined. Once business process logic has been modeled, assembled, deployed and managed according to well defined policies, a certain level of agility in both the IT and business domains will likely be achieved. Nevertheless, this takes significant planning, executive resolve and professional skills to realize in typical enterprise scenarios.

Reaching the next level

Realizing a component-based IT strategy using SOA is a positive first step in achieving agility in an organization. Yet many strategists are recognizing that the next level of agility can only arise from making changes dynamically to business systems (and their IT implementations) in response to changing market conditions, efficiency goals and threats to competitive advantage.

A number of different approaches might be taken in order to reach this next level. For example, exposing business rules so that they can be modified as business processes are operating is an approach being used by enterprises in the retail and travel industries. But exposing business rules and other configuration details is not yet sufficiently evolved to attain an automated approach to dynamic change/control.

The missing components are business events. These are the mechanism for: detecting that significant state changes have happened on-the-fly within a business system and determining what action should be taken and when.

Once a course of action has been decided, rules — and process meta-data — can be adjusted dynamically to optimize business process operations and to defend against threat situations.

Business events may be added to an SOA by creating a layer of event producer components closely related to existing service components (Figure 2.2) together with a layer of

event consumer/processor components which are able dynamically to adjust business process rules and control parameters. This effectively introduces a feedback loop which is able to deliver the agility of business processes and that automatically can adjust those to what is needed for near-optimal business efficiency.

[It also worth noting that this feedback loop may be similar in effect to, but more dynamic than, the 'monitor-model-assemble-deploy' refinement process that may be applied to services used to implement business processes in an SOA. In this process, business activity monitoring (BAM) provides feedback that may be used to fine-tune business rules and process meta-data on a timeline that is faster than original service development but slower than dynamic business event processing.]

Adding business events

Business events may be added to many different kinds of enterprise systems, both SOA and non-SOA. An initial, rather obvious, requirement is that all state changes occurring within business-critical processes are fully understood and KPIs (Key Performance Indicators) are fully documented.

Once this has been achieved, some form of sensitivity analysis must be performed to understand the relationships between the business rules, state transitions and the relevant KPIs. Only when this knowledge has been captured can business events be introduced for full effect.

Sensors must then be introduced into the business logic, or the middleware infrastructure that supports it, to capture the state changes as business events occur (Figure 2.3). As events are produced, some amount of contextual information must be captured as well as the critical information about the state change that has just occurred. For example, if an end user has just ordered a product on a Web page, it is desirable that a business event be produced for later processing.

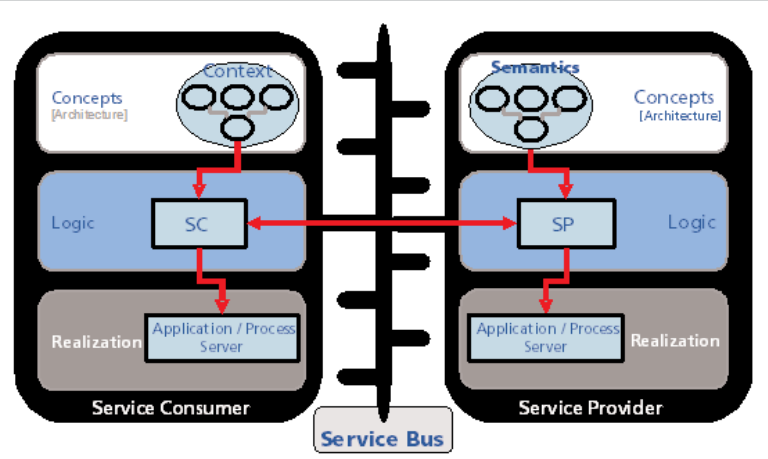
Business events may be produced for specific consumers or, more generally, for any consumers that have registered an interest in specific state changes. This level of decoupling may be implemented using an event bus together with an event registry containing meta-data about events, producers and consumers.

Events may be created/emitted in streams according to:

- **their source context**
- **the state changes captured**
- **a specific window of time**
- **or some combination of such factors.**

The event bus is the middleware component that logically composes these streams, by collecting events from one or

Figure 2.1: Essential service components



more producers — once consumers have registered their requirements. An event store is often included in the configuration when consumers require delayed access to events or sophisticated correlation between streams.

Event consumers are the 'end points' in an event processing network where business events are filtered, transformed, enriched, aggregated and logged into persistent storage. Each consumer must register for specific events or specific streams of events so that the event bus can effectively route key business information to business process controller/optimizer logic. The consumer logic is based upon the business semantics needed to recognize the significance of certain individual events, combinations of events or specific sequences of events (Figure 2.3).

The sophistication of certain event consumers may be surprising to some. Whilst it is possible to create simple consumers that count or detect single event types, business agility demands complex correlation of (possibly) multiple event streams to detect:

- trends
- violations of business rules
- the absence of critical events
- 'situations' that require immediate corrective action.

Such situations might include fraudulent manipulation of valuable resources, such as financial instruments.

Event processing may also be extremely resource intensive. Each event produced might be laden with large amounts of data. Seemingly, best practice is to create events carrying the smallest volume of data needed to satisfy requirements. This, however, sometimes leads to unwanted constraints on potential consumers that have yet to be identified.

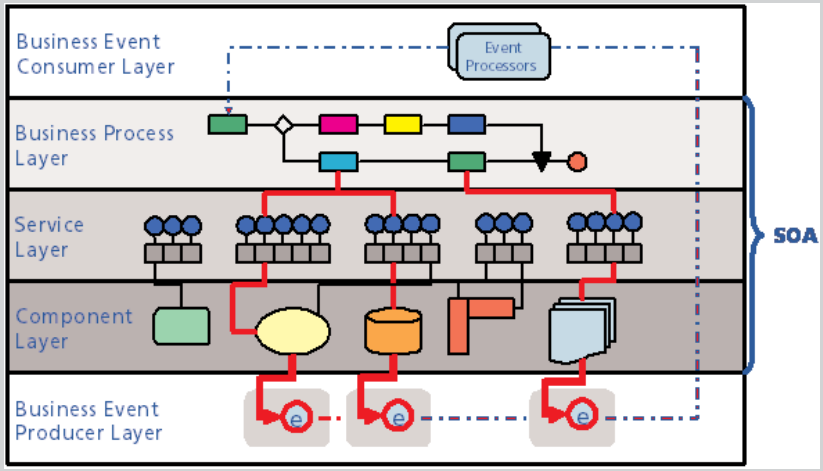
The number of different events produced can also become a burden on infrastructure resources. Best practice is always to produce the minimum number of events required to provide the agility needed.

To realize, therefore, the best possible re-use of event types over time in an event processing network it is important to discriminate closely between candidate state changes and resist the temptation to create composite events at source. If composite or aggregated events are needed, they should be created down-stream by event mediators in the bus — or by event consumers at end points in the network.

A third consideration is the amount of computing power required to process each event. In most real-time applications and graphical user interfaces the rule of thumb is that event processing must be kept to an absolute minimum.

However, in sophisticated business process optimization applications, the amount of computing power required per

Figure 2.2: Business event processing layers



event can become significant. Some of the power required is needed to filter and route events according to complex sets of rules.

Tools-generated code or rules engine technology are often the best way to apply the necessary logic at an affordable average cost per event for this function. Events in filtered streams must then be matched and behavioral patterns detected. This is often best achieved using algorithms implemented in available highly optimized pattern matching engines.

Finally, once patterns have been detected and situations recognized, actions must be taken in the processing of events. This could be creation of additional composite or complex events for further down-stream processing or manipulation of process metadata in order to affect change in the way a business process is operating. The total amount of computing power required per event may indeed be quite considerable in these applications.

Services, events or both?

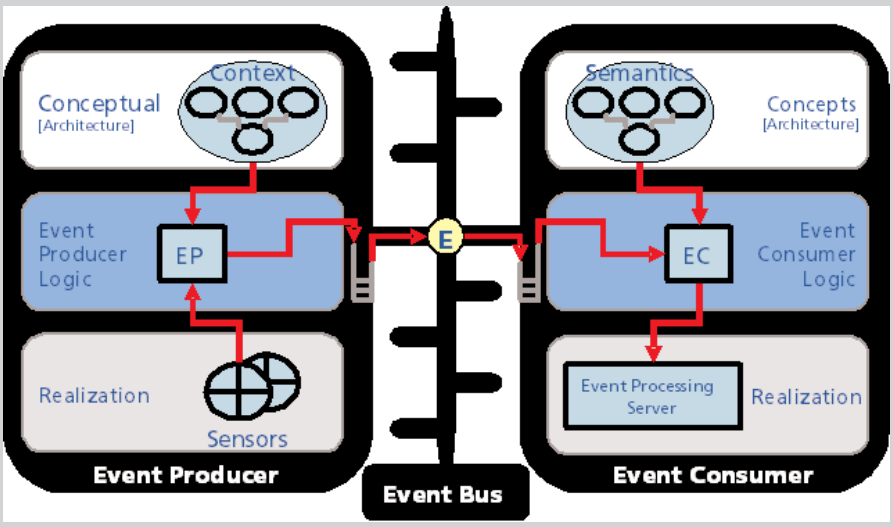
Since enterprise Information Systems must properly represent the real world interactions that characterize everyday business operations, it would be reasonable to expect interactions between IT components to mirror the real world as closely as possible. But the interaction models supported by services and events are completely different and so it is natural to ask which is most likely to produce the agility required.

Service consumers generally interact with service providers synchronously using a request/response protocol that is similar to RPC. Although this is true of most SOA deployments, the service interaction model actually also supports asynchronous one-way notification protocols as well as simple request/response. This enables service components to interact in a number of different ways (even though these are not often exploited in SOA implementations).

In contrast, event producers and consumers generally interact asynchronously using one-way notification protocols that are similar to publish/subscribe (pub/sub). Although this is true of most event deployments, the event interaction model also supports request/response 'pull' protocols. There is considerable potential overlap here between the service and event interaction models — as implemented by the many available vendor middleware solutions.

Service requests (and responses) are most often implemented as messages with an XML-based representation of data flowing over a service bus — even though this is not the only possible implementation in an SOA. Similarly, events are also often implemented as messages with an XML-based representation of data flowing over an event bus although, once again, this is not the only possible implementation.

Figure 2.3: Essential event components



This close match between service and event implementation strategies leads to the obvious notion that some events may trigger service invocations and that some service components may well act as event producers. Figure 2.4 shows this with a convergence perspective.

The advantage of this convergence is compounded if tools supporting such a middleware infrastructure also support a common set of business semantics for modeling, realization, deployment and management of both services and events. In addition, such tools might also support the development of mediator logic (not shown in the bus in 2.4) that transform, enrich, aggregate and route service and event messages in an integrated and consistent manner.

Management conclusion

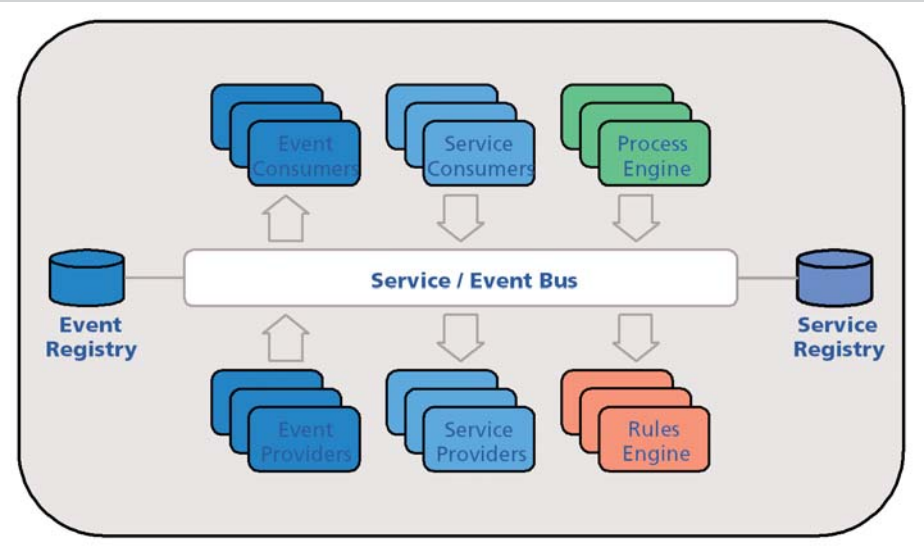
Agility is really not just about how quickly new business logic can be developed using extreme programming or similar methodologies. It is about how quickly a business can change its strategy to take advantage of new opportunities — or react to competitive threats — in today's increasingly dynamic global marketplace.

How best to develop business agility has become a prime focus for many IT departments as they develop and maintain complex operational information systems in support of critical business processes. For many, the advantages of SOA have already satisfied at least some of their agility requirements. Yet, for most, SOA is still a promise that has yet to be fulfilled. SOA, once implemented, can provide dramatically reduced time-to-market and reductions in IT costs as new business services are composed, deployed and then re-used in creative ways. But, as currently conceived, SOA deployment may not achieve the ultimate in agility — namely dynamic reconfiguration of and optimization of business processes as they are in-flight conducting everyday business transactions. Some organizations are already taking this next step in their quest for business agility.

By adding business events to their business services such corporations are able dynamically to process streams of events that carry information relating to in-flight business activities, key performance indicators and suspicious activities that may be a threat to business integrity and competitive advantage.

From an IT perspective, business services may be implemented using SOA methodologies and technologies but a different approach is currently taken for implementing business events. This may simply represent a temporary difference in focus amongst industry thought leaders and technologies available from middleware vendors. Nevertheless, there appears to be sufficient common ground to support a convergence of methodologies and technologies at the conceptual/semantic level, as well as at the logical component and implementation levels, so that agility becomes more accessible to more enterprises over time.

Figure 2.4: Converged service/event middleware infrastructure



Beyond ACID: an adaptive approach to transaction management

David McGoveran
President
Alternative Technologies

Management introduction

Fewer and fewer IT people find phrases like 'transaction management' meaningful. Transaction management software — which has responsibility for transaction integrity, scheduling, logging and recovery — is, today, more or less denigrated in many IT shops. Increasingly, transaction management issues are left to application developers.

When an interaction is perceived as 'transactional', design and control often ignores other applications. Transactions may be created merely for recoverability. Fewer and fewer developers understand transaction principles like isolation or integrity, or how a transaction manager might streamline development and improve efficiency and robustness.

In this analysis, David McGoveran examines a new approach to transaction management that addresses these challenges and how they have evolved. Called 'Adaptive Transaction Management', this patented technology involves re-thinking transaction motivations and fundamentals. In the analysis he describes a portion of the new model and contrasts it with traditional thinking. (Due to space limitations, this description cannot be a complete statement of all the models or principles; but there is sufficient to convey the concept and how this addresses key problems.)

Driving forces

Two driving forces lie behind the shift away from interest in transactionality. They can be identified as:

- programmatic transactional control
- an apparent divergence of business requirements from traditional transactional models.

Programmatic transactional control is where transaction processing was intended to act as an enabler complex for distributed transaction processing applications and their integration. The democratic chaos of programmatic transaction control has, however, had unintended consequences. Essentially it has disparaged uniform, centralized control while putting tremendous power (for example, programmatic choices) in the hands of diverse untrained and often unsuitably authorized and/or undisciplined developers who have not known how to incorporate disparate transaction models. Too often the result has been to promote the notion that transaction management is either easy or else offers little benefit.

The second driver is the apparent divergence of business requirements from traditional transaction models. Today's business transactions often violate some of the assumptions that characterize traditional transactions. (These include using a few relatively simple statements, access or modification of a few data records, few resources affected, short execution times and single authority control. If observed these meant that a single resource manager [see below] was feasible.)

One of the difficulties is that distributed transactions are no longer the province of a single business entity that can mandate a workable distributed transaction processing strategy. Indeed, nested transactions are more common. Distributed transaction components are both geographically dispersed and separated by significant, perhaps asynchronous, communication delays so that any concept of two phase commit is unrealistic. Compensation, though popular, frequently devolves into a difficult set of custom procedures that make applications brittle.

In essence, business transactions have changed. Business to business transaction requirements involve multiple resource owners and authorities. Business processes, negotiations, contingencies, searches and optimizations (for example, finding the best price) involve long running transactions.

Furthermore, today's transactions require data that no longer resides in one species of managed data store, but might be accessible only by Web search. Similarly, the assumption of data consistency and compatibility in a read-only environment — perhaps reasonable when data was uniformly controlled — cannot be relied upon to be valid. Business activity monitoring combines multiple, integrated data sources with complex analysis and decision processing It usually accepts potential consistency and repeatability resolution as the price of a timely, online view of a business activity.

Competitive pressures are forcing timely responses to complex opportunities. Applications are incorporating transactions where every millisecond is critical, internal complexity exists and access is required to massive databases. Some form of change is needed. But to understand how Adaptive Transaction Management is applicable it is necessary to recall the 'traditional transaction'.

Traditional transactions

Traditionally, a transaction is a set of actions that access or change a set of resources. The initial state of a resource set, possibly characterized by transaction parameters, acted on by a transaction is defined as being consistent, and so either implicitly or explicitly defines and satisfies a set of consistency conditions (also known as constraints or integrity rules).

Once defined, the transaction creates a delimitible set of changes from the initial state (including any parameter values). Each change to the resource set (short of the final change) creates an intermediate state, often intended to be inaccessible to other transactions. Eventually, the transaction leaves the resource set in a final state. Barring errors, this intended final state is deemed consistent.

Thus a transaction transforms a resource set from an initial consistent state to a final (and usually different) consistent state. A transaction model formalizes these ideas, ensuring that transaction behavior under various conditions is predictable.

Transaction processing is subject to multiple difficulties. A transaction may use resources inefficiently or may fail to complete operations as designed. Errors may cause the final state to be inconsistent. Transactions may execute too slowly. In simple environments, such difficulties can be handled manually. Automated or semi-automated approaches (for example using a transaction manager) are required in more sophisticated situations.

Transactions often execute under control of a transaction manager that enforces the transaction model. A traditional transaction manager ensures that the initial and final states are consistent and that no harmful side effects occur in the

event that concurrent transactions share resources. It typically enforces the isolation of a specific transaction using a default concurrency control mechanism (for instance, pessimistic or optimistic). If an error occurs before the final state is reached, the transaction manager handles recovery or returns the system to its initial state.

This sort of transaction processing still underlies the most automated of financial and commercial transactions.

An unspoken assumption in automated transaction processing has been that errors are exceptional and the transaction processing cost is minimal. Programming effort, both design and coding, implements transactions assuming that it is permissible to simply start over and re-do the work if anything goes wrong.

This is no longer valid for the majority of business transactions, let alone reflecting the entirety of any business' real world experience. Most business efforts and expertise are about avoiding exceptions, mistakes and imperfections or correcting their effects. Unfortunately, application programmers often treat transaction error recovery as an afterthought, or final 'check-box'. A realistic systemic approach is highly desirable .

Naïve integration of automated transaction processing systems — via, for example, transactional messaging — maintain that the resulting complex application will itself recover from errors if the components do. Nothing could be further from the truth.

As more and more business functions are integrated, automated error recovery and resource management become increasingly important and complex. Errors propagate as rapidly as correct results — and the unanticipated consequence (of these accumulating) errors can be devastating.

Businesses naturally focus on automating business functions deemed necessary to their core competencies and processes whose completion is 'mission critical'. This is a one-way street. With each success in automating a particular business transaction, the value of connecting and integrating additional disparate automated transactions increases, creating ever more complex business transactions. With each integrative step, the need for automated transaction management and error recovery becomes ever more important.

ACID burns

Traditional approaches to automating transaction management emphasize the means to guarantee the fundamental properties of a 'formal' (correctly designed) transaction. These properties — Atomicity, Consistency, Isolation and Durability — are usually referred to by their acronym, ACID. Transactions, especially if complex, may share access to resources only under circumstances that do not violate these properties, although the degree to a transaction manager enforce the isolation property is often at the discretion of the user.

The standard interpretation of the ACID properties leads to specific behavior when one or more of the elements that compose a transaction fail due to an unrecoverable error (generally, one that cannot be transparently recovered). As usually understood, the atomicity property demands that:

- either all of a transaction's changes were successful and the resources are left in the intended final state
- or the initial state of the resources is restored (or unaltered) as if the transaction had never begun.

Thus, the final state resulting from transaction execution is:

- either the initial state
- or a specific intended state.

An unrecoverable error always results in restoring the initial state, typically through a 'rollback' process that effectively preserves the initial state — at least until the transaction succeeds or fails. An alternative is to restore the initial state via an 'undo' or 'inverse' transformation — known as a compensating transaction (about which more later). This method presumes that, for every error, the most suitable compensating transaction can successfully be identified and implemented.

Enforcing atomicity as currently understood often wastes viable work when the initial state is recovered. Additionally, transactions dependent on a failed transaction cannot begin until the failed transaction is resubmitted and finally completes, thereby possibly resulting in excessive processing times or even ultimately failing to achieve the intended business purpose.

The consistency property guarantees the correctness (the deterministic operation) of transactions by enforcing a set of consistency conditions on the initial and final states of every transaction. A consistent state satisfies a specific set of consistency conditions, known in advance, and applied to all transactions in a database or application. Thus a correctly written transaction, when applied to resources in an initially consistent state according to known consistency conditions, transforms those resources into a second (pos-

sibly identical) consistent state. The consistency of intermediate states, created as the component operations of a transaction, are applied to resources and are uncontrolled and unknown.

One problem with this approach is that consistency must be either cumulative during the transaction, or else enforced at transaction completion. Most transactions are assumed to have been written correctly for the required consistency conditions; transaction completion is simply assumed to be sufficient to ensure a consistent state. This leads to a further problem: interactions among transactions constituting a complex business transaction may not result in a consistent state unless the combined consistency conditions are enforced automatically at completion of the collection.

The isolation property demands that concurrent transactions accessing the same resources behave as though each is running in isolation (is independent). This is usually interpreted as meaning that side effects are avoided by preventing other transactions from seeing any intermediate states and usually accomplished by 'pessimistic concurrency control' — locking any resource the transaction touches, thereby ensuring that other transactions cannot modify such resources nor access modified resources.

The most commonly used algorithm for ensuring this is known as 'two-phase locking'. While a transaction is processing, locks on all resources it (that transaction) accesses are acquired during phase one and are released only during phase two — with no overlap in these phases. Obeying this locking protocol algorithm enables interleaving of concurrent or dependent transactions while preserving the isolation property, thereby implementing a form of dynamic scheduling.

This interpretation of isolation necessarily increases the processing time of concurrent transactions that need to share resources, since a locked resource may not be modified by any other transaction until the locking transaction completes. Another problem is that such isolation occasionally creates a deadly embrace or deadlock condition among two or more transactions. In the simplest case, each of two transactions wait indefinitely for a resource locked by the other. Usually, a deadlock is broken by aborting one or both transactions, possibly at considerable cost.

Optimistic concurrency (time stamping) and lock or conflict avoidance (nested transactions, multi-versioning, static scheduling via transaction classes or conflict graphs) also use this interpretation of isolation. Various caching schemes, compatible with a concurrency control scheme, improve concurrency by minimizing the time required to access a resource. Yet existing approaches, and the associated techniques and implications for resource management, fail to meet the needs imposed by today's complex, possibly distributed, business transactions.

The durability property guarantees that the final state of a successfully completed transaction is impervious to system failures, that it is 'durable'. It is intended to guarantee that a completed transaction's specific result can be recovered at a later time and so cannot be repudiated. Traditionally, this is interpreted as meaning that the transaction's final state has, in effect, been recorded in non-volatile storage before confirming the successful completion of the transaction. Usually, some combination of resource states is recorded, along with the operations that have been applied to the resources in question, by a log manager. The recording process is usually known as 'commit' and the step in a transaction at which a 'commit' is processed is known as the commit point.

One way to make an intermediate state recoverable is to request a 'savepoint'. Savepoints are arbitrarily designated so they need not represent a consistent state and are typically not durable. The system will create or return to a specific savepoint only at the explicit request of the user. Savepoints cannot be asserted automatically by the system except by the most rudimentary rule — for example after every operation, periodically or based on resource usage. None of these enable the system to determine to which savepoint it should rollback after a particular error.

When transaction actions are executed (whether concurrently or sequentially) under multiple, independent resource managers, the rollback and commit processes can be co-ordinated so that the collection behaves like a single transaction. In essence, the interdependent actions are implemented as transactions in their own right, but are logically coupled to maintain the ACID properties to the desired degree for the collection overall. Such transactions are called distributed transactions.

This co-ordination is normally achieved via some form of two-phase commit, an inefficient process which tends to reduce concurrency and/or performance. A system failure during a two-phase commit can result in an incorrect state that then requires difficult, costly and time-consuming manual correction during which the system is likely to be unavailable.

Compensating transactions can sometimes restore the initial state of a collection of logically coupled transactions (known as a compensation sphere). In such cases, it may be

necessary to run multiple compensating transactions successfully in a specific order, thereby 'unwinding' the entire collection.

Error handling is a significant aspect of transaction management. With current approaches, complex, distributed business transactions are as likely to fail almost as often as they succeed.

Transaction behavior is difficult for workers to understand because it does not fit with peoples' expectations of how the real world handles problems (we rarely throw away our work rather then think in terms of errors and their attempted correction). Because they do not offer an opportunity to record both the error and the correction applied, adaptive improvements are harder to derive. Much of the value of the experience (how the mistake was made and how it was corrected) is discarded after the correction is completed. On top of all this, error recovery is relatively inefficient.

Coupling inter-dependent transactions while ensuring traditional consistency and atomicity extracts an excessive resource cost. It introduces difficult to manage failure modes and a high cost of error recovery. Yet any attempt to avoid the high overhead of distributed transactions may introduce inconsistencies. The techniques may be compatible only with flat transaction models. But these usually do not provide the required business transactions — and business processes cannot then be implemented.

Traditional transactions are insufficient

As outlined above, the usual interpretation of the ACID properties introduces a number of difficulties. Taken together, these interpretations result in less than optimal use of resources and inefficient error recovery mechanisms. The traditional techniques for preserving the ACID properties — optimizing resource usage and recovering from errors — cannot be applied effectively in many business environments involving complex transactions, especially those pertaining to global electronic commerce and business process automation.

There are numerous optimizations and variations on the traditional transaction model, including split transactions, nested transactions, weakened isolation or consistency requirements, etc. In practice, all these approaches have one or more significant disadvantages.

The idiosyncrasies of the traditional transaction model make the translation of a business transaction into a set of computer transactions an error-prone art that few can learn. Non-technical business workers are even less capable of recognizing the business intent of computer transactions, making the correct navigation of an application all the more difficult.

Such problems have a negative impact on businesses. The efficiency, correctness and auditability of automated business transactions has a tremendous influence on a business's profitability. As transaction complexity increases, the impact of inefficiencies and errors increases combinatorially. Businesses work in a complex, imperfect world, and attempt to impose their own order on events. Constantly in flux, they persist in imposing 'acceptable' states through the efforts of all their employees, from the CFO reviewing yearly, quarterly, daily or even 'real-time' performance reports, to the zealous (or indifferent) stock clerk managing physical inventory.

Introducing a new transaction model

Over the years, many transaction models have been created to address particular transaction characteristics. For example, the saga model was designed to address long-running transactions.

These models typically had an associated cost (such as weakening one or more of the ACID properties or permitting incorrectness under certain conditions) or are applicable only to transactions possessing certain characteristics. Other transaction models redefine, for example, transaction scope, commit points, nesting behavior — and other aspects.

The Adaptive Transaction Model1 begins with a classification of transactions. Like the traditional transaction model, transactions are defined in terms of their adherence to ACID properties. However, the Adaptive Transaction Model reinterprets each of the ACID properties, extending them in a natural way while removing certain restrictions imposed by the traditional interpretations. On the other hand, these interpretations reduce to the traditional interpretations under appropriate circumstances. In addition to the usual ACID properties, the Adaptive Transaction Model adds an auditable property, resulting in what can be called call the A2CID properties.

Transactions can be classified broadly into three types, with corresponding qualifiers or adjectives:

- physical
- logical
- business.

A physical transaction is a unit of recovery; that is, a group of related operations operating on a set of resources that can be recovered to an initial state as a unit. The beginning (and end) of a physical transaction is thus a point of recovery. A physical transaction should have the atomicity and durability properties.

A logical transaction is a unit of consistency; it is a group of related operations working on a set of resources that together meet a set of consistency conditions and consisting of one or more co-ordinated physical transactions. The beginning (and end) of a logical transaction is a point of consistency. In principle, logical transactions should have all the ACID properties.

A business transaction is defined as a unit of audit; it is a group of related operations working on a set of resources that together result in an auditable change comprising identifiable, co-ordinated transactions. If each of these component transactions are logical transactions, business transactions combine to form a predictable, well-behaved system. The beginning and the end of a business transaction are thus audit points — which means that an auditor can verify the transaction's identity and execution. Audit information obtained might include identifying the operations performed, in what order (to the degree it matters), by whom, when, with what resources, that precisely describe possible decision alternatives were taken in compliance with which rules, and that the audit system was not circumvented.

Business transactions can comprise other business transactions. They can:

- last as short as microseconds
- span decades (covering, for example, life insurance premium payments and eventual disbursement which must meet the consistency and audit conditions imposed by law and policy).

There are many reasons to group a set of operations on a set of resources together (for example, to define a unit of work). It is common to refer to any such group informally as a transaction — even though the group does not satisfy any particular requirements.

For the purposes of this analysis and in keeping with this practice, the term:

- transaction is used without a qualifying adjective or other modifier when referring to a unit of work of any kind — whether having formal properties or not
- pseudo-transaction is used to refer to a unit of work that does not preserve all the ACID properties (or our specification and extension of them) required by its classification (it may preserve some).

Pseudo-transactions exist for many reasons including:

- the difficulty of proper transaction design and enforcement
- incomplete knowledge of consistency rules
- attempts to increase concurrency via decreased isolation
- attempts to increase performance at the expense of atomicity
- and so on.

Orthogonal to the foregoing classification, a transaction may have certain other characteristics. It may be implicit or explicit according to whether its boundaries are implicitly or are explicitly declared or identified. A group of transactions will be said to be 'co-operating transactions' if they share at least one intermediate state or coordinated action.

ACID with less burn = A2CID

Within the Adaptive Transaction Model, the atomicity property is refined to mean that either all effects specific to a transactions will complete or they will all fail. Notice that it does not require the transaction itself to succeed or fail, only its effects. In ATM we no longer require the transaction to be effectively predefined with respect to either recovery on failure or the final state to which it transitions on success.

Loosely stated the Adaptive Transaction Model permits transaction to be recovered to any state that does not invalidate work already done or to transition to a final state that satisfies an identified and recorded set of consistency conditions, possibly different from those satisfied by the initial state. These requirements provide atomicity with the viability of traditional atomicity, but with more flexibility.

The consistency property is refined to mean that whenever a state satisfying a first identifiable set of consistency conditions in a category is connected by a set of operations to another state satisfying a second identifiable set of consistency conditions in a category, those operations constitute a (possibly implicit) transaction. An identifiable set of consistency conditions requires a computational decision procedure for determining whether or not a state satisfies the consistency conditions. The consistency conditions partially characterize the state.

The usual interpretation recognizes consistent states only at transaction boundaries as it assumes consistency conditions are fixed and known in advance. Instead, Adaptive Transaction Model permits multiple intermediate and boundary consistency states with each possibly defined by multiple sets of consistency conditions in a category.

This introduces two new concepts:

- categories of consistency condition sets
- consistency points.

Through the course of a transaction, the set of resources may enter a consistent state from time to time; this is referred to as a consistency point. Such a consistent state may be detected automatically, or may be manually asserted by the user (for example, via program code, directives or interactive commands). Consistency points may be durable or non-durable, as required by the circumstances under which they may be used. In effect, a consistency point is a savepoint with the added requirement of consistency and the optional property of durability. (Note that a transition from one consistency point to another constitutes an implicit transaction.)

The isolation property is refined to mean that no two transactions produce a conflicting or contradictory effect on any resource on which they are mutually and concurrently (that is, during the time they are processed) dependent. This makes explicit the intuitive notion that isolation is relative to consistency — that it is not required as long as consistency is not at risk. Sharing of intermediate states and resources is only conditionally precluded; thus transactions can be co-operating. Somewhat more formally, the isolation property holds if the state resulting from any group of concurrent, interleaved, co-operating transactions is consistent with a possible history comprised of a serialization of the implicit and explicit transactions in the group.

The durability property is refined to mean that the final state of a transaction must be recoverable insofar as that state has any effect on the consistency of the history of transactions as of the time of recovery. Thus, if the recovered state differs from the final state in any way, the durability property is a guarantee that all those differences are consistent with all other recovered states and external effects of the transaction history. Unlike traditional durability, the final state of a transaction's resources need not be recorded in non-volatile storage but may be computed. Furthermore, until there is a dependency on it, the final state may be considered one of a class of possible states.

A new property, the auditablity property, applies specifically to business transactions as discussed above. A business transaction is auditable if it is uniquely identifiable and its set of audit characteristics — initial state, final state and some partial ordering of the set of operations consistent with both the system transaction history and the initial and final states — are each well-defined and identifiable. In general, a business transaction will have an associated set of auditability constraints, such as who authorized or performed which operation using which resource.

The foregoing interpretations enable a logical transaction to be understood as a transition from one state in a class of consistent states to a state in another class of consistent states. Each consistent state in a class is defined by a set of consistency conditions, so that two states in the class may satisfy different sets of consistency conditions. Logical transactions thus provide no inherent guarantee as to which consistent state in the class of achievable states will result, unless the consistency conditions restrict the class to one consistent state given the selected initial state.

These refinements to the ACID properties and logical transactions permit a more realistic implementation of business transaction processing. Under familiar restrictions, the new ACID definitions reduce to the old definitions compatible with existing transaction managers. Both traditional transaction processing methods and altogether new methods are permitted.

The new methods make it possible to manage complex transactional environments, while optimizing the resource usage. They extend to distributed transactions, and to business transactions which span both multiple individual transactions (for example, in a business process) and multiple business entities (as is required in electronic commerce and business-to-business exchanges).

Methods and applications

The Adaptive Transaction Model enables numerous transaction processing methods and optimizations. Three important examples will be discussed here including:

- establishing consistency points which minimize the cost of recovery under certain types of error
- transaction relaying which permits work sharing across otherwise isolated transactions, while simultaneously minimizing the impact of failures
- corrective transactions which permit error recovery without unnecessarily undoing work, without so-called compensating transactions while enabling the tracking/correlation of errors and correction.

By contrast with prior approaches, these methods are extensible to complex transactions and distributed business environments. They are also particularly well-suited to business transactions and business process management. Each is applicable to pseudo, physical, logical, business and distributed transactions — all with predictable consequences.

Consider consistency points. Whereas a commit point traditionally requires a fixed set of consistency conditions, a consistency point is associated with a set of identified consistency conditions which it satisfies. These consistency conditions determine how the consistency point may be used. (Note, however, that this interpretation of the consistency property permits considerable flexibility in defining consistency points.)

Various applications of consistency points are almost immediately apparent, most of which do not require that flexibility, including:

- automatic deadlock recovery: rather than being aborted, a deadlock victim can rollback to the nearest consistency point; alternatively, on encountering a deadlock, a transaction can automatically rollback to a consistency point and retry in an internal loop, which is often sufficient to break the deadlock
- automated savepoints: consistency points can be detected automatically and asserted by the system as savepoints
- categories of consistency points: multiple sets of consistency conditions can be defined so that multiple categories of intermediate states can be automatically detected
- categorized rollback: by associating a class of error with a set of consistency conditions that define a consistency point, transaction rollback on detection of an error to an appropriate consistency point can automatically initiated
- optimized commit processing: durable consistency points can be used to distribute commit processing automatically throughout a long running or complex transaction rather than at the commit point
- power failure recovery: after a power failure, transaction recovery can proceed from the nearest durable consistency point.

Transaction relaying is a method of sharing intermediate states among a group of transactions. The principle caveat is that the intermediate state must occur at a consistency point of the originating transaction and the associated consistency conditions should be compatible with the receiving transaction. A protocol for recovery determines which transaction controls each resource depending on the commit state of transactions in the group.

Under many circumstances, transaction relaying may be used in place of distributed transactions, chained transactions or other methods of coupling a group of inter-dependent transactions. The transaction relaying provides a means for efficient, consistent management of inter-dependent transactions without violating atomicity or isolation requirements, without introducing artificial transaction contexts while enabling resource sharing and recovery.

Suppose that a particular business process consists of transactions A and B and that there is an integrity rule or constraint or a dependency that requires transaction B to follow A — because it relies upon the work done by A. In other words, some portion of the final state of resources affected by A (the output of A) is used as the initial state of resources required by B (the input of B). The final state of A is a consistency point, even before A commits.

The usual approaches demand that one:

- either accept the possibility that the final state of A is altered by some transaction C before B can access and lock the required resources (the sequential transaction scenario)
- or accept the possibility that the state of resources needed by B is different than the state of those same resources as perceived by some other transaction (chained transactions)
- or run transactions A and B combined in a distributed transaction, accepting the fact that all resources touched by either A or B will be locked until B completes (the distributed transaction scenario).

Transaction relaying recognizes the fact that A and B may share the state of the resources that B requires at least as soon as A enters the final consistency point for those stated resources and has made that final state durable (assuming durability is required). Unlike chained transactions, it need not wait until A is ready to commit. It need not even wait until locks are released. Rather, the Adaptive Transaction Manager either transfers ownership of those locks directly to B or establishes shared ownership with B (as long as only one transaction has ownership of exclusive locks on a resource at any given time if the ACID properties are desired) — and never releases them for possible acquisition by C.

Unlike the sequential transaction scenario, there is no pos-

sibility that C will interfere in the execution of B. Unlike the chained transaction scenario, transaction relaying does not require transaction A to have committed, the beginning of transaction B to occur immediately after the commit of transaction A, the commit of A and start of B to be atomically combined in a special operation (indeed, B may already have performed work on other resources), transactions A and B to be strictly sequential or transaction B to be the only transaction that subsumes shared responsibility for resources previously operated on by transaction A. Unlike the distributed transaction scenario, resources held by A (but upon which the initial state of B does not depend) are released as soon as A completes: there is no two-phase commit overhead. Unlike split transactions, transaction relaying does not introduce artificial transaction contexts, can be fully automated without sacrificing consistency, and yet enables collaborative transaction processing in which work groups can communicate about the status and intermediate results of their work (including negative results).

Transactions A and B can do additional work on other resources, prior to and after the consistency point, respectively. Transaction A can also do work on the shared resources after the consistency point discussed above, so long as transaction A alters no consistent state on which transaction B depends. Other transactions can have a similar relationship to transaction A, involving possibly different resources or consistency points.

The new ACID properties are preserved if exactly one transaction has responsibility for shared resource modification at any particular time, and that transaction can rollback the state of those resources to the most recent consistency point in which they are involved. (Note that consistency point durability might not be a recovery requirement — as, for example, during deadlock recovery.)

Transaction relaying lets transactions 'publish' their states and/or consistency conditions at consistency points and permits other transactions to 'subscribe' to the state of associated resources. Numerous methods may be used to determine which of the subscribing transactions will gain write permission over the associated resources and in what order.

Corrective transactions provide an alternative to both compensation and rollback when the desired result of a transaction can be understood as producing a state meeting a particular set of consistency conditions (for example, maintaining a balance of debits and credits across a set of bank accounts). Unlike compensating transactions that address only error repair, corrective transactions effectively enfold both error repair and correction.

When an unrecoverable error occurs, the failed transaction is returned to a recoverable consistency point. The error is classified and the corresponding consistency conditions on the affected resources. A corrective transaction is then invoked to transform the affected resources from the most recent consistency point to a state (possibly approximating the intended state) that satisfies an alternative set of consistency conditions (the 'acceptable conditions'). The choice of acceptable conditions constrains the final state to some acceptable state: they may be completely distinct from the initial set or just more general category. Many methods may be used to 'discover' an appropriate corrective transaction. If no such corrective transaction exists, the failed transaction is returned to an even earlier consistency point, and an appropriate corrective transaction invoked. The process is repeated until the acceptable condition(s) in the category are satisfied.

For example, consider a simple business process consisting of two predefined but parameterized transactions, a funds-transfer transaction (parameterized for transfer amount and two account numbers) and a loan transaction (parameterized for loan amount but with fixed account number). If an attempt to transfer a specified amount between two accounts fails because of insufficient funds, an automatic corrective transaction might loan the user the required funds, thereby expanding the consistency conditions to include an account not owned by the user.

In this example, the corrective transaction might be manually predefined by the bank and caused to run as part of an error handling routine. Similarly, rather than debiting the explicitly specified account (for example, checking), it might debit an alternate account (for example, savings or an investment account). Because alternative sets of consistency conditions can be added or removed, the method is more modular and adaptive than trying to anticipate and code alternatives within the transaction.

Corrective transactions replace the rigidity of fixed consistency conditions with a category of sets and invoke an auxiliary set of actions (the corrective transaction) that will transform the current state into one satisfying some set of consistency conditions in that category. Options for achieving a final consistent state are thus broadened. For each set of consistency conditions defining a transaction's final state, the other sets of consistency conditions in its category constitute acceptable consistency conditions.

This concept mimics the real world of business, in which errors are common and a strictly pre-determined result of work is not possible. Rather, business workers must achieve an acceptable result, where acceptability is determined by

alternative sets of constraining conditions and often associated with business risk and opportunity assessment.

The method of corrective transactions requires that each transaction be identified by the consistency conditions that will be enforced or that such consistency conditions be automatically discoverable by the system. Such consistency conditions might, for example, be stored in a repository accessible to the transaction manager, other appropriate software or even a user.

Management conclusion

Research on implications of the Adaptive Transaction Model is on-going. In addition to the applications discussed here by Mr. McGoveran, it fosters new methods for advanced concurrency control, resource management and scheduling.

On the less technical side, its appropriateness for many business application has been described. These include asset exchanges where the parties do not have an initial agreement as to the value of the particular elements, or even agreement as to the particular elements that are the subject of the proposed exchange, beforehand, to allow intermediate positions to be evaluated and the costs and benefits of concessions and tradeoffs explicitly to be assessed.

Certain types of business can be expected to benefit from the Adaptive Transaction Model. These include:

- ***telecommunications rerouting***
- ***inventory management for retail or distributional operations (that frequently encounter spillage, wastage or theft)***
- ***electronic funds transfer message repair***
- ***financial transactions affected by governmental fiats***
- ***billing systems reflecting or affected by collection processes, debtor failures and even bankruptcies.***

1 The phrase 'adaptive transaction model' has appeared in the transaction research literature in conjunction with a different model than the one described here. As the usage in this analysis was both independent and contemporaneous with that literature, the author has chosen to retain its use and name in the hope that the vast differences will be readily apparent to the informed.

Model-driven application development

Peter Bye, Consultant and
Alan Hood, Unisys Systems and Technology

Management introduction

Much effort has been expended over the life of the IT industry in trying to accelerate and otherwise improve the process of new application development. Language and runtime environments of increasing sophistication have been produced, of which Java EE and the .NET framework are currently the most popular. Many tools are available to help developers use these environments effectively.

However, both Java EE and .NET runtime environments are complicated, and Java and C# are essentially third-generation languages. These factors can lead to a concern with technology in the form of programming language and runtime environment — rather than focusing on the real purpose of application development: converting user requirements into working code and databases.

In this analysis Peter Bye and Alan Hood describe why they believe that model-driven development is a much better approach. They argue that this universal approach is used in other branches of engineering. No one would build a bridge or an aircraft, never mind a nuclear power station, without extensive qualitative and quantitative modeling before production, involving collaboration with all the stakeholders. They suggest the same should be true for software development.

The rational for formal models

Informal and formal models have long been part of software development. In the context of this analysis, however, we are concerned with a specific approach. Tools should work at a high level, above programming languages and application execution environments. Plus — the key point — having modeled an application, all the necessary code — together with database schemas and a runtime environment and including middleware — should be generated automatically. Furthermore, the target runtime environment should be selectable at generation time allowing, for example, applications to be generated from the same model for Java EE, .NET as well as other environments, different databases and multiple operating systems.

The idea of generating systems directly from models, without programming (in the sense of using a programming language such as Java or C#), is not new. Today there are successful implementations, which achieve most or all of these goals. What is surprising is that they are not more widely used.

It is worthwhile, therefore:

- **looking at the kind of application developments typically required today**
- **discussing the development processes (including modelling)**
- **describing an example of a model-driven development product.**

What will remain puzzling is why there is a continuing use of essentially third-generation approaches. Possible explanations will be assessed.

New development requirements

In order to respond rapidly to changing business requirements, organizations are under pressure to deploy new IT application services in ever decreasing timescales. Recent years have seen an emphasis on re-using existing applications as components in distributed environments, most notably within the framework of a Service Oriented Architecture (SOA). There is a growing body of evidence that this approach — when compared with developing all functions from scratch — reduces the time to deliver and reduces the risk involved (in many cases),.

Users can be offered new services which are produced by the orchestration of services exposed by existing applications. Products such as Enterprise Application Integrators (EAI) and Enterprise Service Buses (ESBs) provide a variety of tools for generating the rules of orchestration — the sequences of and conditions for service invocation — and the interfaces used to access the exposed services.

However, although EAI and ESB products offer increasingly sophisticated orchestration tools, there is a limit to what can be done by orchestration alone. Substantial new applications have to be developed, possibly acting as components in a service architecture as well as offering external services directly.

As a rather simple example, consider a (fictional) bank, which offers a range of products. In this example, there are three existing applications managing financial service products, each running on a different platform:

- **current (checking) and savings accounts**
- **mortgages**
- **insurance.**

Each system is accessible from teller workstations and from a variety of internal users, for example in call centers. Associated with the record of each product in the databases is information about the customer — name, address, telephone number and so on.

The bank also offers its customers direct Internet access to each product application, providing information such as account balances and statements. The product management applications have been wrapped, or adapted as required, to allow a self-service Internet application to access services.

The bank then decides to extend the range of services to it wishes to offer. These will include:

- **enquiries about the status of all products held, without the customer having to access each system in turn**
- **movements of funds between products, for example from a savings account to pay off part or all of a mortgage.**

The implementation of these new services cannot be done by simply adding new orchestration rules. A new customer relationship management (CRM) application is required. The database of the CRM system will contain:

- **the information about the customer, consolidated in one place instead of in each product system,**
- **details of the various products held by the customer and any rules or conditions applicable.**

The CRM application will function as a service provider in the new Internet services. It will also be accessible from workstations within the bank, for example in call centers and branches, to provide information to bank staff about customers when talking to them on the telephone or face to face. Figure 4.1 shows a schematic of the environment following the introduction of such a new CRM system and the expanded Internet application.

The bank's customers can now be offered new services such as 'get all product status' and 'move X amount from savings account to mortgage'.

The CRM application and the three product management systems are service providers to the Internet application, exposing the required services such as:

- **'validate customer' and 'get customer's product details' from the CRM**
- **'get account balance' information from the product management systems**
- **find 'debit account' and 'credit account' from product management systems.**

So, for example, if the customer requests the status of all products, the Internet application obtains a list of the products held from the CRM service, and the status information from each product management system as required. The retrieved information is consolidated and returned to the customer. Such a CRM application is typical of the kind of development required. It is a substantial application, requiring a new database. It also has to function in two environments:

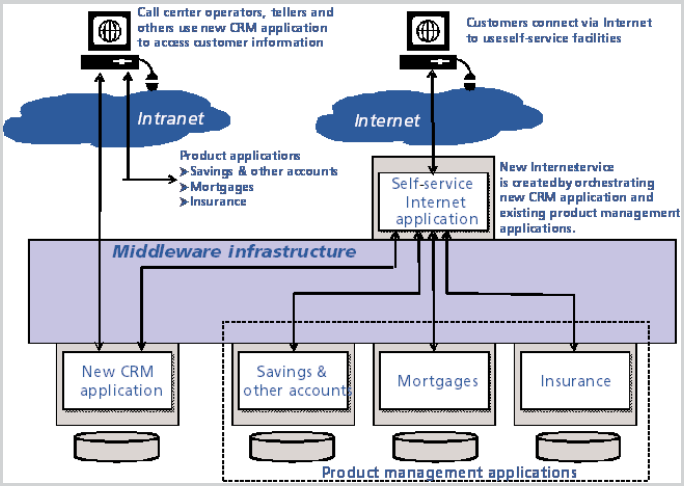
- **as a component in a service architecture, where its services are invoked from the Internet application to retrieve details of products held by customers**
- **by providing a direct user interface (GUI) for access by users — such as call center operators or staff in branches — to obtain information about customers and products.**

How such applications could be developed is the subject of the remainder of this analysis.

The development process and application modeling

The application development process includes all activities from the gathering and formalization of requirements to the generation of the new runtime system. IT developers work in collaboration with business management, users and other stakeholders during the specification, design, implementation and generation phases of the system. The development process should also include activities following initial deployment such as upgrades and enhancements to add new functions or fix problems (Figure 4.2).

Figure 4.1: Bank environment including the new CRM application



As can be seen in Figure 4.2, the result of the development may include the generation of databases and client logic, as well as the application logic in the center, which will need to be deployed into a software environment comprising operating systems, database management systems and middleware. This will depend on the specific requirements; it is true for the CRM application discussed in the previous section, for example.

A great deal of effort has been expended on development methodologies and tools since the early days of computing. This is unsurprising in view of the poor performance of the industry in delivering new systems on time, within budget and with all the required capability. A particular difficulty has been to make the development process seamless — from specification of requirements to creation of the running system. Without this level of integration, dislocations between the various different phases of the process can result in the loss of information and thereafter lead to problems.

The transitions from specification to design and design to implementation have proved notably difficult. It is important, therefore, that methodologies and tools work with concepts that are understandable by users and other stakeholders. They must be implementable within the software infrastructure, suited to the class of problem being addressed and able to work seamlessly from specification to runtime system (Figure 4.3).

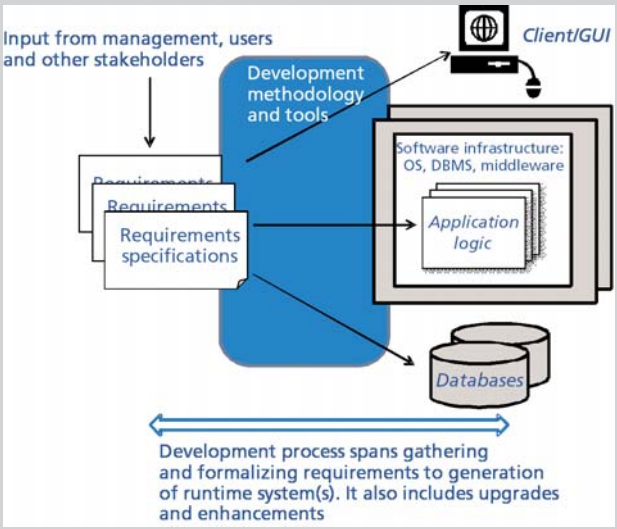
Model-driven development

Model-driven development offers such an approach. Models, in the sense of abstract representations of physical or logical entities, have extensively been used in a variety of fields. Examples include designs for buildings and machines (such as aircraft or motor vehicles). Similarly, mathematical models have a long history of use in science and engineering — representing physical systems, either natural or man-made. Small scale models — maquettes — are used to explore the properties of larger entities without the expense of a full implementation — such as when a model of an aircraft is used for testing in a wind tunnel.

The advantage of models is that they can be constructed and discussed with the intended users or consumers, and can be modified as required — all at a much lower cost and in a shorter timescale than working with a full implementation. When everyone is happy with the model, the full implementation goes ahead.

Although model-driven approaches are currently a subject of much discussion in the IT world, models have always been used in system development, just as they have in any other branch of engineering. Graphical representations may be informal (for example, diagrams drawn during brainstorming on a whiteboard or on the back of an envelope in bar) or more formal. (Flowcharts have been used since the beginning of the IT industry; the various symbols — decision boxes, stored data, processes and so on —

Figure 4.2: Schematic of the development process



were standardized at an early stage.)

More recent developments include the widespread use of UML (Unified Modeling Language), which is itself a tool for producing design artifacts based on standardized representations. Other techniques are also used, for example mathematical models based on queuing theory or simulation (for performance modelling). Indeed, prototype implementations are the equivalent of maquettes in other branches of engineering and proponents of agile development regard prototypes as the best model, with iterations being produced as required to arrive at the desired target system.

A central concern of model-driven development — almost the Holy Grail for IT — is the generation of the resulting runtime system directly from the model. For this to happen, the model must be at a high enough level to be understood by the users and other stakeholders requiring the system — and sufficiently precise to generate the runtime system (thereby avoiding the dislocations caused by transition between different phases of the implementation). In this view, the model is the system. It is transformable by generation processes into executable versions.

Achieving this?

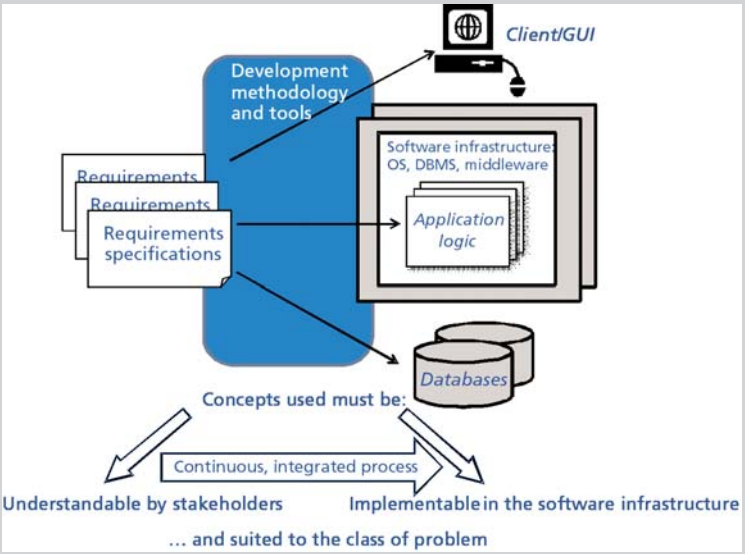
This section discusses the Unisys Agile Business Suite is an example of a product that enables the development of applications — as well as generating the CRM system in the bank's example above — and the subsequent runtime system from the high-level model.

Unisys Agile Business Suite (AB Suite) has its origins in earlier Unisys products from Unisys, such as LINC (first introduced in the mid-1980s) and Enterprise Application Environment (EAE), which was first released in 2000. Each has built on the experience of the earlier products and kept in step with general industry developments.

The goals of AB Suite are to:

- **provide a means of rapid application development based on a model-driven approach which focuses on the business requirements**
- **enable the automatic generation of runtime systems from the model — without the need for intermediate design and coding steps**
- **generate all parts of the application, including database and user interfaces**
- **enable users to have a choice of runtime environment (including widely accepted industry standards) and avoiding being tied to any specific operating system, application execution environment, database management system or middleware**

Figure 4.3: Development methodology and tools



- **support the generation of different runtime systems from the same model, without requiring it to be changed — and to allow different parts of the application to be deployed in different runtime environments**
- **optimize the generated environment to support high volume transaction processing as well as other application environments; this includes the choice of database on various platforms**
- **produce applications able to act as components within an SOA framework; these must be able to include other components, generated with different tools, within a model.**

These goals were also to a large extent those that drove LINC and EAE. Both can generate for Windows, selected variants of UNIX and Linux, as well as for Unisys ClearPath mainframes. For LINC and EAE, the generation process produces COBOL as an intermediate language, which is then compiled for the target environment.

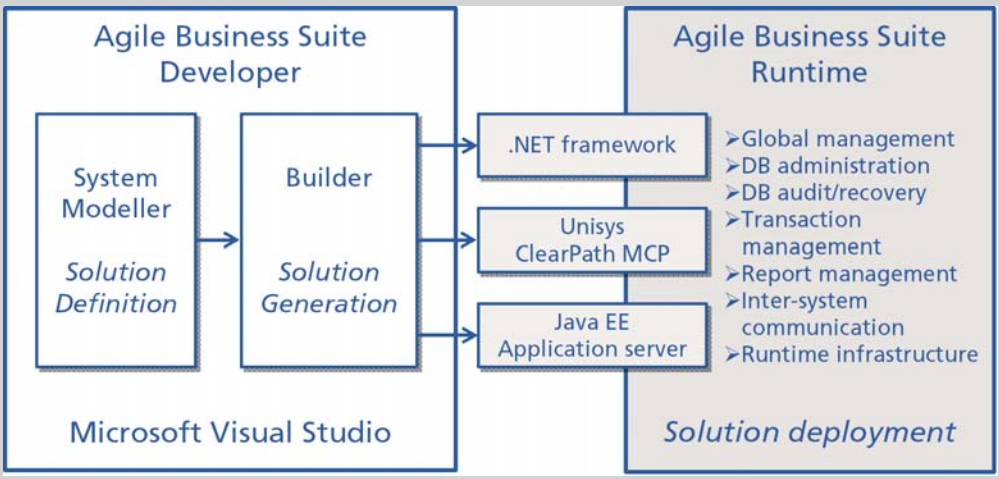
There have also been significant shifts within AB Suite to take advantage of more recent developments. In particular, AB Suite runtime system generation support now includes Windows .NET and Java EE application servers — generating C# and Java as intermediate languages respectively. This is in addition to generating for Unisys ClearPath MCP systems, where COBOL is used as the intermediate language. (Figure 4.4 shows the major elements of AB Suite.) As can be seen, AB Suite comprises two major elements, the Agile Business Suite Developer and the Agile Business Suite Runtime.

AB Suite Developer is used to define and generate applications. It includes the System Modeler, Builder, Version Control and more and is installed on a Windows workstation operating within Microsoft Visual Studio. AB Suite Runtime, which is the runtime system, differs for the various supported runtime environments.

The System Modeler in AB Suite Developer provides a model-based, object oriented development environment. System Modeler also functions as a package within Visual Studio. It is used to define a separate project type, in the same way that various project types can be developed using tools, such as Visual Basic or C#.

Developers using System Modeler define programming objects using a set of pre-defined, stereotyped classes. A stereotype is a pattern, based on established business functionality which has common properties and built-in behavior; using stereotypes can greatly reduce the effort required to produce systems. System Modeler supports the creation of application logic using a high-level scripting language called LDL+ as well as deriving the database structures from its high-level application model. Each class may include presentation, which is defined within AB Suite as a GUI. The

Figure 4.4: Agile Business Suite elements



interfaces can be deployed in a variety of environments, including workstation clients, such as Windows forms, Web interfaces (such as ASP.NET) and programmatic interfaces for custom clients (including Web Services).

Service-oriented environments may contain services generated in a variety of ways. Existing pre-SOA environments have to be accommodated; that is a major purpose of SOA. These may be industry-standard component services implemented using other technologies, as well as existing applications and data in so-called legacy systems. Additional services can be included in SOA environments with new components built with AB Suite. System Modeler enables users to define the interfaces to external components, Java and others, and then use those external objects as part of the full solution model. For example, even though a Java component that has been developed by a third party cannot be modified, its signature can easily be included in the model. The developer is aware of the need to call it from within the AB Suite model, and he/ she can view details of the interface — like its name, input and output parameters.

In the context of a bank, for example, an external class that implements some third party service such as international currency exchange rates can be treated within the model just like the internal class that is used to model a savings account. Calling a method (GetExchangeRate) on the currency external exchange class looks just like calling a method (GetAccountBalance) on the internal savings account class. The difference is that the details of how the savings account is implemented can be viewed and potentially changed, whereas the details of the external service may not be accessed at all from within AB Suite.

Generation and deployment

Sadly, most developers are not concerned with runtime environment details. They do not manipulate the C#, Java or COBOL generated by AB Suite but work at the level of the model from which the runtime system is generated. Generation is performed by the AB Suite Builder component in AB Suite Developer. It generates all the application logic in the form required by the runtime environment:

- **the .NET Framework (C#)**
- **a Java EE application server (Java)**
- **Unisys ClearPath MCP (COBOL)**
- **as well as the user interfaces defined by the developers in System Modeller.**

Different runtime environments may be generated from the same model. The ability to generate for different platforms brings a several advantages:

- **those producing applications for multiple deployments (for example package vendors) are able to generate systems suited to client requirements from just one source model; this helps where organizations have policies for specific technologies which dictate the deployment platform — a policy in favor of Linux, for example, will remove the option for .NET**
- **applications can be generated for different platforms depending on the amount of traffic expected**
- **test and QA environments can be generated for smaller platforms than the production runtime system.**

Databases are automatically derived from the models, specifically tailored to the platform in which the resulting system is to run. A generation for a .NET environment generates tables for Microsoft SQL Server or Oracle database managers. AB Suite determines what tables are required from the model, defines the primary and secondary indices and generates SQL procedures required to perform potentially complex queries that are needed by the application.

Developers do not need, therefore, to define tables explicitly or write any SQL. If components of the same application are generated to run on a different platform, for example a Unisys ClearPath MCP system, AB Suite will use the native database management system of that platform — creating the required structures and access methods.

AB Suite runtime includes services and libraries that enable AB Suite applications to communicate with other components, using both proprietary and standard methods and protocols. Where appropriate, AB Suite integrates with the database management system and transaction management of the deployment environment, for example with a Java EE application server. If the platform does not provide such native management facilities, the AB Suite Runtime environment delivers them.

The ability to generate the entire runtime system brings additional benefits in application maintenance and extension. Many tools have been developed to speed up the initial development and deployment of applications, or application components.

However, unless there is an obvious bi-directional link between the objects in the model and the program units, database structures and user interface components that implement the object, even minor changes can be tedious and error prone. Generating 100% of the application and

database automatically from the model means that changes to requirements do not require the development staff to locate and modify every database table, user presentation and transaction program that might be impacted by the change. They simply change the model and rebuild the application.

Why has the model-driven approach not been adopted?

We chose AB Suite as an example of what can be done with model-driven development because we are familiar with it and because it exhibits the characteristics required for this approach. Experience has shown that applications developed using AB Suite — or its predecessors, LINC and EAE — can be completed quickly and reliably, and perform well in the deployment environments.

We have shown that model-driven development is a viable alternative to more traditional, 3GL-oriented methodologies. Key benefits include improved communications between the users and stake holders, and the IT personnel who produce and maintain the applications. Developer productivity is improved, resulting in a greater responsiveness to changing user requirements. Although the experience with LINC stretches back more than 20 years, the ideas of model-driven development — generating applications from a high-level business-oriented description of the problem — go back further. For example, Teichrow and others in the 1970s worked on defining PSL/PSA (Problem Statement Language/Problem Statement Analyzer) — which was aimed at producing systems from a high-level definition. Apart from LINC and its successors, other products have appeared generating part, or in some circumstances, all of the runtime system.

Why, then, has what is essentially a 3GL approach persisted for so much application development? Generating from a model should result in a reduced development time, as well as the ability to deploy in different runtime environments. We suggest three possible, linked explanations.

The first is that people are unaware that model-driven products are available; there are still only a few, with varying capabilities. In essence people have trouble believing the claims made about speed of development, and the ease and value of automatic generation of runtimes.

The second is inertia or technical conservatism; developers establish a development environment with which they feel happy and they may not wish to leave their comfort zone. It does not take the possibility of the radical step of introducing model-driven development to send some traditional developers hurrying to the barricades. The persistence in the use of ancient text editors for manipulating source code illustrates the reluctance to change. Some of the technical conservatism may also stem from not believing the claims of the tool suppliers. Some will claim to generate the entire application when in fact the result may be rather less. And there may be a concern over the quality of the generated code. One of the reasons for the downfall of many of the 4GL CASE tools from the 1990s was that the generated code was too generic or verbose and did not perform well. Although technology has greatly improved since then, it is still essential to pick a tool suited to the business and application requirements.

The third possibility is more psychological, and has to do with the value and status developers feel about themselves as well as a deep fascination with technology. Environments such as Java EE are complex. Many IT people enjoy learning and understanding the complexity — and feel that they have a value to their employers by having this knowledge. This feeling of a sense of value — of being indispensable — is particularly marked in an age when many people feel insecure about their jobs, as is the case today with the ever-present threat of off-shoring. Application development has often been seen as a lower form of activity than writing basic software. The ability to raise its status by persisting with complex environments needing a lot of understanding is, therefore, all too tempting to application development practitioners.

Management conclusion

Runtime environments are complicated, and Java and C# are essentially third-generation languages. As Mr. Bye and Mr. Hood have described, model-driven development is a sounder approach for building applications than traditional methods. Using what other branches of engineering have practised and proved for years would seem to make sense, at least from an organizational perspective — especially when the whole (including the middleware components) are produced.

But, as they also describe, people are almost certainly the problem. Whether inertia or reluctance or disbelief is the more accurate (or a self-reinforcing combination of these), the result is that model-driven is not being adopted on the scale that it should be.

IDEs for middleware — a beginner's guide

Trevor Eddols
Managing Director
iTech-Ed

Management introduction

Middleware might be thought of simply as software that goes in between two or more applications so they are able to exchange data in a controlled way. This communication between the applications could take place on the same hardware or could occur across a network. What's needed is an easier way of developing the software — an enabling middle layer. This is, in part, where IDEs come in.

An IDE — an acronym that stands for Integrated Development Environment — is simply software that is meant to help with the development of more software. Logically this could include much needed middleware.

IDEs have a mixed history, although they are becoming much more frequently used now. At one stage, it has been humorously suggested, companies selling IDEs would claim the letters stood for 'It Does Everything'. Later, the poor user would feel that a better fit for the acronym would be 'I Do Everything'.

In this analysis, Trevor Eddols makes the connection between IDEs and middleware before continuing to consider:

- ***Eclipse (and Rational)***
- ***NetBeans***
- ***IntelliJ***
- ***JBuilder***
- ***JDeveloper***
- ***BEA Workshop.***

IDEs emerge

IDEs grew out of the need developers experienced when using simple text editors. Using a text editor, like Notepad on a PC or vi on UNIX, enables you to enter your program code. A text editor can also be a highly efficient way to create simple HTML pages. It offers cutting and pasting, so programs can be created quickly using parts of other programs. Most developers have used these at some stage in their careers. But that is about where their usefulness ends.

What the IDE offers in addition is a compiler and a debugger. That means the code that has been typed in — or cut-and-pasted — can be tested before it goes to the next stage. So, just to spell it out, an IDE usually comprises:

- **a source code editor**
- **a compiler (and/or interpreter depending on the programming language)**
- **a debugger.**

Using an IDE, therefore, speeds up the generation and production of working code. Now this code may need a GUI (Graphical User Interface) so that it can be used by its intended end users. Many IDEs can help with this and — because the first attempt at creating the software may not work fully or work in quite the way the end users hoped (requiring the developer to create a second version, and then a third, etc) — many IDEs now offer version control as well. Those IDEs that are used for Object Oriented (OO) software development can go further still — by integrating:

- **a class browser**
- **an object inspector**
- **a class hierarchy diagram.**

Many IDEs are language specific. This means if you are planning to create a program in a particular language, you use the appropriate IDE. For example, Boa Constructor can be used to create a program in Python.

Others are platform specific. For example, Apple Mac users can use Xcode as an IDE (this product is not available on other platforms). Other IDEs, like Eclipse, can be used with more than one programming language on more than one platform.

Visual IDEs

Many IDEs are visual programming environments. Users can create their desired application by creating flowcharts on-screen by moving around the appropriate icons. The icons represent blocks of code and they are linked together with arrows. The final flowchart is then compiled to create the required program. Visual programming like this is meant to be easy to learn (basically boxes and arrows) and is meant to make program development faster and less error prone.

Users (and non-users) often hold strong opinions about this. The flowcharts produced are usually based on the Unified Modeling Language (UML – q.v. www.uml.org), created by the OMG (Object Management Group – q.v. www.omg.org). The current UML specification is at Version 2.1.1 (early 2007).

Currently the most popular (and most used in enterprise development) IDEs are:

- **Eclipse**
- **NetBeans**
- **IntelliJ**
- **JBuilder**
- **JDeveloper**
- **BEA Workshop.**

Of these, perhaps the best-known IDE is Eclipse (www.eclipse.org). Although Eclipse is now an Open Source project, it began life in IBM Canada. Originally, IBM had VisualAge, which was really a number of different IDEs for different programming languages (though its development was closely linked with SmallTalk). IBM needed a replacement for VisualAge, and Eclipse was the result.

In 2001 a consortium was formed so Eclipse could be developed as Open Source rather than proprietary IBM technology. In 2003 the Eclipse Foundation was created.

Why did IBM want Eclipse to be Open Source? The answer is complicated, but it is probably because it (IBM) sensed that Open Source was going to popular with developers, and individuals working on their own would extend the technology in ways IBM perhaps would not have thought of introducing. It also meant that a robust IDE would be available to developers that was linked to neither Microsoft nor Sun Microsystems — both strong competitors to IBM at that time.

Eclipse itself is written in Java, and was originally a Java IDE. It can now be used for Java, PHP, C and C++ — plus other languages. The Eclipse Web site says that Eclipse projects

provide tools and frameworks that span the entire software development life-cycle, including:

- **modeling**
- **development**
- **deployment tools**
- **reporting**
- **data manipulation**
- **testing**
- **profiling.**

It goes on to say that the tools and frameworks are primarily focused on building J2EE, Web Services and Web applications.

The Eclipse Consortium has been working on the Eclipse Project, which is responsible for developing:

- **the Eclipse IDE workbench (which is where Eclipse tools are hosted)**
- **the Java Development Tools (JDTs)**
- **the Plug-in Development Environment (PDE), which is used to extend Eclipse.**

The Eclipse Tools Project concerns itself with developing tools for Eclipse: this includes the C/C++ IDE and the COBOL IDE. The Eclipse Technology Project looks after research and development and education using Eclipse.

In addition to the usual facilities that an IDE provides, Eclipse offers:

- **a syntax highlighting editor**
- **incremental code compilation**
- **a thread-aware source-level debugger**
- **a class navigator**
- **a file/project manager**
- **interfaces to standard source control systems.**

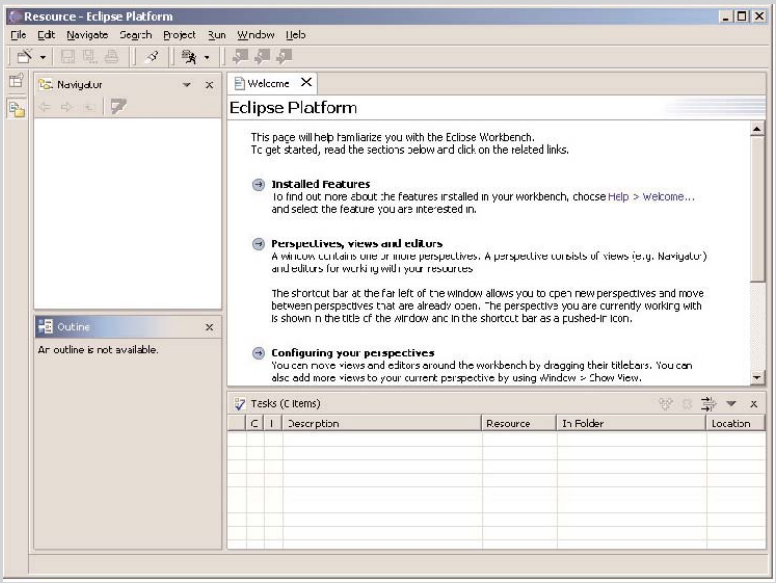
Pre-built binaries can be downloaded to install Eclipse on:

- **AIX**
- **HP-UX**
- **Linux**
- **Mac OS X**
- **Solaris**
- **Windows.**

In order to run Eclipse, users need an appropriate Java runtime. Eclipse can then be downloaded and unzipped into whatever directory they wish. On Windows, you have a file called *eclipse.exe*, which when run for the first time, completes the installation. After that, users will see the window (Figure 5.1).

Clicking on File/New/Project allows users to create a

Figure 5.1: Initial Eclipse window



new project. Once a project has been created, the layout of the Eclipse window changes. The Outline view is now on the left side of the window, the Navigator is replaced with a Package Explorer. This is called the Java Perspective (Eclipse comes with a number of default perspectives).

The next stages are to create the directories that will contain the source code and then add the project code. Next the code is run and debugged — again using menu items. Once it works, it can be made available to end users.

More than Eclipse

Eclipse is not the only IDE that COBOL users might choose. LegacyJ's IDE (www.legacyj.com/perc_ide.html) is available on a range of Linux, UNIX and Windows systems. However, it is itself based on the Eclipse IDE and it permits the development of mixed language (COBOL and Java) applications.

ATX software has COBOL Studio (cobolstudio.atxsoftware.com). This IDE is for COBOL applications. It too is an Eclipse plug-in.

IBM supplies the IBM Rational Application Developer (RAD) for WebSphere Software. Rational was bought by at the end of 2002. Today the Rational Software division of IBM is responsible for the software. Unsurprisingly, it is based on Eclipse and allows for the visual design, construction, testing and deployment of Web Services, portals and J2EE applications.

Looking at its name, it is not surprising to find that it has a built-in WebSphere test environment and, being a Rational product, it is tightly integrated with other Rational tools, such as:

- **ClearQuest (for configuration management)**
- **ClearCase (for version control).**

The Rational Application Developer includes:

- **code and visual editors for database connections and SQL**
- **Enterprise Generation Language**
- **HTML**
- **Java**
- **JavaServer Faces and JavaServer Pages**
- **UML**
- **Web Services**
- **XML.**

NetBeans and add-ons

If you want to try something other than Eclipse or one of its derivatives, you could try NetBeans

Figure 5.2: Sun Java Studio Creator Welcome screen

(www.netbeans.org/downloads/index.html). It is currently at Version 5.5.1. Like Eclipse, it is extensible and is also an Open Source IDE. NetBeans 6 is due soon.

NetBeans enables developers to create mobile applications as well as Web and desktop ones. NetBeans runs on Linux, Windows, Mac OS and Solaris. NetBeans offers all the usual IDE tools, such as window and menu management, settings storage, etc — but in many ways is considered to be fairly basic and not best suited for Web applications.

Sun Java Studio Creator, on the other hand, is intended to excel at creating Web applications. The product provides a visual programming environment. The drag-and-drop features are based on JavaServer Faces technology. Users simply manipulate the JavaServer Faces components using the GUI (illustrated in Figure 5.2).

Also available on the Solaris platform (like NetBeans and Java Studio Creator) is Sun Java Studio Enterprise. This offers:

- **Web application development**
- **debugging**
- **support for Web Services and J2EE (Java 2 Enterprise Edition) application development.**

This product provides a model-driven analysis, design and development environment that uses UML. Users can make use of features such as:

- **the live roundtrip (bidirectional) markerless model**
- **code synchronization**
- **code-reverse engineering.**

Both Java Studio Creator and Java Studio Enterprise are built on — and extend — the facilities and features available with NetBeans. As with Eclipse, NetBeans encompasses an ecology, not just one tool.

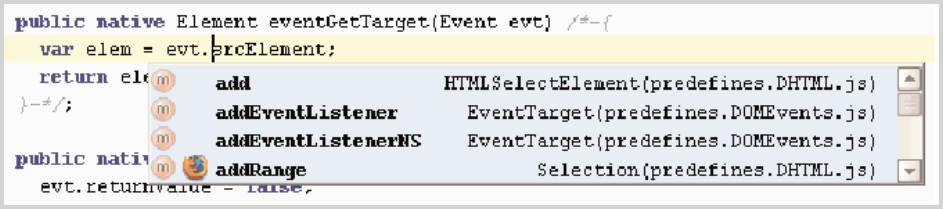
IntelliJ

IntelliJ IDEA (www.jetbrains.com/idea), currently at Version 7.02, is a Java IDE from JetBrains. It currently boasts that more than 400 plug-ins are available for it and runs on Windows, Linux and Mac OS platforms.

This product originally appeared in 2001. It was one of the first to offer a set of refactoring tools. These enabled programmers to redesign their code.

Part of the appeal of IntelliJ IDEA is the inclusion of integrated form design, and the fact that it integrates with Open Source tools such as CVS and Apache Ant:

Figure 5.3: IntelliJ IDEA AJAX application development



- **CVS (Concurrent Versions Control) keeps track of changes to files thereby providing version control**
- **Apache Ant automates software build processes.**

Written in Java, it uses XML to describe the build process and its dependencies. IntelliJ IDEA allows users to create AJAX applications with its native support for AJAX technology. Figure 5.3 illustrates AJAX application development using IntelliJ IDEA.

JBuilder

JBuilder is a Java IDE from CodeGear — although originally from Borland. The first version was written in Delphi, although by Version 3.5 it had been written in more conventional Java.

The ability for users to add on extra tools led to Oracle using the product as the basis for its original JDeveloper tool. In 2006 Borland turned its Developer Tools Group into a wholly-owned subsidiary called CodeGear, which now owns JBuilder. JBuilder is now built on Eclipse.

New in JBuilder 2007 are ProjectAssist and TeamInsight. The former is designed to shorten and simplify the creation of team and project definitions. TeamInsight is meant to enhance collaborative development with its centralized portal, enabling team members to:

- **monitor project activity for the source code repository**
- **track recent check-ins**
- **view quality metrics**
- **see live burn-down charts for project progress.**

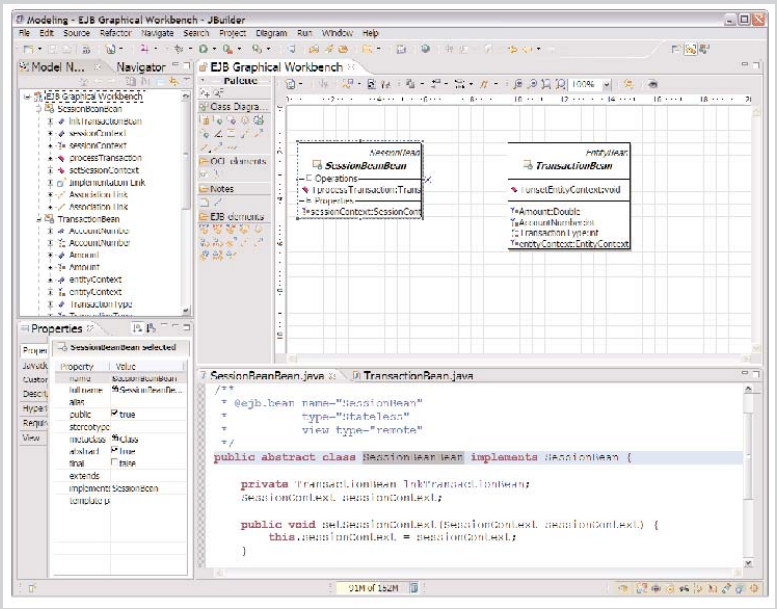
Optimizeit 2007 for Eclipse delivers memory and CPU profiling and debugging. Figure 5.4 shows an example of the JBuilder LiveSource screen, which simultaneously replicates changes to models in the code.

JDeveloper

JDevloper (www.oracle.com/technology/products/jdev/index.html) from Oracle is currently free (since 2005) to Oracle users. It can be used to develop applications in:

- **BPEL**
- **HTML**
- **Java**
- **JavaScript**
- **PHP**
- **SQL and PL/SQL**
- **XML.**

Figure 5.4: Example of a JBuilder Live Source screen



JDeveloper integrates with the Oracle Application Development Framework (Oracle ADF), which is an end to end J2EE-based framework intended to make application development simpler. The original version of JDeveloper, back in 1998, was based on Borland's JBuilder. In 2001 the product was completely rewritten so it was based on Java. This was the 9i version.

The current version (10.1.3.1) was released in January 2007, although a preview of 11g came out in May this year. JDeveloper can be used to turn a Java class into a Web Service. It will also generate the necessary WSDL and all the JAX-RPC components. JDeveloper provides declarative interfaces for the creation and definition of EJBs. This is illustrated in Figure 5.5.

BEA Workshop

BEA Workshop (http://www.bea.com/framework.jsp?CNT=index.htm&FP=/content/products/weblogic/workshop/) is a family of commercially-available products — also based on the Eclipse IDE. The products in the family are:

- **BEA Workshop for WebLogic V10.1**
- **BEA Workshop Studio V10.1.**

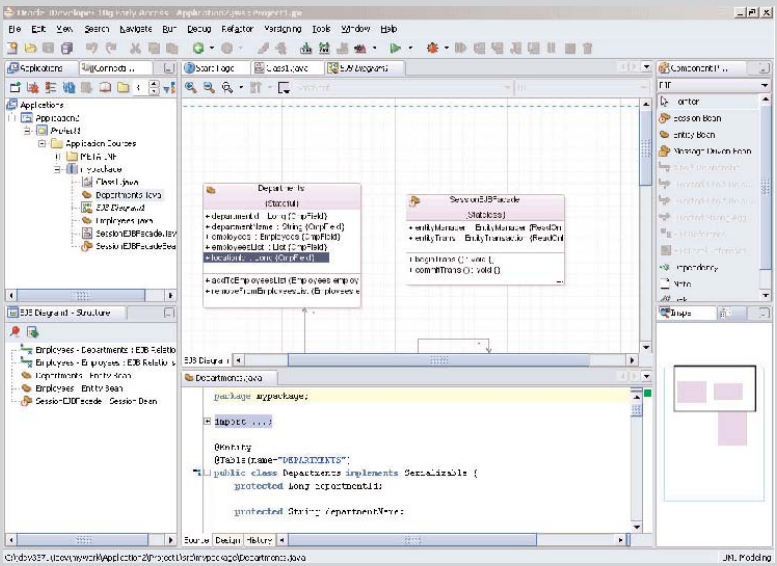
These are aimed at programmers in SOA (Service Oriented Architecture) environments. BEA Workshop for JSP provides a commercial Eclipse Java Server Page (JSP) editor and is free. It provides a JSP source editor as well as a WYSIWYG visual JSP editor, which support simultaneous source/visual editing.

Management conclusion

There are many other Integrated Development Environments available. Many of them are based on Eclipse: some are little more than feature-rich text editors while others can be used on a variety of platforms for a range of programming languages and/or are highly specific about which platforms they run on and with which languages they can be used.

The key conclusion is that all of these — and the ones described above — have developers (and organizations) that use them regularly and that are familiar with their strong points and their peculiarities. As Mr. Eddols has described, all of them are oriented to reducing the time it takes a programmer to create the applications — including the middleware that connects the different software pieces.

Figure 5.5: JDeveloper's EJB creation



Source for Figures 5.1-5.5:

Figure 5.1:
http://www.onjava.com/pub/a/onjava/2002/12/11/eclipse.html?page=1

Figure 5.2:
http://www.softwarereality.com/reviews/creator_ea.jsp

Figure 5.3:
http://www.jetbrains.com/idea/features/ajax.html

Figure 5.4
http://www.codegear.com/article/36556/images/36556/JavaLiveSource1.PNG

Figure 5.5:
http://www.oracle.com/technology/products/jdev/collateral/papers/1013/jdev1013_overview.pdf

Risk management and middleware projects

Nick Denning
Chief Technology Officer
Strategic Thought

Management introduction

Nick Denning is the Chief Technical Officer of Strategic Thought Group Plc (Wimbledon, UK) which he founded in 1987, having previously worked for Logica. Strategic Thought, since inception, has been involved with 'big software', from Ingres into the middleware market with Tuxedo before WebSphere middleware products and the services around them.

In 2001 Strategic Thought launched its first product, called Active Risk Manager. This is now a leading enterprise risk management solution, with customers from backgrounds as diverse as NASA, Lockheed Martin, the UK's Ministry of Defence, London Underground, Thames Water, Nestle and the 2012 Olympic Games.

Much is talked about risk management (RM). Many people, however, find it difficult to adopt risk management techniques. In this analysis Nick Denning illustrates — through the use of selected features of an enterprise risk management solution — the significant benefits for managing projects that can be obtained quickly and with minimum training. In particular he relates this to middleware projects.

What is risk?

What is risk? It is uncertainty. The underlying premise of Risk Management (RM) is that all endeavors and investments serve to increase share/stake holder value, but all investments and endeavours face degrees of uncertainty:

- **are the requirements unambiguous?**
- **will the hardware be delivered on time?**
- **have the suppliers been late before?**
- **are the team members properly trained?**
- **are there any flaws in the design?**
- **will the customer sign off promptly, as defined in the contract?**
- **is an SLA in place?**
- **will the SLA guarantee delivery?**

Most will be able to identify many more known issues which must be resolved as risks and which will have to be addressed in any attempted endeavor. The issue is how account for this risk? The 'old school' approach is to slap on a healthy contingency as a percentage of the total cost and time for the project. However, how many projects (with what seems an appropriate amount of contingency) are still late and over budget? The 'new school' approach is risk management. In this analysis I take an example project and show how one can:

- **structure a project plan in such a way that the approach reflects good practice to minimize risks inherent in a middleware project**
- **apply three point estimation to a good project plan and then apply schedule analysis to determine the likely outcome**
- **identify opportunities and risks that would impact the plan, either positively or negatively, and describe how to deal with these.**

The benefits of following the approach described are that:

- **organizations obtain a more finely grained and accurate mechanism for calculating contingency**
- **the tasks and their associated costs, time and resources required to address risks are built into the plan from the start so the plan is more accurate and realistic**
- **a knowledge of risks is shared, and specific responsibilities for addressing risks are clear and defined in the project initiation phase.**

In the following sections I illustrate aspects of this approach as applied to a middleware project. (If further detail is required the author offers to provide this.)

Applying risk management techniques to a middleware project

In the August 2007 **MIDDLEWARESPECTRA**, I identified a range of risks that apply to middleware projects. That analysis was able to identify that many of those risks related to people risks. Indeed I identified seven key risks — and all of these related to people factors.

Using modern risk management techniques it is possible to provide a considerably enhanced degree of precision around this process. This comes, however, with a price:

- **the process has to be defined**
- **people have to be trained**
- **there is a cost of carrying out this process to generate the risk information**
- **organizational change must be managed to exploit the risk management information generated.**

The objective of this analysis is to demonstrate the application of simple risk management techniques to smaller projects — what might be called 'light touch' risk management — in order to obtain a clear benefit from the implementation of a simple process.

Is that effort worth it?

The IT industry has seen initiatives come and go, with associated specialist practitioners 'gold plating' their discipline to generate deliverables that are ends in themselves rather than supporting objectives that deliver an organization's goals. Is enterprise risk management a similar 'fad' that will pass soon? I believe not. This is because, inherently, risk management is something that we all do on a daily basis in almost every aspect of our lives. The issue is, therefore, not whether or not an organization will practice risk management, but rather whether or not an organization will incorporate RM into its business processes using a software solution because a return on investment (RoI) can be identified.

Experience on very large programs shows that risk management provides a substantial benefit. It can be difficult to measure the benefits of RM at a micro level — how do you measure the cost of something that has not happened? However organizations that have adopted RM into their processes observe that the they have avoided major project failures. Teams of 'specialist risk practitioners' who are responsible for the successes in avoiding failure have developed increasingly complex risk processes. Based on the lessons learned, organizations are currently implementing risk simplification processes. It is necessary to remove the

'gold plate' and focus on 'simple processes, quickly executed' that use accurate data and deliver well managed risks.

Recent legislation is increasing the personal liabilities of directors. Companies must now report on the risks that their organizations face and are called to account if those risks have not been properly managed. Thus executives have an incentive to introduce risk management systems to ensure that their organizations can identify, assess and manage their risks learning from experience and developing the overall capability for risk management within an organization. Indeed, the greatest obstacle to the introduction of an RM solution is not cost but:

- **managing organizational change**
- **training staff**
- **motivating people to undertake risk management**

because each person involved must see the benefit that arises.

Implementing organizational change is difficult as those that have implemented BS5750, ISO9001, AQAP 13, CMM and others will attest. We generally advocate a combined top down (corporate-led) and a bottom up (project, practitioner or evangelist-led) approach to capture accurate data with which to ensure that all staff involved in managing risks. We have identified that, as the capability of such experts increases, then risk management becomes more accomplished.

An example

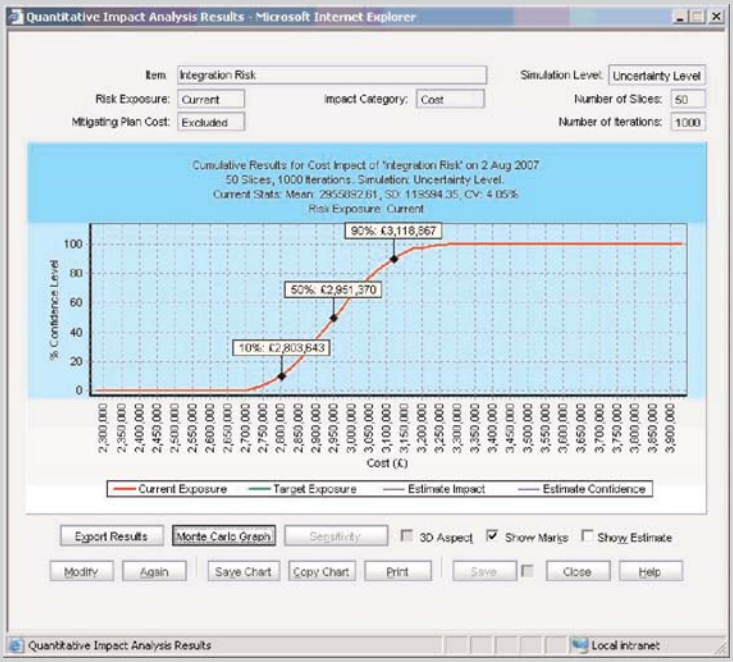
To attempt to understand this, I am going to use an example middleware project and apply some straightforward risk management techniques to it. To set the scene in perspective, and to illustrate the benefits obtainable, understand that the work described represents about 3 man days of effort. This includes:

- **the construction of a project plan**
- **the allocation of people**
- **the assessment of a single risk and a single opportunity.**

Remember, in this context, that the objective is to try to avoid treating risk management as a science — which is expected to deliver right/wrong and go/no-go answers — but rather as a process aid. For instance it is not essential to capture every risk that might be relevant to a project. Rather the imperative is to focus on the major risks.

The simple process that might be followed goes as follows:

Figure 6.1: Quantitative Impact Analysis



- **identify the requirements to deliver against**
- **identify the tasks that need to be completed to deliver those requirements (rather than following the normal practice of just adding 10% contingency at the end to every task)**
- **assess every task and say what is the shortest time in which it is possible to complete, the most likely time and the longest time — and then apply the same to the associated costs.**

Many project plans are simple task lists which might be equally well set up in Excel. In practice there is limited project management going on in them. It is, however, necessary to continue to build a plan by identifying the dependencies between tasks. Providing one can link the tasks, one can schedule analysis in a risk management tool.

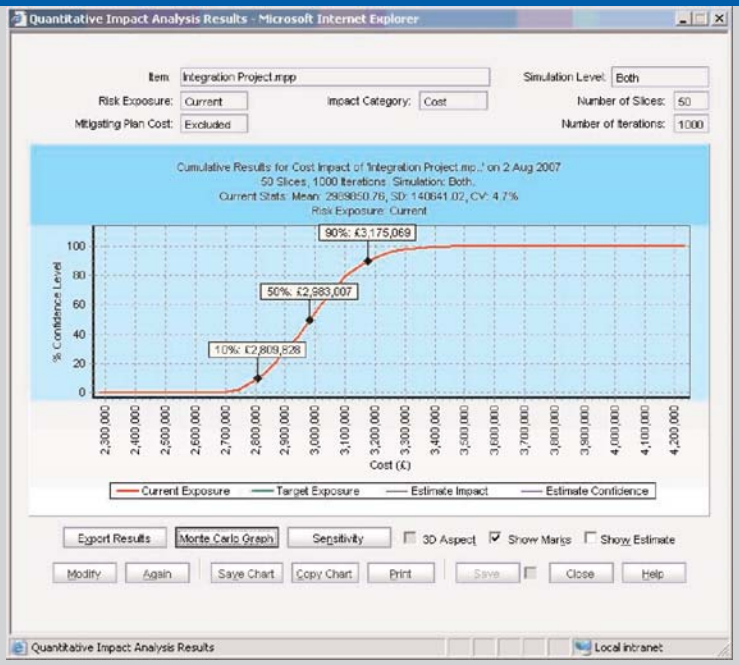
Microsoft Project (MSP) is quite flexible. Having created a first cut project plan in MSP and defined the expected cost and duration for each task I can now do three point estimation to assess the best case, most likely and worst case outcome for each task. From a Gantt chart one can selec (say) Add Columns and include the columns Cost1, Cost2 and Cost3 together with Duration1, Duration2 and Duration 3. It is then possible to have the locations hold three point estimates; now copy the initial cost and duration values into Cost2 and Duration2 and assess each task and enter the best and worst cases for cost into Cost1 and Cost3 respectively. Then repeat this for duration.

With this, I can start to load the project into a risk management tool and analyze the project directly for the most likely predicted out come in terms of date and cost, using Monte Carlo simulation techniques. This will walk through all the possible routes through the tasks using the three point estimation and a probability distribution to determine a range of probable outcomes for the project. In contrast to 'simply' just adding up all the worst case numbers and coming up with a huge project cost and length (which tends to produce a result that assumes that it is very unlikely that everything is going to go wrong), this will produce a more sensible result.

Now it is time to run quantitative analysis tools on the system to look at likely cost outcomes and schedule outcomes. There are currently no risks in the system so the project will be analyzed based on the uncertainties defined for the project tasks. The project estimate (Figure 6.1) was approximately £2.80M based on the summary totals of the tasks. This analysis tells us that there is:

- **a 90% chance of bringing this project in at £3.12M**
- **a 50% chance of delivering at £2.95M**
- **a 10% chance of delivering at £2.80M.**

Figure 6.2: Predicted outcomes with risks



Now I can perform additional calculations for separate segments of the project, selecting a particular part of the work and breaking down the structure to be analyzed. Using the built-in math, this gives an estimate that has (this far) taken just a few minutes to create. Most people do not even need training to reach this far, though most need a little guidance from a risk consultant in the organization.

Adding risk and mitigation

At this point I can start to add risks. To keep it simple, a pretty straightforward risk has been selected. In the project there is a particular point where the design has been confirmed and it is time to order hardware. The assumption is that a 20 day lead time is sufficient and this has been incorporated in the project plan by putting a 20 day lag between tasks.

But what happens if the equipment cannot arrive within 20 days from when the order was placed? In putting in the cost some assumptions have been made about the cost and delay to the project and team — who are all fired up and ready to go but suddenly without the systems environment on which to work. A team of eight people doing nothing for two unexpected weeks incur a significant cost.

To analyze this, first the risk is entered. Next the risk is scored to quantify the cost and likelihood of the risk occurring after which I can consider any mitigation activities which, if carried out, would reduce the likelihood of the risk occurring and the cost/time impact if it did occur. There might be a number of things that can be done — these do not necessarily have to cost money. Nevertheless it is necessary to record these in order to plan ahead.

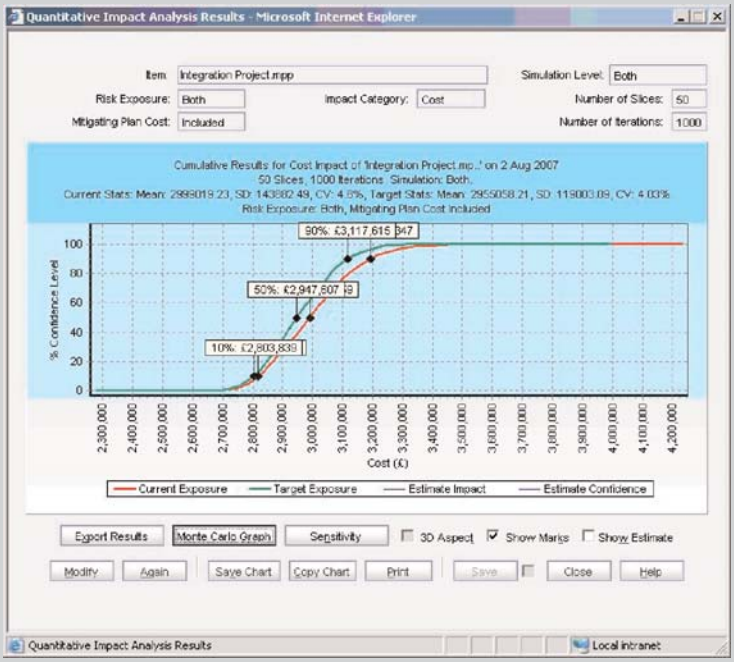
Risk assessment

Having entered this risk into the system one can now reassess the project. Figure 6.2 provides an assessment of the exposure to the specific risk showing that the predicted outcome at the 10%, 50% and 90% probabilities rise to £2.80M, £2.98M and £3.18M respectively.

The need now is to investigate the likely outcome if I invest in mitigation activities in order to try to reduce the risk to the target level. This can be shown on the green line in the S-curve below (Figure 6.3) — giving a reduction in the expected outcome cost of 10%, 50% and 90% to the £2.8M, £2.95M and £3.12M — and suggesting that it is worthwhile to carry out the mitigation tasks. (Typically one would analyze many more risks and would expect a greater potential benefit having analyzed all the risks in this way.)

So far, so good. The risk process has been addressed and assessed. Ways to mitigate it have been introduced and costed. The analysis has determined that it makes sense to carry out the mitigation. Now it is time to:

Figure 6.3: After risk and mitigation



- **add the new tasks (that enable the mitigation to occur) back into the original plan**
- **manage down the risk to the green target level.**

(It is possible, of course, to identify mitigating actions where the cost is not justified, and the green line is actually further to the right than the red line. These are mitigations that I would probably ignore.)

Has enough contingency been allowed?

Typically a project manager will agree a percentage of the total project cost and time for contingency. But how accurate is that? In the above analysis task by task I have considered the required contingency for each task. Due to the detailed analysis exposed to all, there should be much greater confidence in the initial plan and the associated costs and time predicted in Figure 6.1 using schedule analysis.

I then analyzed the risks of the project and identified the mitigating activities and their costs before including in the plan the tasks attributable to the mitigation activities. I also made provision for the resource that will be required to carry out those mitigations. The result is that integrity of this planning process is a major improvement over simply adding 10% to the time and budget.

Opportunity investigation

Risk is usually considered threatening. This is not the whole story, however. One should also use risk management tools to investigate opportunities.

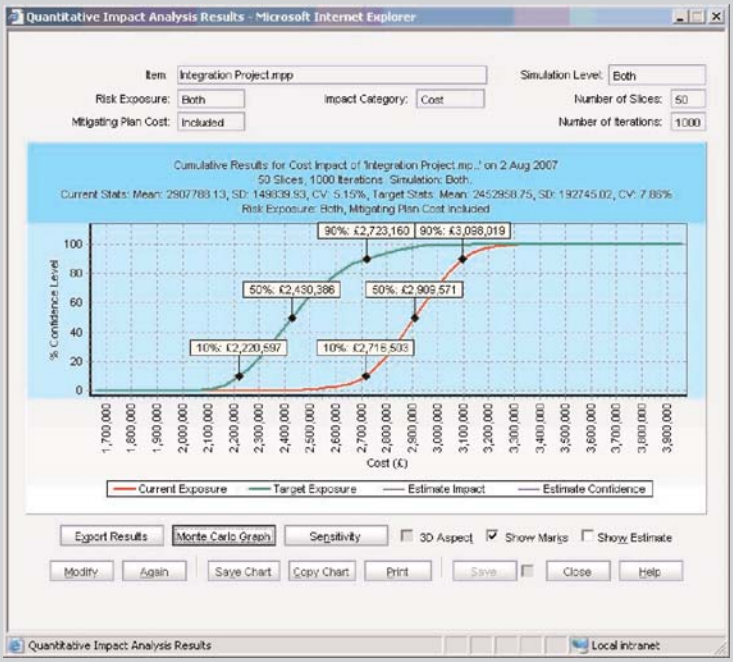
For instance, when building middleware solutions, a project typically uses many environments for development, test, UAT and so on. These can be extremely expensive to construct. I will now consider the opportunity to exploit virtualization, using products like those from VMware.

Such an approach can reduce the cost of the hardware and software required. It may also be able to deliver a more effective Business Continuity/Disaster Recovery solution.

Let me assume, however, there is some organizational skepticism (that perhaps VMWare's technology is not to be trusted). I will, therefore, build into the plan a model which considers the opportunity for savings, including the cost of investing to demonstrate that VMware is a viable technological option — and following the same processes as those described earlier. This results in an S-curve (Figure 6.4).

Impact analysis shows the current (red) and target (green) expected outcomes — and includes the cost of mitigation — demonstrating a significant improvement. Thus the investment in virtualization software and the costs of

Figure 6.4: Analysis for an opportunity



demonstrating to the design authority that it is appropriate are justified by the savings to the projects that will be achieved.

Despite the organizational skepticism surrounding virtualization, it should be possible to build a business case, using this analysis, to explain that there is a 90% chance of saving £350K and a 50% chance of saving £580K on the overall project. It then depends on good project management to ensure that the tasks on the project plan are followed through.

Key middleware project risks

In the introduction above I referred to the key risks relating to middleware projects. Time and space prevent me setting up and analyzing each risk. Instead I will now select one of those risks and see what further benefits can be obtained from risk management when managing those risks.

Organizations can build up a knowledgebase of generic risks related to their industry and specific risks that apply to their organization, and ensure that risk managers consider whether or not the risks apply to their project. It is important to maintain the confidence of the business sponsor that the project is progressing under good management. By producing project plans and risk plans and then delivering progress in accordance with the plan that should maintain this confidence.

Indeed, even if a risk does impact the project, having demonstrated that the risk was identified, assessed, that mitigation action was taken and that contingency plans have been prepared, the project sponsor should be confident that the team can deal with the issue.

Management

With the risk assessment complete the priority is to manage the project through its lifetime to ensure that all the mitigation plans are carried out as planned. In effect the need is to manage risks down.

Figure 6.5a shows a waterfall chart which shows when the mitigations entered above need to be completed. To manage the project all I need to do is return to the risk management tool and update the risks as each of the project stages are encountered and completed. All of these changes are then audited as historical data. They provide the basis for the reports.

How does the project sponsor know that risk management is being undertaken? The following fragment from a report (Figure 6.5b) shows whether each person is completing the relevant risk actions by the due date.

Management conclusion

The key conclusion to be drawn from Mr Denning's analy-

Figure 6.5: Waterfall (6.5a) and report fragment (6.5b)

ARM
Active Risk Manager

Risk Owner / Risk Level	# B/F this month	# New	% Approved	% Reviewed	% Reviews On Time	% Reviews Late	# Closed C/F	#Open
Fred Smith								
NIL	4	0	0%	0%	50%	50%	0	4
Medium	2	0	0%	0%	100%	0%	0	2
Low	3	0	0%	0%	67%	33%	0	3
High	1	0	0%	0%	100%	0%	0	1

sis is that one can take a typical project plan (even one prepared in Microsoft Project) and create three point estimates for each task to model contingencies on a task by task basis. Using industry standard risk management software, within a few key strokes one can obtain a good idea of the likely outcome for any project based on the best assessment of the tasks that were identified.

Using a knowledge base of risks, one can consider which tasks might be affected by those risks. From this one can assess the likelihood and cost of each risk and predict the overall impact on a given project. The same can occur for opportunities to reduce the cost of the project.

With this information organizations can then determine whether there is a return on investment that justifies undertaking additional tasks in order to mitigate risk further. This enables management to answer the perennial question of how much contingency should be allowed. When there are a number of projects competing for scarce resources and limited budgets this approach offers a clear view of the likely cost that must be set aside for each business activity and enables the selection — or discarding — of projects for implementation. The decisions will be based on a risk adjusted return on investment calculation, one that has been carefully assessed.

In addition there is an ongoing reporting framework — to check that everyone carries out their designated tasks so that risk is progressively reduced. Rather than have to wait until key milestones, organizations can have confidence that the project management approach is effective and they can obtain early warning when projects go off track and why.

This is remarkably simple. Almost every operation given in this simple example demonstrates how straightforward this can be. As so often in system, and just as applicable to risk as in other areas, the best management practice conforms to the KISS principle — keep it simple, stupid.

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

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

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

Email and Internet
Email:
spectrum@middlewarespectra.com

World Wide Web:
www.middlewarespectra.com

ISSN 1356-9570

[incorporating FINANCIAL MIDDLEWARESPECTRA ISSN 1460-7220]