A CTO reflects on the evolution of middleware

Nick Denning
Chief Technology Officer
Strategic Thought

Management introduction

Nick Denning is the Chief Technical Officer of Strategic Thought (Wimbledon, UK) which he founded in 1987, having previously worked for Logica. Strategic Thought, since inception, has been involved with 'big software', from Tuxedo (in its various guises/owners including BEA where the company helped BEA set up in the UK and ran its European support center for a year) to Oracle to IBM and Microsoft.

Strategic Thought initially was primarily a services provider with a specialization in high availability and transaction middleware implementations. 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, Nestlé and BP.

In this discussion, Mr. Denning reviews 20+ years of working with middleware (of one form or another). He looks back, but also forward to examine what is happening today and what he thinks must evolve for tomorrow. In particular he emphasizes the need for a rethink of the role and responsibilities for 'architects' within an enterprise.

Where it started ...

From my perspective, middleware became serious with the development of communications interfaces between applications using TCP/IP in the late 1980s, primarily driven by the City's need for high speed communication of price data feeds between financial applications and those provided by the likes of Reuters. Once one organization implemented these interfaces, suddenly we saw many organizations implementing similar interfaces. To do this, however, most were ignoring the reliable and available products — such as SNA and DECnet — which were platform dependent and instead were 'rolling their own' solutions on UNIX, which was a substantial challenge. Yet this willingness to 'roll your own' was the basis for much subsequent innovation — the technology was good and did what it said 'on the tin' — but it still needed substantial coding to provide high throughput and to be robust and reliable in the event of failure. Quality of services without transaction integrity was an often ignored problem.

These early 'roll your own' IP implementations were often well done. But they were also extremely difficult to maintain, not least because they were delivered by some of the cleverest people around who had no pressing commercial need to make the results simple. I saw people using DEC technologies to communicate between mainframes using DEC MessageQ and similar techniques. But this was horrendously complicated, and expensive, to keep running. Once this was realized there was a strong move to productize these communications interfaces.

A simple alternative was FTP. This was rapidly accepted and became trusted. Much systems integration was done — indeed is still done — by the exchange of files. The fact that FTP was free and rapidly available on almost all platforms — certainly once UNIX had become popular, including the proprietary versions — created one of the earliest and still most accepted forms of middleware standards. One consequence is that we still have a huge legacy of FTP implementations, which remain productive today (file transfer is easily understood and undertaken). However FTP is also dangerous because of the same quality of service problems associated with shell scripts that fail to trap and manage errors during communications.

Around the early to mid-1990s, the SQL-net technologies became established, supporting the two tier client/server architecture of database applications. Indeed, for a while, many people thought middleware was 'just about' SQL connectivity. When organizations tried to increase the number of users accessing two tier applications across a slow WAN, Transaction Processing (TP) monitor application — such as CICS, TUXEDO and ENCINA — began to be deployed on the new open systems as key enabling middleware for standard business applications. TUXEDO dominated as the enabling n-tier application middleware of choice (until it was finally superseded by multi-threaded J2EE and COM/MTS application servers). It was an excellent and highly reliable middleware technology that was equally frequently underrated. Although in some ways it had similarities to SQL, its key difference lay in the way it enabled applications to communicate using the ATMI API. Applications linked via this API could be decomposed so that one could place these to minimize network traffic and message latency. The messaging capabilities also enabled applications holding a database connection to receive instructions from many users, process the task on behalf of the remote user and return the reply — thus removing the dreaded cursor processing across the network and enabling tens of thousands of users to access a database which had a limit of (say) 500 concurrent connections.

DCE (the Distributed Computing Environment) tried to bring standards to this market. Although the technological base was sound, DCE was too complex; it used fundamentally synchronous communications (when TUXEDO had both synchronous and asynchronous models) and was difficult to understand. The result: it never quite got going, particularly with the arrival of Java and the benefits of compile-once, deploy many-times approach.

The next big middleware breakthrough was DCE's antithesis — the introduction of reliable asynchronous message queuing. This occurred with the introduction by IBM of MQSeries (now known as WebSphereMQ) around 1996 (it had a 10th Anniversary celebration in 2006). The concept of asynchronous messaging was a big move forward because it enabled developers to decouple applications and to buffer messages. This meant that applications could 'fire and forget' in the certain knowledge that the reliable queuing and message transfer would not forget any messages.

One consequence of synchronous messaging is a huge network of applications where, if any one element (hardware, operating systems, communications, database or application) fails, then its associated applications fail as well. Synchronous processing had been, and in many ways still is, the dominant form of inter-application communication (it is certainly the form that most developers understand, however 'inefficient' it can be in a distributed environment). It is also much simpler to program.

Asynchronous messaging decoupled these elements. It was in stark contrast to most roll your own implementations which were synchronous — you called another application and you waited until the response returned (and, if no

response returned, you stayed waiting). Decoupling simplified at many levels. Yet developers also had to think through the potential issues when using the asynchronous programming mechanism within an application, such as those associated with timing, causal effects of order and time and the complexity of locking, blocking and multi-threading. Even now this explains the relatively limited and slow take up of AJAX and publish and subscribe.

Microsoft confirmed the validity of asynchronous messaging when it introduced MSMQ and included it within its Windows NT and Server 2000 products. In my personal view I believe Microsoft missed the point with MSMQ — in that it (MSMQ) was and is a Windows-only technology. The whole point of asynchronous application messaging technology lies in its ubiquity, the ability to communicate between all the platforms in an enterprise (for example MQSeries supported within a few years some 20+ operating systems offered by multiple different vendors). Except for small businesses, there were and are relatively few businesses that only possess Windows; most organizations had or have some UNIX or mainframes or older platforms like Tandem or Stratus. Message queuing works when applications on all of these can intercommunicate — which pretty much explains why MSMQ waned while MQSeries won most of market share (and skill support).

Evolving to EAI, brokers and buses

Towards the end of the 1990s middleware was established. There then arrived a semi-chaotic time when there many vendors chasing different specialties in the Enterprise Application Integration middleware space. Examples included New Era of Networks (NEON), Mercator, SeeBeyond, Crossworlds, webMethods, Sonic and many others (many of these have now been acquired by larger players).

The basic technology of middleware messaging was its simplicity. It was possible 15 minutes after you put the CD in your machine to be productive. However, what had started as being simple — enabling a small number of applications to work together — was often not architected and rapidly became a highly complex spider's web of multiple point to point connections between applications. Furthermore different vendors' middleware did not talk to each other. Vendors even promoted this — despite customer distaste — in the vainglorious hope of establishing a market dominance that could not be arrested or overthrown. All understood that overcoming the problems and inefficiencies that application silos represented was a huge business opportunity.

It did not take long thereafter for the concept of a bus or of a hub to arrive. These are concepts that are often confused. To me a bus is a backplane, where anything can communicate with anything directly; a hub is a central point of communications through which everything communicates (though that can become a single point of failure potentially). From a topology viewpoint you can argue that there is no real difference. To me the primary difference lies in the concept of order:

- **if you communicate through a hub to send a message to somebody else, all messages are always delivered in the same order in which they reached the hub**
- **on the other hand, if you use a bus and your connection between A and B fails, the connections between A and C and C and B can still be still working; while A cannot send anything to B it could be sending to C and C could be sending to B (in effect making the assumption that A has the same view of the world that B does, which is no longer true while A to B is down).**

This has some subtle but fundamentally important implications. In addition, there are some genuinely interesting (if obscure) logic traps in the whole EAI/hub/bus middleware space as a consequence of those types of difference. Too often these were insufficiently explored, sometimes with disagreeable consequences for user organizations.

That said, hubs were and are effective, even if they did introduce a single point of failure. Similarly, buses worked, so long as you architected to avoid introducing unintentional ordering issues. The result was a growth of Business Continuity/Disaster Recovery (BC/DR) approaches to produce reliable and predictable hubs and backbones and hubs and buses to eliminate single points of failure or of ordering issues.

That said, it can be argued that the interest here was about infrastructure. At this time there was little or no business logic, just better and improved middleware technology.

Top down or bottom up

Technologists usual work bottom up to provide enabling technology. Middleware was similar at its start. This, however, suddenly changed because of the difficulties encountered delivering working middleware solutions which needed to carry transactions across the enterprise. The first middleware solutions were often delivered to enable data replication when the data was not in a form that could be replicated by standard RDBMS products. The message payloads needed intelligent routing, transformation and often protocol conversion. This was essentially the EAI offering —

stateless processing of messages between applications. Middleware designers flipped. Their thinking became top down, focusing on the implementation of a business process controlling the movement of messages from application to application and enabling transactions to be tracked across the business — stateless middleware had become stateful.

Hub and bus products attracted because they offered new capabilities for connecting applications and overcoming the application silo barrier. But they also forced organizations to rethink, from the top down rather than from the technology up. This is where the attraction to business processes arose, together with all the 'objects' that are associated with each business process. From this evolved concepts like global objects, global business objects and application-specific business objects. The intention was to enable business objects to communicate with other applications so as to fulfill the business process. To me this was a breakthrough in and around middleware. It opened genuinely new doors to thinking about how businesses deployed and used IT. The downside was that much of the thinking required was complicated, and lost to most IT (and business) people.

The fax machine effect

At this point let us look at the impact of middleware from the point of view of the 'fax machine effect'. The fax machine was, and is, an important piece of middleware (even if it is not software). Think of all the times that organizations receive a fax of data from one system which is then re-entered into another system (the number of times this happens, even today, remains enormous). In effect, the fax machine with its users was the original middleware — sitting between two (or more) applications.

In addition, the fax machine represents a starting point for a particular style of execution and acceptance of middleware. A significant factor in the adoption of the fax was, I have been told, due to Wal-Mart which mandated that if you wanted to do business with Wal-Mart you had to communicate by fax. This swiftly accelerated the take up and acceptance of fax machines. After the fax machine, increased automation has come to be applied. Today, the need for all Wal-Mart suppliers is to be able to work with AS2, an Internet enabled implementation of EDI. If you cannot use AS2 you cannot supply Wal-Mart.

In effect, the introduction of much middleware is being driven by a small number of large businesses deciding what suits their business processes, and then requiring suppliers and partners to adopt this. As those suppliers and partners make the changes to satisfy the likes of Wal-Mart (but this could be General Motors or Ford or Boeing or Unilever or …), then that technology trickles into the infrastructure of suppliers and partners — and is re-used. In effect a few decisions (made by a relatively small number of big organizations) to adopt open protocols at the technology exchange level (to satisfy their business process requirements) is driving middleware changes in customer, supplier and partner organizations.

Once a middleware standard becomes adopted for communications between organizations I think this has a significant impact on proprietary providers, particularly as AS2 is a standard which enables different products to communicate. It is not necessary to have the same product at both ends. This approach is the primary benefit of the Web standards approach even though intercommunication between individual products continues to be a challenge.

The past 5 years

The collapse of the dot.com boom forced many changes, not least in the ways that organizations spent on IT. In Strategic Thought's case, this was good for business as organizations focused on using their existing assets by building middleware interfaces to exploit the capabilities they provided, rather than on new developments. We saw many middleware projects start to take off post-Y2K, and we have been busy at it ever since. I think the real issue about market consolidation, after the dot.com bubble, was that the likes of IBM and Microsoft realized how important middleware was and produced heavyweight products which captured a significant percentage of the market when uncertainty — some originating from the dot.com crash itself — occurred. There was the feeling that 'if our business is so dependent on this middleware, we will buy from the heavyweights (like IBM, Oracle and Microsoft) because we know we can rely on them to continue to exist.

Initially these interfaces were ad-hoc but increasingly they are being build to deliver Web Services interfaces, particularly as the Web Services standards evolve to provide security, quality of service, instrumentation and other capabilities required for business. This approach also promises much for the implementation of components breaking down the application silos that EAI seemed more to cement in place rather than supplant.

Where did the move to Web Services come from? This is an interesting question, for in many ways their arrival was both unexpected and unprecedented — given what had occurred in the previous 5 years of dot.com feasting.

In my personal view, Web Services are one logical outcome from the frustration caused by there being too many middleware vendors. Customers were increasingly frustrated that all this middleware — that was to enable integration — did as much to prevent it as to deliver.

Whatever the origin of Web Services — and there have been many contributors and players, from Sun with Java to the Global Grid Forum to IBM to the open source community to the W3C and OASIS — Web Services represent the start of the IT industry (users and vendors) joining together and meaning 'we need a set of standards'. Committees were formed, which performed real work. From these committees, Web Services concepts, definitions and then implementations started to arrive.

The key point was, and is, that if you have standards, then people, processes and applications can communicate. This is the whole point to integration. Having made that point, it is also true that within the Web Services world (thus far) there has been painfully little that exploits asynchronicity. In one sense Web Services have taken huge strides forward. But in reverting to 'what everybody knew' (synchronous processing), the advances have not been as great as most of us would wish.

Yes, Web Service thinking exists about buffer services, cacheing and various alternatives which might mitigate synchronicity. But these do not add up to fully fledged asynchronicity. This is something to be regretted.

A point also on the Web Services standards. They seem quite complex and I do not pretend that they are my bed time reading. I am generally skeptical of committees and we often refer to the camel as the race horse designed by a committee. However, while the horse runs faster in good conditions, in the desert when the going gets tough the horse dies and the camel survives. So, I think, it is with the work of these standards committees.

In 2007 and 2008

Superficially, and looking from the outside, the middleware arena may appear to have gone quiet — at least by comparison to the froth and noise of the late 1990s. That is not, however, how I see it.

There really is a huge amount going on around the middleware space. The big difference is that this is not so much about technology. Instead the focus is on the business and its processes, architecture, the interfaces between components in the architecture and the mechanisms to enable components to communicate across those interfaces.

The really important aspect of having middleware is that it enables us to build applications which can span the enterprise and therefore are far more complex than a single departmental application.

Thus the problem space (that middleware has enabled us to start to address) is becoming larger and larger and larger. At Strategic Thought one of the biggest challenges stems from this. We see our clients simply not possessing the people with the methodologies to build enterprise wide solutions. (Think how large a task it is to roll out SAP across an enterprise, let alone design a new solution of this scale.) This is particularly apparent when whole enterprise processes are involved as no one person in the organization has a full grasp of the process.

I have been asked to attend many conferences around how to deliver. Typically there are 300-600 people in the audience. The discussions are wide ranging but are generally inconclusive. Everybody has a vested interest. All think their point is really important, but significantly, most fail to understand the interests of their colleagues. It is impossible to get everyone to have a common view of the problem because it is too large. Rather, these problems need to be decomposed into manageable chunks. Once decomposed into manageable portions, one can address each problem, in turn, in a series of steps.

We see what we often call the adoption-curve challenge. This is a concept issue. The people who are coming to use middleware do not know much about it, or how to exploit it. We have found too many projects running into real problems in the early stages, not with making the middleware work but with understanding the underlying concept of business processes, how you map interests within and beyond a business.

Let me offer an (anonymous) example. One of our clients, a large bank, had created a middleware team to develop a specific solution. That team, apparently, produced nothing for a year. The rest of the bank organization started to say 'you guys have spent $3M-4M, and delivered nothing'. Yet, this team had been working extremely hard and it was producing. Much of what was being created was conceptual, laying out the ground rules for how future implementations should occur. But it (the team) had made a vital mistake: it had not focused on obtaining and delivering an early win to give the rest of the bank the confidence that their activities were worthwhile and justifiable. Strategic Thought was able to assess and report on the good work, then assist in focusing on the deliverables — with the result that the project was successfully delivered and met its substantial ROI criteria early.

Skills and understanding

From another perspective, some time ago Strategic Thought was retained by a medium-sized insurance company. We tried to take this insurance company's people conceptually from ground zero (where they were then) to where they wanted to be — in two weeks of discussion and presentations. Not to put too fine a point on it, this insurance company's people did not 'get it'. Our work was terminated.

About 18 months later I met up with that organization again. It had taken them that long to come to grips with the concepts I had been trying to introduce to them in that two-week period. In effect we had overwhelmed them. Theory is no substitute for practice. They were not capable of accelerating from 0-60 in 2 weeks; they needed the full 18 months. Put another way, 'you cannot have a baby in one month if now is when you become pregnant'. Certain things take time.

There is, therefore, a real adoption-curve challenge about how organizations take on new Web Services technologies and see through the design and then implementation. This is particularly true in end user organizations where loyalty and dedication to support a system over many years is highly valued and creates a resistance to change and fear of the unknown, by comparison with the services industry where a person's fee rate is often determined by his or her knowledge of the latest in demand technology.

The logical arrival of the ESB

In reality what we really have is a requirement for excellent architecture. This must be able to decompose any given whole problem space into smaller sets of problem spaces — where each one is each big enough, well enough defined, appropriately sized and with the necessary interfaces (human and system) — that an organization can reasonably expect a team to deliver.

In effect what I am saying is that someone has to commit to spending money on an architecture team. That architecture team then has to sit between the business users and the developers. It must work very carefully and closely with the business users, and equally closely with the developers. It must create projects with budgets which are deliverable and ensure that developments conform to the architecture. The environment in which architects must now work is not the same as it was ten years ago.

Indeed, I will go further. I find that many people consider middleware solutions as just another application. This influences their thinking: they seem to expect to throw US$200K, US$500K, US$1M or whatever, and have 'it' resolved. Middleware should not be approached like this. The difficulty is that if you do approach it like this you almost always ensure that the outcome will be yet another silo or stovepipe — which is exactly what middleware and a suitable architecture are trying to overcome. There is no doubt that this is a challenge, but it is a necessary one.

The good news is that I have seen the architecture model work. One large telephone company for whom we worked had an excellent architecture team in place. But even that does not always ensure success. In this telephone company, the challenge was that the business would say to the architecture team 'we want this'. The architecture team would say 'well, to do that properly is going to take 10 months and cost US$3M'. Too often the business people would then go (behind the back of the architecture team) directly to the developers who would respond: 'No problem; we can knock that up in a couple of months for US$400K'. Despite the excellence of the architecture team the result would be yet another stovepipe which was either incapable of subsequent integration into the business process or would cost even more to integrate than the original estimate. (This has shades of EAI, as discussed earlier.)

What I am really talking about here is the evolution of the Enterprise Service Bus concept and the implementation of business processes around this. This was what Crossworlds was all about — automating business processes across a bus, hooking in user interfaces and invoking those underlying services when the business process needed those underlying services. The change here is that the orientation was business-process-oriented.

Unfortunately this is not easy for people to grasp if they have not thought in this way for a while. In turn it makes the architecting and planning of solutions that much harder. The essence lies in the concepts and if you or I do not comprehend the concept, we are not talking the same language.

Yet this is an area of extreme productivity. In my view, owning Enterprise Service Bus middleware competence is going to differentiate service providers.

Recreate the architect

Having said all this, I often ask myself how many organizations are making the type of investments that I am talking about? When I am honest with myself, it is only a few. This is a challenge, not least because embracing and following through with an architectural approach requires discipline and is difficult. Even today it tends to be software vendors

who come in saying 'our SOA — or ESB or whatever — products can do all of this for you' while they try to find a 'low point for entry'.

What I am not seeing is demand being driven by businesses asking 'how do we deliver an architecture plus obtain a better return on investment?' Businesses are demanding solutions driven by the need for automated work flows but relatively few are putting in the architectural investment, because they do not understand what is required to engineer that architecture. People say the words.

To most people, an architecture is just a technical architecture or even just a list of products. It is too similar to a 'strategy' which is 'Oracle' or 'IBM'. What does this mean? Yet the reality is that this is what too many people are thinking.

There are, therefore, some real 'connection gaps'. Part of the problem is that everybody thinks that because we all wear suits, everybody who wears a suit must be a good IT person. The range of quality of expertise in the IT sector is highly variable with a level of engineering capability that remains relatively low. The fact is that implementation specifications are weak. This is associated with complexity and the inability to break down a big design into smaller, manageable elements. In my personal view, we need a new job description of 'Business Architect'. Too often today, the term 'IT architect' is used when all it really means is 'designer'.

Lessons learned

My first lesson learned is that the issue of the adoption curve is one that is critical at the moment. It takes time to do deliver complexity. It takes time for people even to understand the underlying concepts: what is the problem you are trying to address? Yes, there are different ways in which you can achieve this, depending on the caliber of the people you have. But what is utterly important is to find a way, in any implementation, in which you can bring together ideas at many different levels, from many different perspectives, about what you are trying to achieve.

Almost as important is the need to create small projects which enable the stakeholders to participate and understand why certain decisions and actions are being taken. Just as Rome was not built in a day, so people must have the time to evolve and learn. Technology solutions are capable of far more than people are able to grasp or organizations are able to implement. We must address the many conceptual gaps that persist.

Another area where experience says we need change is that too many people are still thinking narrowly about the delivery of functionality. Organizations do not seem to want to invest in 'nonfunctional enhancements', except in extremis (like for security when it is too late or redundancy and fail over after the solution crashed and was unavailable or upgrading to the current release after having been hit by triple support costs for 'out of support' products).

So, for me, the core issue today is about how to:

- **break problems up into manageable chunks which can be understood**
- **bound these chunks with interfaces which can be the limit of understanding for everyone else**
- **work out which of the standards a designer or an architect needs to implement**
- **place the focus on implementation in short phases to enable flexibility and the ability to meet the business's most urgent needs.**

The technology part is easy (relatively). The real challenge lies in the design approach. Business architects are a necessity — and all organizations are short of these.

Management conclusion

From Mr. Denning's perspective the real issue about middleware is that it should enable the decoupling of applications and thereby the satisfaction of the business processes that are the heart of any organization's ability to function. When you cannot change either ends — perhaps because the client end and the server end are in different organizations — there is a constant and real risk that middleware is inhibiting business flexibility. This is no longer acceptable to most organizations.

In addition, he makes the point that the role of the IT architect must change. This role must take on a greater 'business process' dimension and sit between business users and developers. Such architects should also be the custodian's of observance of the enterprise architecture.

Middleware may seem to have disappeared from the radar screen that covers publicity. Nevertheless, as Mr. Denning shows, middleware is alive and very much kicking, albeit addressing ever larger problem spaces.