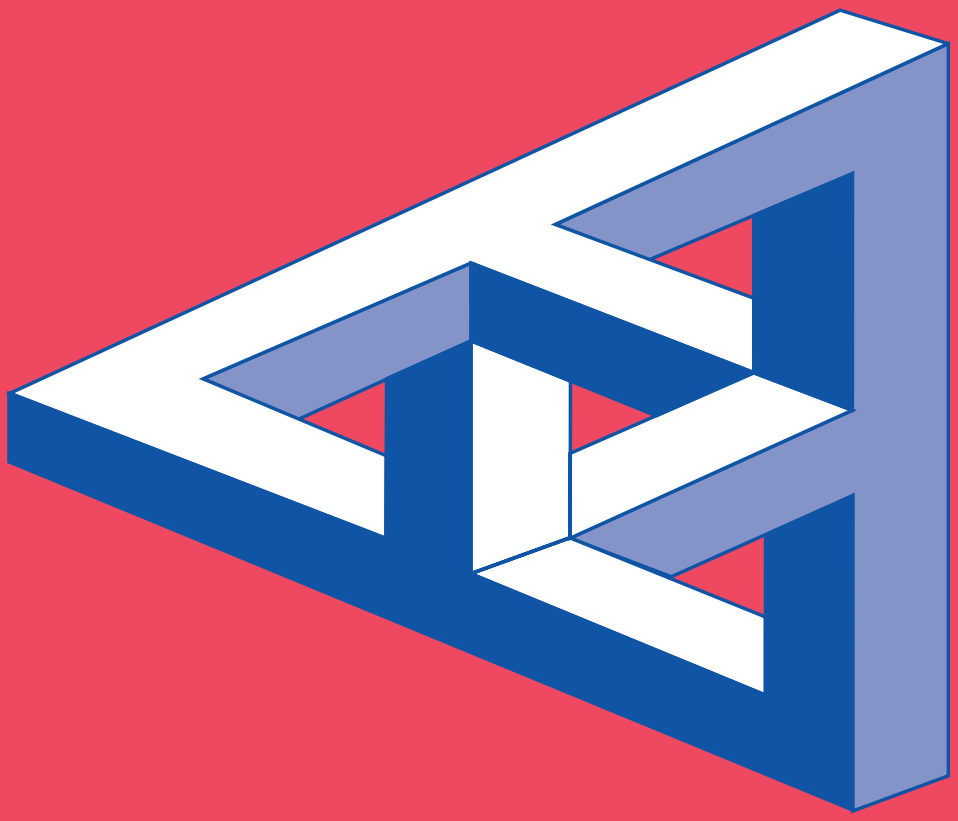
MIDDLEWARESPECTRA 02/97



Volume 11 Report 1

Contents

Middleware and networked computing

Bill Donner
Chief Architect
Reuters Limited

Management introduction

Bill Donner is the Chief Architect for Reuters Limited, the global information, communications and services company based in London's Fleet Street and with offices worldwide. To support the many Reuters services, it operates as many different and disparate organizations, with interesting implications for both middleware and networked computing. In this interview Mr. Donner describes the challenges which Reuters faces — with over 50,000 servers and 375,000 workstations within the Company alone.

Although Reuters may be regarded by some as an information distribution monolith, what he describes is applicable to most sizes of business — because Reuters runs as both a collection of small and medium enterprises as well as a global organization. His thoughts, therefore, about the challenges facing him as Chief Architect address many of the issues which are found at all levels of middleware and networked computing.

He comes well equipped for the role of commentator, having run Reuters' transaction systems in the US before moving to London. Prior to joining Reuters he worked for a subsidiary of Security Pacific Bank (now part of Bank of America) building a money market trading system. As is well known, financial trading systems are amongst the most complex of advanced systems and with an expectation that dependable middleware and networking are available, often globally.

Magnitudes

Let me start by offering you a feel for the scale and range of our development organizations. At the last count, we had some 29. As Chief Architect my role — through a variety of committees and other methods of oversight and influence — is to try to:

- **bring some cohesive technical message to all these groups**
- **offer an external technical perspective (to the individual groups)**
- **promote convergence, where possible — while still encouraging diversity and creativity**
- **stimulate innovation**
- **persuade everybody to march to roughly the same tune**
- **offer investment protection.**

As you can imagine, I am sure, this is at times extraordinarily difficult given the number of differing development teams and organizations which we have in Reuters. Compounding the challenges is one of our strengths — our global geographical spread.

In addition, you have to understand that Reuters is both:

- **a consumer of middleware and networking**
- **a producer of middleware and networking.**

For example we have traditionally built much of our own networking and middleware over the years. We have bought software, whether on mainframes or other systems — including products from vendors as diverse as IBM, NCR, Digital, Oracle and many, many others.

We also sell middleware and connectivity products in their own right. TIBCO (which used to be known as Teknekron) produces the TIB middleware product for connecting publish and subscribe applications — which is particularly useful in information dissemination. Indeed TIBCO is one of the largest middleware vendors — if measured by sales or profits — around today (see also the August, 1996 **MIDDLEWARESPECTRA**, Report, page 8).

How did we reach this position of being both consumer and producer? It is simple. Reuters operates as a series of businesses. Most of these — whether delivering trading systems or information systems or disseminating news and editorial or whatever — operate in a fast, fluid and relentlessly dynamic environment. Each has its corporately set targets — but the task of each 'business' is to deliver against its own objectives (which, in aggregate, have created the Reuters' decades of success).

Add to this that we have made acquisitions over the years — many of which bring in new technology of some form — and you will understand why we have such a challenge when trying to manage middleware and networking in a coherent manner. Indeed, it seems almost that every time we think we are moving to a slightly more consistent approach a new variant or initiative is waiting for us.

I do not think, however, that we are materially different from most enterprises. Most have similar problems, except perhaps not on such a global scale.

Partial inventory of Reuters middleware

To give some idea of the variety — and complexity — of our middleware combinations, let me describe briefly a few of our systems. If you look at our transaction products, these have been very successful selling a number of different services to our customers. For example, we have a conversational dealing service where you and I can enter into a two-way negotiation about trading foreign exchange (FX).

This has its own middleware. It has its own infrastructure. It is not IP-based (Internet Protocol) and dates back quite some time. Nevertheless there are formal APIs and there is formal software on which you layer the application protocols for these conversations. We 'only' have 25,000 or so FX traders who use this system. Today we still actively build additional applications using these APIs on top of this middleware — for particular markets.

If this is one family of middleware in our transaction protocols, going around the globe we offer matching systems. This is where you and I trade but it is done anonymously through a broker — with Reuters essentially functioning as an electronic broker. Globex and Dealing 2000-2 use the same basic architecture on a DECnet base (which we will be migrating to IP). Over the years we have enhanced it so much (adding functions and capabilities) that it is no longer a standard DECnet; we did this because standard DECnet had its deficiencies in the Reuters context. Today we have a whole hierarchy of repeater boxes and higher level protocols which we build on top to:

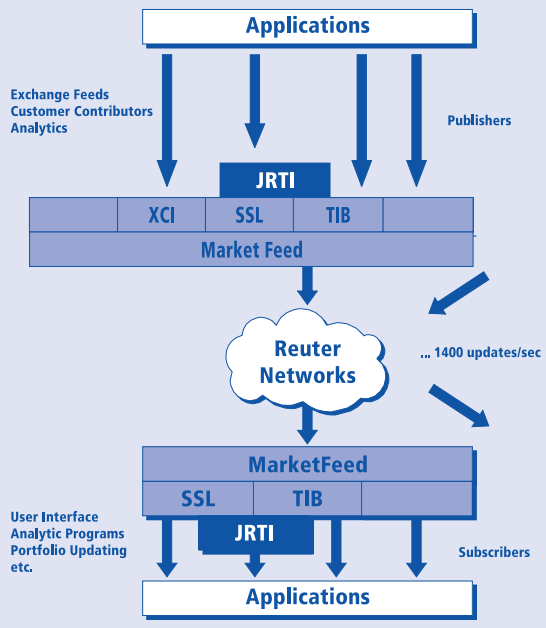
- make it more reliable
- introduce sequencing and error retransmission
- handle cases of broken connections
- etc.

Indeed there is a distinct middleware culture around both of these transactional systems — at customers as well as within Reuters. Whole applications are built on what originally was a limited capability but which is now 'taken for granted' — proven middleware.

Instinet is different again. It is our off-floor trading exchange where people can trade out of exchange hours on NASDAQ. It uses a completely different technology for distributing information and, for trading, a diverse collection of elderly DEC equipment with a set of protocols we developed called Etherspine (a broadcast Ethernet technology with layered protocols on top). We are now upgrading this to UNIX headends and NT distributed servers.

Besides these most obvious of our middleware creations, there are many additional different middleware efforts. However, it would also be fair to say that today the focus is more on application development (using this previously created middleware) as opposed to creating new middleware. After all, we have made our middleware, and we have made it work.

Figure 1.1 Many middleware alternatives



Middleware and networking go together

Our fundamental core business is delivering data from a variety of sources to a variety of end points — our customers. With this view of Reuters it does not take long to understand that we are in the middle. We have:

- data feeds coming in (news, trading statistics, trades, etc.)
- analysis and transformation of data into information (sometimes as 'simple' as co-ordinating or collecting a specific form of data for onward delivery to customers; at other times vastly more complex)
- delivery, via different data feeds, to the customers (to hard wired screens, into customer systems, etc.).

Our value add is that we:

- combine data
- perform analytics on data
- massage data
- vet the data for accuracy
- weed out bad data points
- deliver the result of any or all of the above to our customers.

This happens with us, Reuters, in the middle (Figure 1.1). To deliver what customers are seeking, we are continually looking at introducing higher level protocols which customers can use to:

- exploit what we feed them
- create their own applications using our data to address new or alternative opportunities in their businesses.

A good example — in the traditional data and information delivery of quotes and prices and the like — is SSL (Source Sync. Library). This has become — at least for Reuters' customers — a de facto standard for enabling them to take data out of our delivery mechanisms into their applications. As customers have no standard platform we make SSL available on many platforms. It is arguable today that SSL is an accepted, open, albeit specialty financial, middleware standard.

That said, most of our applications sit above still more layers of middleware which sit on top of SSL. For those who do not like a standard API set, we now have SSL Foundation Classes (SFC) which are object oriented and support C++. We have a number of efforts underway which will produce similar Java classes for

SSL. Later this year we will take the Java dimension a stage further — with our Java Real Time Interface or JRTI (pronounced 'jerty').

Increasingly our primary programming is moving to a Java model. It would not surprise me if we do all our mainstream user interface programming in Java — because of its multiple hosting capabilities (using the Java Virtual Machine presence on everything from Windows to MVS, as and when we need it). We have always been unhappy with C++, yet I am always 'impressed' with the agony that a small group of people from Bell Labs. were able to fob off on the entire IS world over the past ten years.

This matters because we want to avoid a repetition. We will put our own 'layer' on top of DCOM or RMI or CORBA — so that we can be independent of either. That is where JRTI comes in to play. It will be the Java Real Time Interface for getting quote data out to the UI. Whether JRTI goes over DCOM or RMI to talk to the server will be pretty much moot to our applications people. Our challenge is providing — but hiding — the middleware so that we can concentrate on building the information applications and delivery which is what our customers buy from Reuters.

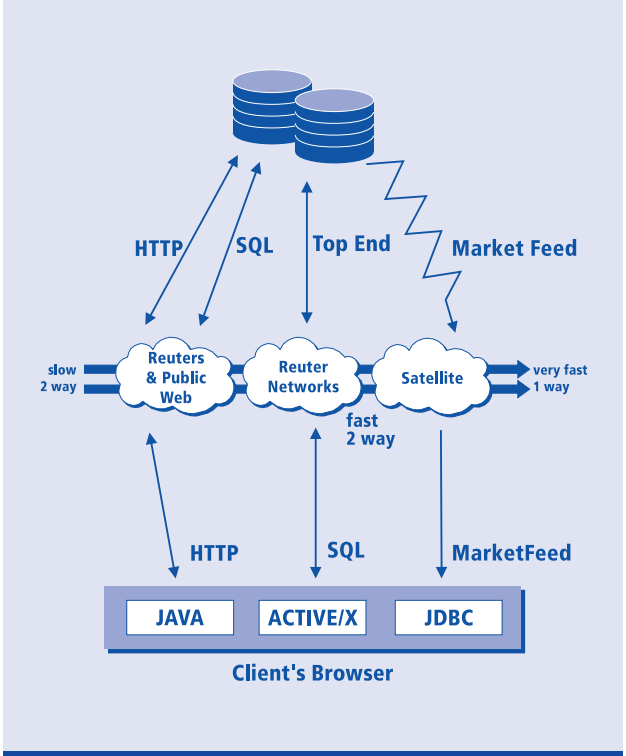


Figure 1.2: Client browser integrates Web, DB, ...

These have become, or are becoming, middleware standards for ourselves as well as our customers — because it is how our:

- **customers get to our data**
- **terminal programs get to our data**
- **own workstations get to the data.**

If this is one side of the house, we also have the TIBCO-inspired protocols. At a very low level, there is the TIB which is a multi-cast, reliable, IP-based solution for distributing data over a local area network and, more recently, over a wide area network. On top of the TIB come a variety of other 'middleware functions' — from queuing to distributed transaction processing.

TIBCO also offers a TIB variant — called the Rendezvous Software Bus — which is a stripped down, low cost TIB without most of the specialty financial interfaces. Intuit incorporates this in all its latest editions of its PC-based Quicken product — in order to provide the interworking with banks, credit card companies, stock brokers and the like which makes Quicken so attractive.

My point here is that networking and middleware go hand in hand. They are inseparable. If you did not have the distributed dimension (connected by networking), you would not need middleware (certainly not as much as distributed systems require). Middleware sits above networking (or includes it). Whatever — the two are inseparable as the underpinnings for distributed computing.

The systems management dimension

As we push ever more complex combinations of networking, middleware and applications out into the market place (and create all kinds of new solutions), we are faced with a systems management dimension of enormous proportions. We have over 50,000 servers and 375,000 workstations in the Company (and this excludes any element of internal or external customer systems). We need to be able to manage these day to day as well as occasional upgrade activities.

We are actively pursuing the acquisition of a large, comprehensive systems management environment with which to instrument and control all our servers and desktops worldwide. As we migrate into the future, we know that everyone of these boxes must come loaded with the required systems management agents to allow proper management — from day one of installation.

Is this middleware? I would argue 'Yes'. It is in the middle, involves networking and will play a key role in enabling us to run our business. But let me tell you of

an uncomfortable development (at least for vendors) which we routinely apply to all forms of middleware, indeed to all forms of system.

The system management environment which we select will not be bought from the vendor with the most elegant systems architecture or most function. It will be bought because it offers the best overall deal. But it will not be the vendor who offers the best architectural principles who wins. At the end of the day it is value effectiveness (including price) which sells into Reuters — assuming that something is not outrageously deficient in some respect.

This is a real change from a decade ago. It is also the way we sell our networked middleware, because it is the way our customers buy from us.

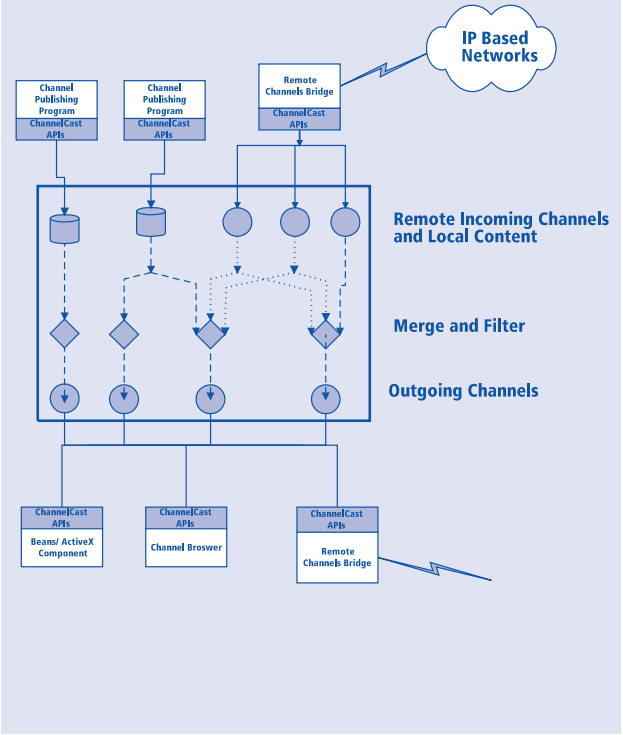
Technological excellence is presumed. It is no longer the defining purchasing criterion.

The impact of the Internet

Reuters began looking at Internet possibilities some two plus years ago. For me, as Chief Architect, the Internet is both a blessing and a menace.

The reason that it is a menace is that it is so fast moving; staying ahead is a problem. On the other hand, nearly all parts of Reuters are expecting the Internet to have an impact on their business — whether as:

Figure 1.3: TIBnet — reliable IP multi-cast



- **a competitive threat**
- **alternative or additional delivery channels**
- **new audiences and customers**
- **combinations of these.**

We are building our public Internet presence. We are also building an internal net for our customers — where you will have to be a real Reuters' customer in order to have access (Figure 1.2). Interestingly — from the middleware and networking viewpoint — the technologies for each of these various initiatives are, in most cases, the same. Therefore many of the technical issues are similar.

The result is a great deal of interaction between our various subsidiaries and development groups. With the importance of TCP/IP to the Internet being so clear, one unexpected result is that the Internet is encouraging the adoption of some common technologies.

This is a case of practical results forcing convergence, moving the Company forward. However, as I will describe, the Internet is not a short cut to standardization (with the possible exception of IP, the Internet Protocol itself).

But as far as everything being Web based is concerned, I think that we have got to look at the Web as a tool, as a vehicle. It is something that we feel we will be able to use to help management control and distribute information — although this will be in parallel with our existing distribution network. I know that we will have satellite distribution and that we will be moving to the merging of real time, fast moving financial information with more traditional static Web type pages. We can already see how the Web will improve delivery to the customer — even though the result is, in fact, a merger of Web and traditional technologies.

Over time our hope is that the Web will drive the shape of desktop/client-side integration — running under Java/Java Virtual Machines and/or ActiveX. That is why we are doing so much development with both of these. We think that users will push us this way. We hope they do because having them tell us what they want 'as middleware' is much easier than us trying to impose a 'Reuters view'.

Evidence of our moves in this direction can be seen with our December announcement that we will re-do information distribution using TIBnet (the combination of TIBCO's TIB, CISCO routers and various other middleware — see Figures 1.3 and 1.4). Using TIBnet's reliable IP multi-cast based distribution scheme (with channel orientation) we will be able to propagate news, quotes, business briefings, etc. over our IP net-

work. With browser simplicity enhanced by Java/ActiveX function, we can deliver to a UI (the browser) that is being adopted in the marketplace. Indeed I do not think it will be long before we are going to be asked for (or offer) personalized content delivery:

- **one to one delivery**
- **group oriented delivery for corporations**
- **channel oriented delivery — for communities of interest.**

Delivery is also changing. Today we have both satellite and terrestrial delivery, to different customer segments. Both of these will move to a unified satellite terrestrial IP basis. Once that is accomplished — the lowest level of middleware — a unified IP scheme will allow us to connect irrespective of whether you receive via terrestrial lines, satellite, cable or whatever. What will matter is that the software interface to all these will be the same.

The key point here is that without consistent middleware in place it is extremely expensive for us to support what we have. We write special purpose software to achieve everything. With a consistent IP-base at the foot of the middleware stack and Web-browser presentation acceptance, we can fill in the middleware in between — in whatever ways customers and products pull us, without having to impose our view on others.

On another note, we are extremely interested in cryptoboxes and similar technologies. These allow you to encrypt information in a container and to charge, account and collect for it on a per access basis.

I am convinced that this will apply to Reuters. After all, our business is about supplying information and delivering it. If we can make it available in new ways such as over the Internet — with a minimum of overhead and complexity — we can open up new markets for that information.

Similarly SET technology and similar electronic commerce over the Internet will become increasingly important to our business in Reuters:

- **they (SET, cryptoboxes, etc.) are other forms of middleware, ways of capturing secure transactions and shifting them around**
- **once established we will not have to worry about how many times information is seen and yet we can be assured of payment every time our information is seen.**

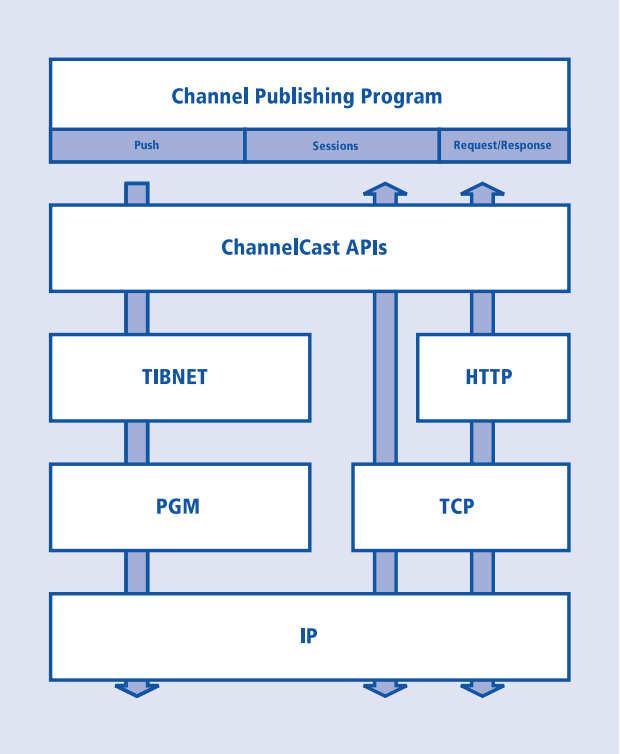


Figure 1.4: TIBnet — explored further

The shape three years out

In my view, in three years, we will be Internet standards oriented to at least some greater or lesser extent. At its simplest, or lowest level, there will a steady increase in the introduction of IP into all our networked computing and delivery. We will not have completed a 100% conversion (not all customers will want to move that quickly) but the process of change will be well under way.

In addition we will see Web-oriented material everywhere, irrespective of whether it is for:

- **public or general consumption**
- **customer information (that you subscribe to receive it in some form)**
- **internal purposes.**

We will have vastly expanded our databases. Already we are taking existing data — accumulated over the years — which we are putting into a form like a warehouse which is available to customers (rather than being just an internal data warehouse). Currently customers access this:

- **through a Reuters determined protocol**
- **via our networks**

- **using the workstation and user interface software we provide (although this may change, over time, as we are able to use Internet-based individual charging systems — for example IBM's cryptolopes).**

With information on FX, equities, financial instruments, back news, etc. we can provide historical access globally and a whole series of products are being constructed and released around this. Confirming that we do buy middleware, we are using Top End — as a message processor rather than for distributed transaction processing — to enable this. Top End has the additional advantage that it allows us potential access to CORBA — another form of middleware which I expect to see adopted by at least some of our development organizations over the next three years.

Beyond these areas, we will have centrally manageable systems. These will be operated from two global centers — one in London and one in Singapore. From these we will watch every server in our world, via networking and middleware. It will be these that keep our systems glued together and commercially coherent.

For the customer this is absolutely the right approach. When we sell our subscription based services, customers buy our information as well as processing services — not middleware. We give them what software they need in order to accomplish their objectives.

In effect the customer pays us as a utility — to hide middleware/networking and make it work for them. This puts real pressure on us to understand what we are doing.

Managing for consistent middleware

My objective as Chief Architect is to try to steer new developments in a consistent fashion, to try to plan the orderly migration of older systems into a convergent future, a future being based around a limited number of standards and choices.

We have already committed ourselves in certain areas to a number of key technologies, for example to:

- **IP**
- **Windows NT, as the basic standard operating system for desktops as well as for the new communications components which we will be constructing in future years**
- **consolidation of programming languages, around C/C++/Java.**

Besides all this we will still have UNIX (traditionally used in the trading rooms but which I see declining over time). We will have specialty systems — from large databases on UNIX to IBM mainframes supporting business functions. We will keep these so long as the commercial situation requires — because Reuters is an organization that succeeds by delivering what is needed by its businesses rather than what is technologically or politically correct.

Our pragmatism recognizes that old systems do not just fade away. We still rely on them. Our customers still rely on them.

We have many legacy systems (including middleware and networking protocols) which are vital to both our present and future business. In some cases we are having difficulty recruiting individuals with the necessary skills to develop and maintain such systems. That is why we are focusing on C, C++ and Java as our core languages for development.

Introducing rationality

Does this mean that we will manage to introduce a single coherent set of middleware and networking protocols? I do not think so. I do not even think we want to do so now.

The consensus view of senior management is increasingly that it is not feasible to expect a wholly integrated desktop/client side architecture — at least at this particular point in time. We are all too well aware that Reuters has a great deal invested in multiple types of middleware. The Holy Grail of a unified client-side architecture is several — at least five — years away. To try to demand that our many businesses accept a single replacement does not make sense; it could slow down everything else we are achieving.

Two years ago my view is that we thought that there would be some point around which most of our technologies would naturally converge — perhaps around Internet-related technologies. Two years on we continue to buy in enough diversity to make that convergence difficult. That the sheer scale and complexity of our existing systems may be too difficult to change is just another inhibitor.

Add to this the sheer pace of technological change — from COBOL to Pascal to ADA to VB to Java and ActiveX in the languages field alone — and, to me, this diversity only goes to prove that a single choice or approach is only possible with an absolute hand guiding developments. Unfortunately, such absolutism usually degrades into a lack of originality and vitality — which would negatively impact our ability to react to

change. At Reuters we are curiously — and simultaneously — conservative, dynamic and adaptable.

It takes time for new technologies to go fully operational and to be absorbed into our corporate culture. Partly this is because they acquire credibility once they are in production (we particularly dislike vendors who withdraw products before we have even had time to deploy them — as has happened on more than one occasion). However, when we do adopt them, we are aggressive in how far we push them out into the world.

I am confident that we will we have a loose but flexible infrastructure three years from now (for example across systems management). I am equally confident that we will also have a broad collection of middleware, from TIBs to Top End to residual home grown as well as various new forms middleware and networking, supporting our transactional and other businesses.

Lessons learned

Number one, with so many different groups inside Reuters and with a culture of autonomy, what I have learned is that even thinking of introducing an absolutist approach is probably futile:

- **it can destroy innovation**
- **it is largely counter to the Reuters' individual enterprise-based culture**
- **it is too easy today for somebody to bring in something alien or foreign and successfully make use of it, irrespective of any diktats that might be issued from London or elsewhere**
- **it presumes that any given technology is best for a purpose at all times — an assumption which we have continually and repeatedly disproved in practice.**

That is not to say that the pressures for us to standardize are not great, financial ones being the most obvious. But what I have learned is that the trade-off between technological perfection and anarchy is heavily weighted to the latter: partly because change is omni-present and partly because perfection is astonishingly expensive.

We can wake up here in London one day and find out that we now have a growing chunk of business dependent upon some XYZ middleware product from some obscure company in Sydney or Hong Kong which may not be in business in 18 or 24 months from now but is significantly assisting Reuters' bottom line today. Faced with a choice of customers who generate revenues today and selection of the 'right' technologies for the future, our businesses will always prefer and side with the former.

Aligned with this is my second lesson learned. This is that picking winners for the future is near impossible. There are just too many competing solutions coming out for different problems (and combinations of problems). You only need look at the Internet to prove this.

In my view it is almost impossibly hard to navigate through who will be the winners, or losers. For example, we have — at the last count — nine different search engines being employed in various parts of the business. Which is best or most appropriate? We do not know yet. Only time will tell. In effect we are letting market pressures take the decision for us. This seems to be, at least for us, effective and allows our mix of vitality and creativity to produce continuing commercial success.

Management conclusion

Reuters is an organization with, probably, as much middleware and networking orientation as it is possible to find today. Not only has it been actively participating in the Internet world for the past three years, but it has a long and proven history of delivering networked services using purchased as well as previous generations of home grown middleware operating in a highly distributed business model (in all senses).

This places Mr. Donner in an exceptional position to comment on the many dimensions and challenges of managing middleware and networked computing, in what some may see as a contradictory environment. His conclusion — that trying to force convergence with a heavy hand requires an authoritarian and centralized approach which in turn is impractical — may disappoint many in IT who might have preferred to see consistency above all else. His reasoning, however, on why it is better to shape and influence, is convincing — for at least two reasons:

- ***its inherent pragmatism***
- ***its underlying acceptance that market pressures (whether within Reuters alone or including outside ones) are a quicker deciding factor than internal debate or direction.***

What he illustrates is that finite middleware and networking choices do not have to be made. Reuters has proved this with its successes over the past century and, more importantly for IT, during the last decade.

Putting middleware into perspective

Aubrey Chernick
Chairman
Candle Corporation

Management introduction

In the second half of 1996, Candle Corporation announced that it was entering what it called Solutions for Networked Applications (what others often encompass in the term middleware). Not only did it hire significant executives from other middleware user and vendor organizations but it announced a number of acquisitions (PowerQ, Amsys, etc.) and relationships (Level8 for the MQSeries-to-Falcon bridge) to accelerate its entry.

Middleware, thus far, has not been a market place noted for riotous success. Yet its increasing importance is demonstrated by the ever rising tide of attention from application vendors, systems vendors as well as users. In this interview, Mr. Aubrey Chernick — the Chairman and Founder of Candle Corporation, a company long noted for its systems management capabilities — discusses:

- ***how he sees the evolution of middleware***
- ***what characteristics it must possess to be successful (for users as well as for vendors)***
- ***why Candle decided to become a player.***

What is middleware?

Middleware is a great invention. Used appropriately it is what separates out applications from individual operating systems. In the multi-platform, heterogeneous world which we have to work in today, this is extraordinarily valuable.

What is so interesting to us, in Candle, is that middleware is what gives portability across — and interoperability between — applications. To us, the freedom that this offers should allow developers to utilize middleware to enable them to focus at higher application levels. In this way, they can leverage their:

- **skills**
- **knowledge**
- **time**
- **productivity.**

Their parent organizations then benefit from all of these — plus the inherent capability of middleware to enhance portability. As such middleware becomes the key to unlock a door which has, for so long, been inhibiting the ability of different applications to work with each other across different platforms and environments.

Middleware everywhere

Middleware is all around us. This is a fact that few appreciate except that, in one sense, good middleware is what you should not need to know about — if it is working properly.

You do not believe me. Consider what most organizations have installed in some form or other:

- **databases, and even file management, are middleware (sitting, as they do, above the bits and bytes of low level storage structures); VSAM is a form of middleware and when you move up to databases and relational DBMSs, all you are obtaining are bigger forms of middleware**
- **similarly, most TP monitors (CICS, Tuxedo, IMS/TM, Top End, Microsoft's forthcoming Viper) are middleware; they sit between the user and the data as 'transaction management middleware'**
- **messaging and/or RPCs are middleware: messaging is inherent in IMS just as RPCs (derived from the DCE model) are the backbone of DCOM**
- **data access (as in Sybase's Open Client/Open Server or Oracle's SQL*Net) are about connecting clients to data servers and services — still more middleware**
- **etc.**

My point is that much 'narrow' middleware is already in use, albeit in narrowly focused ways. It is being used to leverage individual application productivity and, to some degree, corporate knowledge. In our view it is quite possible to capture data definitions in middleware — say in an RDBMS or MIB — where you possess:

- **not only the development activities**
- **but also ways of defining how your organization operates (the sequences of CICS transactions or the flow of database accesses, etc.).**

Narrow middleware, therefore, already goes much further than most imagine. It is both an insulation layer (between applications and operating systems and communications) and represents a knowledge layer — encapsulating standard definitions and operations. Using these you can model transactions and interactions in order to understand how the various components work with each other across each system.

In this sense middleware assists beyond pure application productivity. It captures knowledge. Given this, it does not seem much of a leap of imagination to expect to find 'broad' middleware across multiple platforms — be these MVS or Windows NT via UNIX, Windows 95 or any other operating or communication approach — as the medium of connecting more than is achieved today.

We are already seeing this reality on our own as well as our customers' systems. The issue is increasingly — who will provide the support to link up the combinations of middleware (whether intra- or inter-business or both) to deliver desired results?

Longer term this may not matter as, in my view, broad middleware will increasingly be utilized horizontally between:

- **program**
- **departments and divisions**
- **companies**
- **businesses.**

Do you buy middleware?

This is one of the questions that still fascinates me. It is about whether users:

- **buy middleware**
- **assume it comes with whatever else they purchase (as in Quicken or R/3)**
- **wish it to be a valued-added service.**

Put another way, it seems to us that different users expect to install it while others expect it to be hidden. Still others prefer the customization opportunities which a service approach brings.

In consequence, we in Candle take a pretty broad view:

- **in part this is because we are both builders and users of middleware**
- **in part it is because we are also providers of middleware — in our traditional systems management products.**

For example we:

- **create it in technology form; this is where we undertake the development to get into all the gray details of APIs to figure out how to make the various pieces of software work together**
- **purchase it in product and exploit or adapt it for our own purposes.**

With both of these, we create new forms of middleware to sell to others (embedded in our software products). The key consideration is that it is often technically more efficient if we build middleware ourselves, albeit with higher origination costs and complexity.

That said, we have an acid test. If we can repeatedly reuse technology forms — for example by embedding them in our products (like Command Center or OMEGAMON) — we will have successfully leveraged our skills and that investment. If, however, we use it only once, we will probably have a deficit on the investment for the function obtained. This is when, and why, we use off the shelf:

- **compilers**
- **databases**
- **etc.**

The result is that Candle products are a combination of our own written code and technologies plus selected purchased middleware technologies (be these databases or messaging or POSIX or DCE or whatever). It is this ability to apply middleware which helps us separate out our (systems management) applications from operating system specific issues.

Choosing between embedded and direct middleware

Having decided between the appropriateness of the technology or the product form, I find that there is still another decision to be made. This is between whether to expect middleware to be embedded or to utilize the middleware directly. In our analysis the decision between these two will depend more on circumstance (and the particular objectives) than on principle.

If, for example, a company wants to buy a certain type of application package, often a vertical package which may have middleware built into it, then our conclusion is that it is probably best for the customer to derive the desired productivity from a point solution — even if that is to be provided across several platforms. In this instance, expecting the middleware to be embedded makes sense. The focus can be on the application benefits, in large part because the essence of the application is vertically-oriented.

If, however, a customer wants more than a point or vertical solution, a different approach is needed — one which may have more to do with enablement or infrastructure than with a specific solution. We find this most often in organizations which are looking to middleware technology to provide a broad-based connectivity between multiple point solutions and existing legacy systems. In these circumstances we believe that use of middleware:

- **is more challenging**
- **creates an opportunity to utilize middleware directly — in ways that are more akin to my technology approach except that they use off-the-shelf or packaged middleware as the starting point (rather than writing that middleware from scratch).**

As with the choice between developing our own technology and buying in product, the key determining factors will be:

- **vision**
- **investment — in terms of both time and payback**
- **organizational complications (too often the various different IS silos in an organization do not care greatly about using anything which is beyond their specialist, silo, boundaries).**

In the context of this last, middleware looked at from an infrastructure point of view is pretty close to looking at middleware from a technology point of view.

Introducing middleware

We find most companies start to use middleware in one of two different ways:

- **most of the time they first bring in middleware for one or two specific applications because they have an explicit or tangible business need, a problem they are trying to solve today for which agreed objectives, milestones, budget constraints, etc. are available; in this sense they see middleware as an tactical enabler for addressing their point problems**
- **in a few cases companies possess technology assessment groups who look at middleware as a general purpose, strategic, enablement technology with which they want to tie together other technologies (particularly existing silos); what is different about this approach is that these groups do not primarily see middleware as supporting specific vertical packages or point solutions — rather they see the broad picture into which narrow (point) implementations are expected to fit.**

In this latter case the dilemma they are attempting to resolve is the generalization of how application organization(s) should relate to middleware. When middleware is viewed in this light it is often — at least initially — little more than a concept, a set of technologies on top of which one can instantiate individual applications.

This is the top down approach. It is also the hardest to justify and to carry through — because it relies on vision and commitment to results which are not necessarily immediate. It is not unlike committing to the Interstate road system before one knows that vehicles will drive between LA and Denver or Chicago or Miami or Baltimore.

Bottom up is not always best

Our experience indicates that it is easier for most organizations to adopt middleware from the bottom up — where you:

- **face specific and immediate point issues**
- **have some proof of concept measurements (pilot projects, etc.)**
- **know the specific problems which you expect middleware to address (between applications A, B and C)**
- **can envision all the elements (technical and personal) which will deliver the solution.**

The disadvantage of this approach is the expectation that you will be able to stretch horizontally a form of middleware which was originally purchased for vertical purposes. The presumption tends to be that once middleware has worked for one (or maybe two or three) problems, then one can exploit past experience into a generalized application enablement.

In effect, this is deciding that middleware should be strategic rather than merely tactical — but after the fact. This approach — of moving from the specific to the general — runs the distinct risk of attempting to fit a quart into a pint pot. It is as if, when trying to solve the initial problem of carrying cars between San Francisco and Oakland across the Bay, one invests in ferry ships when the strategic solution is in fact a Bay Bridge.

Making middleware successful

In my view two different characteristics will shape the way middleware is considered to be successful. These will depend on how people actually utilize it and how it is sold.

To me, if middleware is ever going to be sold as a part of a solution, then the customer really needs to understand how middleware is part of the high level solution. Take, for example, a financial package where middleware is embedded. In this instance, the customer:

- **obtains the benefit of middleware**
- **avoids the pain of having to develop directly with middleware.**

This is probably the simplest and easiest introduction. But it is mostly applicable to vertical solutions and, for middleware vendors, it means that the primary customer is not the end user but the solution provider — the financial application provider. In this context commercial success comes because of function enabled, not because of the sale or implementation of middleware per se.

In contrast, success comes in a very different form when middleware is seen as a means to offer strategic leverage in a horizontal form. In this instance, where companies face the classic gap of how to justify enablement technology as a strategic foundation, it does not really matter what form of middleware is involved — be it a database, TP monitor, queuing, messaging or other type of middleware.

If you want a database you go through a cultural change as you work with that new database — as PC users are finding (no differently than their mainframe and mini brethren a generation earlier). Similarly, moving from flat files on UNIX (or any other system) to relationally-based systems is just as great a cultural shock. Developers not only have to familiarize themselves with new APIs but may be obliged to refer to some corporate group to define data elements — because those data elements are part of the corporate asset base.

The major problem, with respect to the internal adoption of corporate middleware, is that cultural and technical silos remain. People invariably prefer the ability to use their own choices in their own 'sub-system' — because they are comfortable with this; they do not want you and your department or a corporate edict to ram some other choice down their throats They want to use what they are confident in, comfortable with, is proven to them. The difficulty is that they do not want to have to experiment with their own business applications in order to justify someone else's infrastructure.

My point is this. Success in the strategic adoption of middleware is much harder to achieve than tactical or point introduction of middleware. These two approaches — tactical and strategic — describe well the ways we see many of our customers thinking. In turn, this raises the question of how to encourage the acceptance of middleware — an issue of equal relevance to users as vendors.

If middleware is too complex or too ethereal or too remote, the odds are that it may not win acceptance from customers — whereupon vendors lose. Success does not come this way — unless it (middleware) is supported by expertise. That expertise, at least in the initial years, is not to be found in many of today's user organizations.

If you accept that middleware is necessary to make multiple systems work together, and as much in the strategic (long term) as in the vertical (short term and largely tactical) sense, there is a clear need for services to accelerate its introduction (and thereby adoption). For middleware to succeed it needs acceptance.

This illustrates why Candle is so interested in middleware — and working with IBM, Microsoft and other relevant vendors. It represents an opportunity. We can:

- **encourage the acceptance of horizontal messaging middleware into MIS**
- **help companies design, implement, test and manage messaging middleware**
- **assist organizations to go through a cultural evolution which puts in context the API for a database, the API for a TP monitor, the horizontal middleware as a corporate standard, etc.**

Moreover, this can go further.

Middleware for inter-business

Typically, most organizations see middleware only as an internal (to that organization) choice between:

- **the vertical (as in point solutions)**
- **the horizontal (as in the strategic approach to connecting disparate systems).**

Another way of putting the vertical dimension is to envision middleware (comprised of components like databases, TP monitors, message oriented middleware, ORBs, RPCs, etc.) as sitting:

- **beneath the application**
- **above the operating system**
- **between platforms.**

There is, however, a very different context for middleware, which:

- **really has not been pursued that much**
- **is a key consideration to us, at Candle, as we enter the middleware arena**
- **is at least as much about inter-business as intra-organization.**

This is the use of middleware beyond departments trying to communicate with departments, subsidiaries linking with subsidiaries. It is about connecting unconnected businesses so that they can connect applications on an inter-business basis between Company A and Company B or Company C ...

The rationale for middleware here is, if anything, even simpler to understand than its internal introduction. In any selected organization there is a presumption of co-operation and broad common goals (whether it is actually true or not). Between companies, there cannot be that same presumption of co-operation or even commonality of systems.

Enabling companies — with very different systems — to be able to work together is an objective which we see middleware fulfilling. Yet, this does not fit the conventional purchasing pattern of software (either as technology or as product).

It is here that we see an increasing demand for middleware-aware services which can complement the skills in two or more organizations. Such services — which is why we are staffing up with people who know R/3 as well as Microsoft's 'Falcon' and IBM's MQSeries — must bridge between:

- **each organization's applications (whether legacy or client/server or object oriented or ...)**
- **different application vendors products (hence our interest in SAP)**
- **different middleware vendors' products (whether from Microsoft or IBM or Digital or ...)**
- **the various flavors of infrastructure to be found in (particularly) large organizations.**

The impact of the Web as middleware

Taking this a step further, think about combinations like the Web, the Internet and legacy applications. Today, one of the most sought-after characteristics of the Web is the potential to:

- **continue to use legacy applications (without having to re-write them)**
- **while, at the same time, link new applications (and platforms) to these legacy applications.**

To me, the Web is the epitome of:

- **middleware**
- **empowerment.**

From inception the Web allows you to create new applications, in hours and without needing to worry about corporate bureaucracy. It provides connections across Internet space, locally and globally as well as between businesses.

Nevertheless, as such initiatives succeed, they require an increasing amount of 'assistance'. This can come in many ways:

- **from the introduction of the dead-hand of bureaucracy (as we are seeing with some companies enforcing 'standards' to deliver a homogenized look and feel)**
- **to the opening up of new markets (via so-called electronic commerce, SET, etc.).**

When you connect the essentially lightweight Web (HTML-only) to legacy systems (with their heritage of industrial strength operations) you are not talking only about technical connections but also about cultural connections.

Speed and the CEO/CIO challenge

What we have observed is a collision of cultures because Web people do not have the same value system or expectations as legacy system people. Not only is there a repetition of the technical conflicts which I discussed earlier, but there is a philosophical difference which is captured by looking at the implications of speed.

In legacy systems, speed is predominantly and historically about throughput per second, about maximizing the utilization of system assets. In contrast, the Web is about content and being able rapidly to change that content to adapt to constantly changing markets and circumstances.

This conflict is similar to what happens with 'conventional' middleware — although I would argue that the Web and Internet are just a form of middleware. The result is, in the short term, going to be substantial ten-

sion between different value systems. CEOs and CIOs are going to have to match two competing (but which should be complementary) desires:

- **the business desire to be seen to be responsive**
- **the operational desire to ensure that all is efficiently and accurately processed.**

The CEO, or Marketing VP, will want back end systems to change as quickly as the Web. The CIO is going to want Web systems to have the stability of existing back end and legacy systems.

Thus far I have presented this in conflict form. But it does not have to be this way (although it may be a cultural characteristic to start by seeing it this way).

Instead what we see happening, in organizations that are well led, is middleware allowing proven legacy systems to interwork with whatever is new, albeit working over and above the Internet. Middleware, in this context, allows different 'system philosophies' to coexist while providing a mechanism to connect the two so that you obtain the best of both worlds. After all:

- **what is a CGI script other than a primitive legacy application?**
- **what is SET other than the linking of Web browsers to merchant servers to highly traditional credit card and banking systems (which are the ultimate in legacy systems)?**

This will not be easy, which is why I put the emphasis on leadership. Organizations with good strategic leadership will, and should, use the Internet/Web as a constructive Trojan horse to persuade both ends of the IS continuum that each adds most value when co-operating. If middleware allows each to make progress individually plus allows the two to work together in mutually dependable ways, then middleware will come into its own.

In one sense, all that the Web does is put new packaging onto an old problem. If that packaging results in better connected systems, all will win. This is why the Web is another dimension which justifies Candle's interest in middleware; it represents a faster route to delivering strategic connections — and inter-business as much as intra-business.

Lessons learned

What I have learned about middleware in recent months is that it does not compartmentalize conveniently. On the one hand there is the separation between:

- **the vertical, where middleware is generally hidden or embedded**
- **the horizontal, where its technological rawness is often all too visible.**

At the same time there are two broad categories of middleware; those which are:

- **already accepted, like databases, TP monitors, database access, the Web, etc.; this is 'narrow' middleware**
- **yet to be accepted, from message oriented middleware, distributed objects, etc.; essentially this is 'broad', distributed, middleware.**

That said what I have also learned is that tying together the above products and technology is not always as 'simple or convenient' as advertised. If anything, middleware is too much the province of technologists to be attractive to businesses. It is as if rental car agencies were trying to attract customers by offering a deal to 'put your own car together' at every airport rather than the 'turn up and drive off' deal which is what users (businesses just as much as individuals) really want.

We are, at Candle, therefore, organizing ourselves to:

- **have offerings across the middleware panoply**
- **but without necessarily declaring ourselves to be a middleware vendor.**

What we have learned is that, to succeed, we should be in the business of providing solutions for networked applications which support networked businesses — rather than in the technology per se (although we intend to be amongst the best practitioners in using relevant middleware technologies). When we say we must support networked businesses we have to communicate that we have understood the importance of starting with the business view first — not the middleware technology one.

Whether you call this middleware application enablement or application infrastructure, it is all, at root, technology. But it can only be justified if a business wants what can be delivered with such technologies. We should seek to avoid being perceived primarily as a middleware company but rather concentrate on being

recognized as the organization which knows how to exploit what middleware can create as benefits.

My objective for Candle in this area is, therefore, that we demonstrate to obtain added business value through provision of solutions for networked applications (be these Internet-based, client/server based, legacy-based, server to server-based or combinations of these). Only then can we provide the solutions for such networked businesses.

In effect, our vision is that we become a catalyst and enabler to assist the coupling of applications and businesses. We are trying to evolve, define if you will, those new business segments which are higher up the totem pole than just enabling middleware technologies which happen to support particular platforms.

Management conclusion

In this portrait of middleware, Mr. Chernick looks deeper than many of the 'traditional' middleware vendors. Rather than concentrating on the middleware itself, he is more interested in what middleware should represent to its adopter.

Where he seems to hit the nail upon the head is in separating the tactical (where middleware is embedded and thereby largely hidden) from the strategic. If middleware in the tactical instance is immediately productive, strategic adoption of middleware is all about being able to do business with others with different systems. Whether this is 'in-house' or 'inter-business' does not matter; either (and/or both) add value to existing legacy as well as new systems.

With more transactions than Web hits, where goes CICS (and Encina and MQSeries)?

Alfred Spector
General Manager
IBM Transarc Software

Management introduction

The November 1996 **MIDDLEWARESPECTRA** *included an interview with Bill Coleman of BEA Systems. In this interview, Dr. Alfred Spector discusses his views on:*

- ***transaction processing***
- ***where it fits into middleware***
- ***what are IBM's strategic transaction directions and initiatives.***

In August 1996, Dr. Spector — previously CEO of Transarc which was acquired by IBM in 1994 — was appointed as General Manager of an integrated business including both Transarc Corporation and a number of IBM's middleware groups. These products include: from IBM — CICS and MQSeries; from Transarc — the Encina Monitor and DFS (the Distributed File System, successor to AFS); and related products and technologies — from GDDM to the Encina Toolkit. In addition to these, Dr. Spector's responsibilities include further development of core software services including DCE and internet-compliant security, timing and directory services.

With $800M+ in transaction processing-related software revenues, IBM is the transaction giant which — at least for the moment — dwarfs all others. Indeed it is said that 'there are more CICS transactions processed each day than there are World Wide Web hits'.

Transaction processing: part of middleware or a different species?

To me transaction processing (traditionally TP or OLTP) is a part of middleware. I have always believed that TP is an essential technology for building robust applications, both on:

- **a single computer**
- **a network of computers.**

My reasoning is this: TP, as delivered in products today, incorporates the combination of fundamental programming, systems administration and execution facilities which are necessary when you want to build a robust application or collection of applications. For example, transaction processing has always offered:

- **run-time support for managing concurrency, whether on a server or within an object**
- **integrity, that is to say calculations which are accurate and consistent**
- **responsiveness**
- **systems manageability — being able to manage applications as well to monitor what they are doing and in what context they are operating**
- **programming constructs specifically designed to simplify use and programming of transactions**
- **support for distribution, such as load-balancing, resource location, etc.**

Therefore, I reason that transaction processing is integral to producing the reliable environments in which applications can execute in understandable and expected ways. And that, after all, is exactly what both customers and commerce expect if they are to transact business. Anything less introduces doubt into the transaction — be that your credit card balance, an entry on a bank statement, an insurance policy detail, an order taken for 1000 widgets or whatever.

Furthermore, as our systems environments evolve to be ever more physically distributed, the need for TP grows — to master the uncertainties introduced by that very distribution of systems. In networked environments, transactions will be used increasingly outside the traditional domain of business systems — probably in many of the applications that update multiple objects or servers on a network.

Now the issue is, how do you define middleware? One way is to describe it as 'anything that is above the operating system and below the application' — which, by the way, makes the RDBMS part of middleware. In this approach, TP clearly is a member of the 'middleware species'. Even when one considers a single system — whether a System 390 or a UNIX or an NT platform — the collection of functions that comprise TP must be there to support a large variety of applications. One obvious example is handling multiple, simultaneously running threads.

Heterogeneity and keeping TP separate

Years ago, transaction processing started because it addressed areas where operating systems were weak. The same characteristic can be seen with other 'middleware'; in one sense its purpose is to provide the uniform level of functionality required for implementing reliable or distributed applications on one or, even more clearly, on multiple heterogeneous platforms. So, there can be middleware for messaging, naming, managing time, security, various data models, etc.

While today's operating systems are much richer in services than the systems of the 1970s, it is notable that even Microsoft has chosen to keep its database (SQL Server) and future TP offering (Viper) out of its modern server operating system (Windows NT). This says to me that while some forms of middleware may indeed, over time, migrate into the operating system (for example, RPC support), other forms will remain as add-ons to supplement operating system facilities. This will be particularly true of components which are:

- **distributed (particularly in systems that have heterogeneity)**
- **not needed by everyone.**

The reason for the latter is financial sense:

- **base operating systems will have those features which (almost) everybody needs and are willing to pay for**
- **those features that are needed by the more specialized users will make more financial sense when offered as a layered system.**

The TP marketplace

If you look at the broad trends, our expectation is that unit volumes will increase; that is to say the number of machines running transactional applications will

increase as will the number of these applications. We think this is fueled by three trends:

- **the performance for a given cost of hardware continues to improve — so the cost of automating systems (from a hardware perspective) continues to decline; this opens up the smaller and medium enterprises to what TP can do for them**
- **businesses around the world are seeking to reduce costs by automating whatever is eligible; TP offers the robustness that extends that eligibility envelope which, in turn, expands the opportunities**
- **acceptance that networking (especially as demonstrated by the Internet) opens up inter-enterprise as well as intra-enterprise automation possibilities.**

I know that, when I started in this business, corporations were extremely leery about putting their computers on a public network of any variety. In fact it was not all that long ago that IBM's own intranet (VNET) was segregated from the rest of the world.

That has changed. Today's presumption is that businesses will have greater opportunity and efficiency if they use Internet technologies. So, our analysis is that these three trends will drive unit volumes.

The second issue is then about overall revenues. Traditionally IBM's TP revenues have come from the mainframe. However, over the past few years the non-mainframe business has been growing — for all of Encina, CICS and MQSeries — on platforms as diverse as OS/2, NT, multiple UNIX platforms (IBM, Digital, HP and Sun), Tandem, etc. Yes, mainframe revenues for us are likely to decline as a percentage of our total business. But with the growth in unit volumes, we anticipate overall sales rising. We are positioning ourselves to be present on millions of transaction servers as well as being on tens of thousands of our large mainframe servers.

Choosing TP rather than just a database

Some might argue that databases eliminate the need for separate TP middleware. That at least was the 'appeal' of TP-Lite in the early 1990s. For a large number of applications, however, this argument has not stood the test of time. MIS experience has demonstrated that the management of business process logic — the programming which vouchsafes the application — is every bit as complex as the storage and manipulation of data. Equally, there is little dispute that it is middleware of some form that facilitates the development and management of the application logic. The question is — what component will supply what is needed?

Five years ago a prevalent view was that this would be delivered by stored procedures running within the database. Two things happened:

- **with the enormous productivity of networks it became clear that inherent heterogeneity meant that all automation tasks would not be completed via one single database or vendor; real solutions often include a diverse collection of different databases, abstractions and applications which need a consistent process integration mechanism which is able to deal with that heterogeneity**
- **MIS experience with stored procedures revealed that these work fine in relatively simple instances but, as the scale increases, they are insufficient; you do not have to believe me on this — Bill Coleman of BEA Systems offered the evidence in the November, 1997 MIDDLEWARESPECTRA (page 20).**

The net of this is that something like TP is needed. It is interesting that BEA (the startup which acquired the rights to Tuxedo), Microsoft (with its future Viper technology) as well as IBM are following similar paths. Developing this a little further, I can envision instances where organizations with multiple with databases will eventually add or introduce some form of distributed transaction processing in order to integrate both data and processes:

- **within their organization**
- **between organizations.**

What this allows those organizations to do is:

- **continue to use their existing database(s) and applications**
- **extend the reach of these — without changing what exists — to introduce links to other databases and processes via a transaction manager.**

If you think about any application, there is an underlying schema for ordering the data that needs to be there. As you start to put application logic around the schema, you need carefully to compose that logic if it is to be manageable in the longer term. If you are a customer with existing stored procedures, it is unlikely that you will throw these away — they already are an essential part of your applications.

But this is insufficient if you want to invoke these stored procedures (say) with other stored procedures on other systems or with SQL calls, VSAM or other middleware. You need more — for example if you want to provide a service via existing applications to respond to a browser on the Internet.

The net effect is that databases are not in competition with TP in many application areas. They are complementary, despite what was thought by some in the early 1990s. Indeed, my business works closely with the database vendors (including IBM's IMS and DB2).

Synchronous is not all

To create these distributed transaction applications requires more than just TP function. It demands common services, for example directories and security services, to:

- **provide secure access**
- **generate common times**
- **find servers or business objects.**

Equally, the synchronous — TP approach — is not the only way forward. As MQSeries has demonstrated, users have many situations where the best way to integrate applications may be by reliable asynchronous messaging where an application sends messages reliably to another application.

Combine all of these and you can build an inverted pyramid (Figure 3.1) which includes:

- **common services — like security and directory services**
- **synchronous communication and the integrity mechanisms which surround it**
- **asynchronous communication and the integrity mechanisms which surround it (see also my analysis published in OTM Spectrum, November 1993, page 4).**

A point I like to make here is that the points of the inverted pyramid are connected. They are not necessarily separate pillars operating in splendid isolation. That is to say, they are not mutually exclusive — even between synchronous and asynchronous. For example:

- **CICS or Encina use the same naming services at the same time as MQSeries.**
- **MQSeries may exploit Encina as well as CICS while using security services (and MQSeries will be integrated with our security mid-year).**

The pyramid demonstrates why we are continuing to invest in pushing MQSeries further as we:

- **expand onto new platforms together (we just announced partnerships where MQSeries will support Apple and Stratus, to add to the 20+platforms already supported)**
- **introduce a new release in 1997, with improved integration, simpler administration, installation and better performance**
- **add additional programming interfaces (for example, MAPI support, so that MQSeries can offer the reliable infrastructure beneath mail enabled applications)**
- **etc.**

We are even researching how to store MQSeries queues in relational databases so that many customers will have one less store to administer.

Overall, the choice is the customer's. At one customer we will see asynchronous messaging being used alone without any new transaction processing environment — often to bridge the gap between very different applications. At another customer we will meet traditional transaction processing which has little need to possess asynchronous facilities. At a third it may be that the way to tie different TP systems is through use of an asynchronous mechanism to enable two existing synchronous applications to work together.

It is when these components work together that they provide the basis for building the applications of the future. This is why it is so architecturally encouraging that both BEA and Microsoft — although not as developed in delivery terms — have clearly come to similar conclusions. I think it's fair to say we in this industry now have a pretty rational architecture for how distributed and mission critical applications should be supported.

Organizing IBM's transaction business

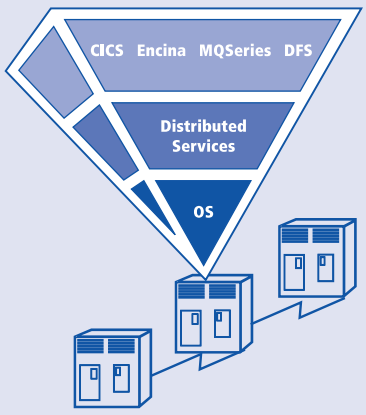
The IBM Transarc Software unit within IBM is a middleware business unit operating under the aegis of the overall IBM Software Business (which includes Lotus, Tivoli, databases, application tools and networking software amongst its responsibilities). As such, the transaction systems unit is not in the database or applications or groupware businesses. Our business includes transaction processing, distributed services and enterprise file sharing of unstructured data.

What we provide is transaction server software (you could call it our 'transaction series') that contains the CICS and Encina programming styles on a common engine. We are already by far and away the largest — whether measured by revenues or installations — transaction vendor. We intend to grow our overall penetration of the business world to way beyond our existing mainframe base.

As I discussed above, we also produce MQSeries, the asynchronous messaging products which run on more than 18 non-IBM hardware platforms (plus System 390, AS/400, RS/6000 and Wintel). Our objective with this product is to remain the message-oriented middleware of choice — with the broadest as well as the deepest penetration of all sizes of business. In addition to these three families we offer:

- **the global Distributed File System (DFS), which offers a single view on unstructured byte-oriented data**
- **core infrastructure technologies for our own as well as for many other IBM products — core security, naming and automated time synchronization; these products include DSSeries, our DCE-compliant product set, as well as internet-related technologies, such as LDAP and public-key security technology; you will see these differing service technologies become part of a common base technology set**
- **a collection of other products — from the IBM firewall product to transactional processing tools — and specific applications built for a particular marketplace.**

Figure 3.1: The middleware pyramid



Geographically we have already rationalized how we develop our products. Working from West to East:

- **Austin (with teams in Raleigh and Endicott) has responsibility for the evolution and delivery of infrastructure services — including directory/naming, RPC, security, time, etc.**
- **our Transarc subsidiary (Figure 3.2) in Pittsburgh will produce all transaction servers for non-mainframe platforms (for both the CICS and Encina personalities); it also has responsibility for enterprise file sharing via the DFS product**
- **Somers (NY), where I work, is our business headquarters and directs overall strategy (I operate with a very small staff)**
- **Hursley has responsibility for our mainframe (OS/390 and VSE) CICS products as well as MQSeries; in addition it is responsible for all 'extended transaction' origination — from Java support through Internet linkages to Lotus Notes connections to long running transactions and various other initiatives that will likely be launched for our high performance customers before being incorporated in other products.**

Worldwide, we provide sales and services from a number of locations, including the above ones. In addition, my team and I do not forget about IMS, a product which is dominant in its (high-end) segment. As an example of how we are further integrating IMS with our distributed offerings, we will soon have single unit of work processing (using the Encina Transactional RPC) between our distributed transaction products and IMS.

Our business strategy

That sounds, maybe, like the previous IBM TS development organization of the last few years — albeit with the closer alignment of Transarc Corp. But the transaction systems business today goes further than it used to. We now possess our own:

- **global integrated marketing operation**
- **jointly managed sales force — comprising both Transarc and IBM sales representatives, which operate as an integrated unit.**

In essence we have control of our own destiny — from development through marketing to sales. This is a significant change from past IBM practices.

Another change is that you will see us competing more aggressively in the market place. One measure of this will be the many ways we intend to make ourselves appealing and accessible, beginning in early 1997, to smaller and medium size businesses (and I am including departments in this category) in addition to the larger enterprises. We know we have to do this if we are to reap the rewards of a marketplace that is growing.

Our underlying strategy for a number of years has been to recognize the need for common services, transaction processing and asynchronous messaging. All are needed for IBM to offer a rational set of robust products that enable robust enterprise (from smallest to largest) process automation. This part of the strategy does not change.

What does change are the focal points. We are now more focused — not necessarily an IBM strength of the past, but one I am personally intent on delivering. We recognize that there are two distinct (if overlapping, at the departmental level) segments:

- **the traditional IBM market — made up of relatively large customers doing relatively complex processing; these are existing users of CICS, Encina, MQSeries and others of our products; they are our lifeblood today and we have no intention of ignoring, neglecting or otherwise diminishing our support for them; as these customers automate and integrate more and more of their processing, we will continue to expand our solutions**

The future roles of Transarc, Encina and DFS

In 1994, IBM acquired Transarc — where I had been a founder, Chairman and CEO. In the six years when Transarc operated independently, it developed our initial product AFS, then DFS (now shipped by many vendors including IBM), the Encina Toolkit and the Encina Monitor — using the DCE services described by the then Open Software Foundation (and now part of the Open Group).

In my view, what made Transarc successful was its combination of development and marketing cultures. This produced:

- **consistent, rapid growth over its entire history**
- **the family of Encina products, which brought modern TP to UNIX over a common set of services (DCE)**
- **a distinct group of vendor and customer adherents (just look at the success of DECORUM).**

Transarc is now part of the IBM family of companies. Within that group, Transarc's utility is even greater than it has been in the past — for IBM can capitalize on the products, the talent of the people and its unique service offerings. In my new organization, Transarc will retain development responsibility for DFS as well as having responsibility not just for Encina but for all the transaction series products for distributed platforms.

But, in my view, another part of the Transarc heritage will prove equally valuable. That is, Transarc has deep experience in the heterogeneous, distributed marketplace. It has been able to sell large transaction systems and help customers be successful with these.

This is an enormous asset for IBM and one which I intend to capitalize on in our merged TP and distributed computing-oriented businesses. The net result is that Transarc will live on and thrive as an important entity within IBM.

Figure 3.2: Transarc in IBM

- **the 'new' opportunities — where we must communicate exactly what value we offer to a larger and more diverse set of customers (from users to system integrators to package and tool developers to software vendors to business executives); these represent — directly and indirectly — our growth opportunity.**

We know well how to work with the first of these groups but our focus on the latter has not been there. That will change. I said earlier that we would compete like our competitors. Attracting partners who know that they will profit from using our products is the key. This is particularly true for the smaller business customers. We have to make sure that our products:

- **are not just focused on solving the most complex technical problems of our large customers**
- **but also embody ease of use combined with the robustness which all businesses demand.**

I am personally determined not to allow ourselves to lose sight of either our large customers or our new ones. Just like the car industry uses Indycar and Formula 1 car racing to develop specialty technologies that can then be generalized into more commonly applicable forms, so we intend to do the same. In this way we can:

- **satisfy our most complex customer requirements**
- **develop a significant amount of experience before undertaking generalized simplification and introducing ease of use**
- **avoid marginalizing ourselves by over concentration on the high end market.**

So, our fundamental strategy is not changing. But our execution is.

We already are a big business which has grown by providing transaction processing, asynchronous communication, file sharing, as well as infrastructure. Our business strategy, therefore, is to grow bigger by providing more function to vastly more users; with this users will achieve more in their businesses through increased capabilities to process many more dependable transactions in ever more innovative ways. The Web is just a starter.

Opening new markets will clearly be a measure of our strategy's success. I expect to be doing deals with third party software vendors — some of whom may come as a major surprise, given past experiences — who recognize what we have which will assist them.

Challenges, threats and opportunities

I think our greatest opportunity is that there is more need for what we offer than ever before. With the explosion of distributed systems plus the Internet, the demand for dependable processing (whether asynchronous or synchronous) cannot be doubted. We see more articles in the press, more customer interest and more competition than has been around for more than a decade. In this context, probably our greatest threat is our own inability to communicate and then sell the reality that IBM has plenty to offer.

In the early 1990s, it looked as if relational databases would be the prime threat. Few people think this today. Instead IS is turning towards what TP offers as middleware. For example, that is how our Transarc subsidiary was able grow to having over a thousand deployed sites and 300 employees.

Instead, possibly the greatest challenge is introducing new technologies to customers at a rational rate (for them, not us) which:

- **presents the benefit of the new technology rapidly**
- **minimizes customer disruption.**

Put another way, this challenge is about how to roll out new technologies that work right, do not require hoards of technicians and are available at the right rate. Keeping things simple for customers is important. With increased opportunities comes increased competition. I regard this as good news:

- **as Oracle, Sybase and others demonstrate, noise attracts attention; in the past IBM's strengths were without much notice; with our new competitiveness, there will be no shortage of fanfare**
- **we have the broadest heterogeneous coverage of platforms with the largest number of proven technologies and customers; BEA has a strong UNIX core: but so do we on the four large selling UNIX platforms — AIX, Digital Unix, HP-UX as well as Solaris; Microsoft is NT — yet we have CICS, Encina and MQSeries already on NT**

- **we support many other platforms as well, particularly for MQSeries and, of course, we integrate with the mainframe servers better than anyone else.**

If we cannot win off an opening hand like this, we will be utterly remiss. Our biggest risk in the past would have been organization. But, with the increased determination to be a leading middleware provider and our recent re-organization into a collection of integrated software businesses, IBM has recognized the opportunity. With an integrated sales and marketing team, and the symbiosis with Transarc (the same symbiosis that has been working well with Lotus and Tivoli), we think we are ready.

Lessons learned

As a relative newcomer to IBM, one lesson has become firmly imprinted. This is that customers, particularly larger ones, place an enormous importance on the long term. They really are looking at multi-year, sometimes decade long, commitments. They want the same commitment from their vendors.

Without doubt this is a vital business strength. We have been around the TP business for a few decades — forever in IT terms. Competing with IBM will be difficult for other, as will competing with the formidable skills which my 2000+ global team possess.

On the other hand what I have learned managerially is that — within a company the size of IBM — the clarity of the strategy has to be honed precisely or else the message gets lost, both internally and externally. Since taking over the transaction business I have been hammering home what I have discussed here: common services, TP, asynchronous messaging, enterprise file sharing, etc. — and in terms of what problems these technologies solve for customers.

But my last lesson learned is probably the most important of all. Irrespective of what glories you possess, these are worth nothing unless you concentrate on the basics. Neither customers nor markets will forgive you if you fail to:

- **meet the requirements**
- **launch the products**
- **market and sell them**
- **possess a consistent message.**

IBM has not always been good at doing these simultaneously. If we are to meet our strategic growth aspirations, we will need to fulfill them all. And we will.

Management conclusion

IBM is the largest supplier of transaction software by far. Most of its success has been achieved through organic growth over the past 20+ years.

In 1997 it is poised for change. With the abundance of technical riches inside Dr. Spector's transaction business, there seems to be no good reason why the growth of the past two decades cannot be continued. Strengthened by the addition of Transarc, it is clear that IBM's competition is going to have its hands full — if Dr. Spector manages the change — with IBM moving from being the sleeping giant into becoming the aggressive software business it has the potential to be.

Does Viper measure up to traditional OLTP?

Dr. Tom Heywood
Electronic Business Centre
University of Southampton

Management introduction

Microsoft Corporation has recently made available a Beta version of its Transaction Server (code named Viper) — together with various documents describing its features and customer benefits. Arguably this is the first major new transaction processing initiative in the past two decades (the one possible exception being NCR's Top End in the early 1990s). Naturally there is great interest in Viper as it is yet another example of Microsoft taking direct aim at a traditional mini/mainframe preserve.

This analysis seeks to make a preliminary evaluation of Viper in comparison to existing transaction systems. It concludes by suggesting some of the key questions which will need positive answers before large — and small — enterprises should feel comfortable deploying it for mission critical applications.

Expected capabilities of transaction servers

At a superficial level, all transaction systems (be these Tuxedo, IMS/TM, CICS, Top End, ACMS, Encina, etc.) appear to be much the same. That is, they all aspire to provide the customer with the same capabilities.

For the purposes of this comparison, the major capabilities can be summarized under the following three headings:

- **application level capabilities**
- **end user access level capabilities**
- **system level capabilities.**

At the application level what is expected includes:

- **a program environment which permits ACID (Atomicity, Consistency, Isolation and Durability) observing applications to be written**
- **an application development and testing environment.**

At the user access level what is required is:

- **an interface (preferably GUI-based) to applications**
- **access to applications for remote users.**

At the system level the key characteristics include:

- **the guarantee that the ACID properties are maintained**
- **efficient sharing of system resources, including databases**
- **performance, from high transaction rates through 24x7 continuous operation**
- **fast and efficient recovery**
- **access to remote resources such as databases**
- **fragmentation of the system (so that applications as well as data can be geographically distributed and yet be able to operate as a single total system)**
- **integration of disparate, often 'legacy' as well as client/server, elements into a single total system**
- **system management such that sets of distributed subsystems appear as one single total system.**

This examination of the Viper Beta — taken as being representative of what Microsoft's Transaction Server (MTS — Figure 4.1) will be when eventually available later in 1997 — is based on these three capability levels and their supporting characteristics.

Application level considerations

Existing transaction systems, such as CICS, have developed their capabilities over more than two decades. However, progress has not been equal in all areas. One of the initial objectives of these systems was to provide a friendly programming environment.

Ease of programming, however, has come to mean much more than the original need to hide multi-threading, telecommunications line-sharing and the like. Today 'modern' ease of programming is expected to include:

- **a comprehensive application development environment**
- **visual programming capabilities**
- **a choice of the popular programming languages**
- **natural use of object technology.**

During their life times, systems such as Tuxedo or IMS/TM have made many and significant improvements to ease-of-programming. However, the high priority given by IBM (for example) to satisfying its top end customer requirements for improved system level capabilities has meant that improvements at the programming level often did not kept pace with the high pace of invention in languages and development tools elsewhere in the systems area — particularly those emanating from the PC world.

In addition, there exists one inherent problem connected with providing a new application program development environment. This has had a major influence on restricting programming improvements.

This problem stems from the fact that the programming environment which transaction middleware must provide for its applications is unique to that system. Because it is the not same as that provided by the native operating system, compilers and other application development tools and environments do not immediately carry over from the underlying general purpose operating system environment without some element of rework.

One consequence has been the relatively small number of application development environments which have been made available to transaction processing programmers. This is not to suggest that full application development environments have not been available; they have. But it is fair to say that these have:

- **been few in number**
- **have come with a high cost**
- **do not readily translate from one platform to another.**

In turn, this has meant that programmers for such systems have had to undergo lengthy education and training specific to the chosen environment. Such training, if not education, is frequently not portable to other systems or platforms.

One other problem is applicable only to the mainframe environment. Here there are restrictions on the system functions which applications may invoke. These restrictions were necessitated by the use of subsystem unique tasking which have had a further complicating influence that restricted production of application development environments.

Microsoft's Transaction Server architecture — the application level

One area where the application development tools for existing transaction systems have not been scarce is in the first layer of the three-tiered (Presentation Logic : Business Logic : Data Logic) application model. It has been possible to implement the Presentation Logic layer using a wide variety of modern tools and paradigms. Innovation has not been lacking.

These innovations have provided:

- **a front-end to the Business Logic, which can be implemented in any way convenient on the transaction server**
- **a natural reservoir of capabilities (system and personnel) which MTS looks well set to exploit.**

The Microsoft Transaction Server claims (until it is available it is impossible to say one way or the other) to have made server application development much easier, and so much lower cost. It has set out from the very beginning to allow applications to be developed from any language which produces ActiveX DLLs.

It has done this by not making the transaction processing environment unique. The transaction processing environment is made available simply by the system administrator declaring that an application is transactional. New functions are provided as API extensions in a library. As a result the applicable language products include:

- **not only current Microsoft Visual Basic, Visual C++ and Visual J++**
- **but also a wealth of software development tools produced by third parties (many having been originated to address the Presentation Logic challenge) from vendors as diverse as Sybase (PowerBuilder), Delphi (Borland) and even COBOL (Micro Focus).**

Given that there are tens, if not hundreds, of such tools, the expectation is that the list of supported development tools will grow and grow. This looks to be a major strength of the Microsoft Transaction Server environment.

One important side effect is that the cost of educating a programmer with the skills needed to write for the new environment will be much less than that necessary to:

- **move to a completely new tool**
- **learn one of the existing transaction environments.**

There will be a cost connected with learning the new parts of the API specific to the Microsoft Transaction Server. Nevertheless the number of these API extensions (so far) is quite small. This compatibility should also mean that the large market in ActiveX third party components will expand to encompass the MTS world.

Taken together, what Microsoft looks set to achieve is to enable most PC/client programmers to participate in MTS. With millions of such programmers — whether in mainframe COBOL or UNIX C or C++ or PC Visual Basic or PowerBuilder or whatever — this compares very favorably to the alleged 300,000 (at its height) CICS programmers. The appeal to organizations at this level is manifestly clear.

The Internet dimension

If one part of popularity can be achieved, or measured, through an appeal to a breadth of development skills, then Microsoft has added another point of appeal by integrating the MTS closely with its Internet Information Server (IIS). Bringing transaction processing close to the hottest issue of the decade is a real attraction.

Because of this close link the Presentation Logic layer and some of Business Logic layer of the three tier model can be written as active Web pages — using any mixture of HTML, Java and probably any other technique that bursts upon the Internet world. The abundance of easy-to-use development tools and trained programmers will again ensure that any lack of transaction skills will be significantly less of a problem for Microsoft Transaction Server systems than existing transaction systems.

This strategic thrust by Microsoft is, possibly, the single most significant factor in the whole MTS picture. Microsoft seemingly has judged that the Internet will grow:

- **from being an environment where companies publish information**
- **to one where companies provide interactive access to services (including transaction capabilities).**

The Microsoft vision is that access will move from passive reading (receipt) through interactive requests for information — for example price quotations for a specific business activity — to business transactions which carry financial and other commitments. It is these latter stages which will require true transaction processing capabilities. On the evidence so far, MTS is designed to provide what is needed.

End user access

The various existing transaction systems available to customers (Top End, Encina and CICS, to mention just three) have, over the years, much increased their end user friendliness and the ability for end users to connect to many different and geographically distributed transaction servers. For example, end user friendliness has been addressed by the use of:

- **field-oriented displays**
- **screen mapping facilities.**

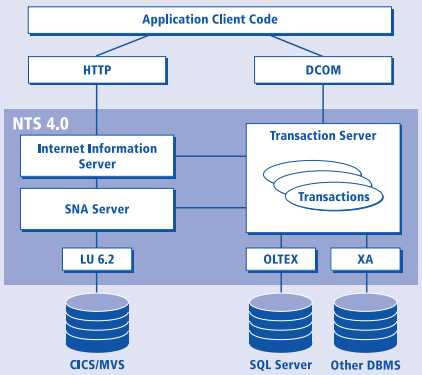
These have performed an excellent job, especially in the high throughput data entry arena. Indeed, many existing end users have felt bemused by the windows-like point-and click interfaces which have seemed to slow users down rather than improve productivity.

However, where this has not been the case — where GUI interfaces have proven their benefit — then the GUI capability has been added through the provision of GUI clients. It may be argued, therefore, that existing transaction systems — with GUI clients — have provided all that is needed by current transaction users. Similarly access to disparate and geographically distributed systems has been provided via initial access panels or through transaction routing.

But this is not the whole story. The Internet explosion is having a profound effect. In a world where new end users are most familiar with using Web browsers and expect to discover the existence of servers rather than work only with those for which their system has been programmed, existing transaction systems will need to change. Already there is much activity to make available capabilities to run the data streams for mapped displays through Internet browsers.

Enabling mainframe transaction applications to be accessed via a browser ('putting 3270 screens into Netscape') is one dimension, and a valuable one. But it should not be confused with what can be achieved with an active Web page using ActiveX components — although it should be possible to provide much of the 'Internet Server' discovery capability by front-ending existing systems with Internet servers.

Figure 4.1: Microsoft Transaction Server Architecture



Microsoft's Transaction Server architecture — the end user level

As indicated above about application development level, the Microsoft Transaction Server was developed with an architecture and design adjusted to the Internet. The close integration of the MTS with Microsoft's IIS means that the requirements of new end users, and the many businesses who will be new to transaction processing, will likely be met in a way that naturally evolves from their use of Microsoft platforms (Windows, Windows 95, Windows NT) and even first contact with the Internet itself.

At the same time, Transaction Server does not restrict access to Internet users. With a distribution architecture based on ActiveX, an application which is built

Does Viper:

- **really tidy-up resources allocated to an application after it terminates, so that no matter whether the application ended normally or crashed, all the resources (for example memory) that were allocated are returned to the system? Does this work for subtasks (essential if a system is to run for 24 hours a day, for 365 days a year)?**
- **itself never leak storage?**
- **work satisfactorily with a lot of long running tasks?**
- **facilitate long running interactive applications?**
- **allow workflow applications to be written?**
- **cater for very high transaction rates (does DCOM work as speedily as other protocols?)?**
- **really not fail more often than is acceptable (or than the competition)?**
- **support swift, warm restarts?**

Figure 4.2: Sample questions to ask of Viper

from these components will be able to use DCOM for its Presentation Logic to communicate with the Business Logic Layer. Microsoft even asserts that this will be relatively easy (which begs an implicit comparison with the subsystem specific characteristics that constrain traditional OLTP products).

At the system level

Existing transaction systems have enjoyed many years during which to develop the capabilities listed above under this heading. In that time, the like of CICS, IMS/TM and others have made great strides from the frail opening positions they occupied during the late 1960s and early 1970s. As discussed earlier, the objectives were originally: ease of programming (essentially hiding what are regarded as lower level functions); use and sharing of teleprocessing lines; concurrent access and update of files; and efficient sharing of other system resources (main memory, processor, etc.).

However, as users became sophisticated and transaction systems began to play a more central role in essential enterprise operations, new facilities were added:

- **separate databases could be accessed and shared using the two phase commit protocol**
- **recovery was more assured and faster**
- **security became more of a central system issue**
- **continuous operation became a reality.**

This latter had implications not only for reliability and recovery. It also required that system data tables and, ideally, code should be able to be updated without needing to stop the system. These improvements were not easy to provide. Furthermore, it took a considerable time before confidence in the added strength of these capabilities was accepted. This is hardly surprising.

Today, system level robustness is the forte of existing transaction systems. That is why such systems process such a large percentage of the world's business transactions. Indeed some have even argued that there are more CICS transactions executed per day than World Wide Web hits (but for how long?).

Simultaneously, system management considerations have become more and more important as systems have become both larger and more distributed over geographically separated sites. In traditional transaction processing, this has received much attention over the last five years.

There now exist some highly sophisticated products to make life in this area very much easier because this area cannot be thought of as passive — providing only tools to add and delete resources and stop and start systems. The best tools include monitoring capabilities which allow for automatic triggering of warnings, or alerts that remedial actions should commence, if some system resource is close to exhaustion or saturation. Arguably this area has become the strength of existing TP systems.

Microsoft's Transaction Server architecture — the systems level

MTS lays claim to all of the functions in the system level. This is bold talking (although Microsoft has had the advantage of being able to watch what others have achieved).

The central engine of MTS is composed of:

- **NT Server 4.0 base services**
- **the Microsoft Distributed Transaction Co-ordinator (which is part of Microsoft's SQL Server 6.5 database)**
- **addition of the MTS services to complement the above.**

This combination means that much of MTS's base capabilities are built upon pre-existing technology and so possess a history of customer usage to back up claims of mainframe-like reliability. Included in these base capabilities are:

- **high speed thread and process dispatch/termination**
- **transactional database update co-ordination**
- **process isolation.**

As with application development, Microsoft is determined to make ease of transaction use a major strength of, and attraction to, its Transaction Server. On the evidence of the Viper Beta it is providing tools to make it easy to manage the resources of a complete system.

The main tool here is the Transaction Server Explorer which is designed to facilitate the management and deployment of the components of an MTS solution. Wizards are provided to assist with:

- **creating new 'transactions' (packages)**
- **adding components, servers and clients**
- **configuring transactions and security.**

In addition Microsoft promises to make the interface for system administrators very familiar to those used to managing non-transactional Microsoft servers — and to add XA access to/from other (non-Microsoft) XA-compliant resources. Having claimed all the above, however, it is also only fair to point out that MTS is unproven. Viper has yet to become MTS — as a released product. MTS has yet to receive the years of 'beating' with which the likes of Tuxedo, IMS/TM, CICS and others can claim proven system capabilities.

The weakness here is less, necessarily, in the architecture or implementation. It is more in the unfamiliarity and uncertainty — which only time can cure.

Management conclusion

Microsoft has started down the path of providing a transaction server with four main apparent aims:

- ***making it as easy to develop new applications for the Microsoft Transaction Server as it is to develop them for desktop systems***
- ***provision of business-critical functions — for example security and reliability — as found in existing transaction servers but with complexity hidden from (or transparent to) the developer***
- ***making the management of a complex network of Microsoft Transaction Servers and their components no more difficult than a set of departmental servers***
- ***ensuring that the Microsoft Transaction Server is equally well suited to business transactions that are being produced and accessed using the Internet and World Wide Web techniques as those being produced and accessed using more traditional methods.***

From what is currently available as beta 'product', it appears that the Microsoft Transaction Server has started down the right road to meet these objectives. Ease of use, deployment and management clearly figure highly in the thinking that has gone into the design.

However, experience with new software in the past has taught us all that such complex systems take time to mature. What might appear at first to be a good easy-to-use interface can turn out to be not so good when all the necessary functions are added.

To avoid being unintentionally 'mugged', there is value in asking detailed questions, all of which need to be answered with a resounding 'Yes' before deciding to employ MTS in a mission critical application. One such selection of such questions is shown in Figure 4.2.

Having said all the above, one other caveat is applicable. On launch MTS will be available on Windows NT — and only Windows NT. This will change in time through the good offices of the likes of Digital, Software AG and Tandem.

But consider the range of platforms on which Tuxedo sits (multiple UNIXes and NT) or CICS — which you can buy today on MVS, VSE, AIX, Digital UNIX, HP-UX, NT, OS/2, Tandem and with Solaris to come. The trade-off between the system dimension, the range of platforms and the ease of use and programmability is set to be an interesting confrontation.

Middleware : the next generation

Allan Lees
President & CEO
Softwire Corporation

Management introduction

In recent years companies around the world have been implementing middleware in order to resolve a critical dilemma. Legacy applications — which are vital to running their business — are also endangering their future. This is because these applications cannot readily be modified to suit the changing business requirements which demand that these companies rapidly be able to:

- ***offer new products and services***
- ***support these with appropriate systems.***

They need to be able to do this in order to survive and prosper. Yet inflexible old-style mainframe systems often lack the necessary architectural flexibility to be able to adjust.

In this analysis, Allan Lees — President and CEO of Softwire Corporation of Larkspur, CA — examines how and why more than 'traditional' middleware is needed to support the performance and robustness which mission critical systems require and how this can be used to connect existing high performance legacy systems to new ones and thereby build 'real-time bridges' between generations.

Defining the next generation of middleware

Businesses are under increasing pressure to resolve this dilemma — and to do so quickly. Simultaneously, the pace of change is fast — and getting faster. Regulatory walls are being dismantled; new players are creating entirely new services, and customers have more choice and ever less 'brand loyalty' than before.

Businesses need, therefore, a solution which enables them both to continue to use the legacy systems they have invested in over the years and simultaneously create and deploy the new mission-critical systems that will support future growth. Conceptually, middleware vendors have the solution: an intermediate tier that links legacy systems to new applications in order to get the best of both worlds. However, middleware has often not lived up to its early promise — because it has been difficult to use, administer and carry forward.

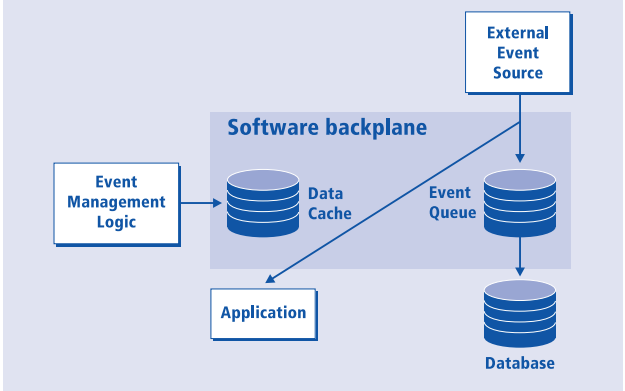
In the August 1996 **MIDDLEWARESPECTRA** (page 2), Charles Brett set out five requirements of middleware: ease of implementation, availability of tools, flexibility, accessibility and value. To support mission critical ser-systems, the next generation of middleware needs to meet Brett's criteria plus add to these, by providing:

- **real-time performance**
- **guaranteed delivery and transactionality**
- **industrial strength robustness**
- **ease of use**
- **ongoing development**
- **openness.**

Real-time performance

Message oriented middleware products typically deal with tens, or perhaps a few hundred, transactions per second (TPS). For customers looking to middleware to help them offer new mission-critical services, it is our analysis that this is inadequate by at least an order of magnitude. More and more, research indicates that businesses are facing the need to implement real-time systems. If this is the case, then middleware needs to provide real-time performance too.

Figure 5.1: Simple schematic for software backplane



Take, for example, a new telecommunications billing system (but it could be retail ordering system or ...). Typically it will need to take high volumes of customer detail records (CDRs) from a switch, parse them, generate multiple events and update several databases — all within fractions of a second.

Similarly, big trading houses need high-capacity real-time middleware to reduce their exposure. Such exposure can occur, for example, when derivative models lag trading changes in the underlying securities.

At the same time, new Internet-based businesses are springing up every day. They bring with them the need for new kinds of enabling technology. An Internet trading service will lose customers (and open itself to lawsuits) if, for example, a customer executes an electronic trade when the stock read $23 3/8 on the home computer but then ends up paying $23 7/8 because of any time lag before the provider completes the trade. Similarly, with electronic deposits and debits becoming increasingly the norm, banks that rely on overnight batch runs to update information on their net cash positions are finding that they are increasingly exposing themselves to unacceptable risks.

All of these situations share a common need — for middleware which can handle large numbers of transactions per second — in effect to provide real-time performance. Such performance can be achieved through the combination of several features:

- **queues to provide throughput and latency flexibility**
- **in-memory data caches**
- **compiled event management logic**
- **software backplane architectures (for example, the TIBCO Information Bus, Talarian's SmartSockets, Softwire's Distributed Services Environment, etc.).**

A simple schematic for this form of middleware is shown in Figure 5.1.

Guaranteed delivery

If executing a trade at the wrong price is bad, the consequences of a system losing a trade altogether are potentially disastrous. Guaranteed delivery of events is increasingly important for almost all kinds of business.

The days are gone when telecommunications infrastructure providers could ship products that might lose approximately 30% of the CDRs coming from a switch. At the time, this was not perceived as a problem: the monopoly carrier could simply increase everyone's bill to compensate. Today, itemized bills and cut-throat competition make such stop-gap solutions unacceptable. The consequence is that telecommunications companies must solve such underlying problems.

Financial services providers are facing similar challenges. Once they could absorb small 'disappearances' of money in their systems. Today, now that the sums are growing larger — with transfers of money more frequent and with customer expectations of accuracy rising — relatively small discrepancies have the potential to become large absolute errors.

Middleware in such settings must guarantee that it delivers all the messages it receives to their final destination. Transaction-based middleware can compensate for local failure by using the log to roll back a transaction and rolling forward once an alternative is established. Sometimes this transactionality will be synchronous, as has always been true of transaction processing monitors. Unfortunately, a purely synchronous approach has two significant weaknesses:

- **with two phase commit, adding nodes exponentially increases the probability that no transaction will be committed anywhere because of a regional failure somewhere**
- **batch-oriented legacy systems are not easily compatible with TP monitors**

Broadly, the asynchronous approach is preferable — because it can:

- **deliver high throughput (using queue-and-catch-up)**
- **mimic a synchronous system when necessary (as Brett noted, asynchronous middleware can be almost synchronous, but synchronous can never be almost asynchronous).**

Timing should be a feature, a choice. It should be tunable according to specific requirements — rather than the inflexible constraint which is too often built into the heart of today's middleware. Together, the use of queues and logs provide the guarantees that every event will:

- **be accounted for**
- **ultimately reach its intended destination**
- **be delivered according to the timing requirements of the fundamental business process, and not according to the somewhat arbitrary requirements of the middleware used to implement the supporting information systems.**

Robustness

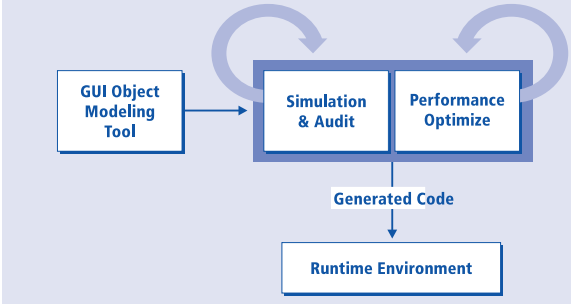
Mission-critical middleware also needs to offer high levels of robustness. Imagine a common situation where a complex piece of middleware links a variety of live systems — and suddenly a key database goes down.

Middleware needs to find and implement the appropriate rule or set of actions to deal with the problem (for example, to fail over to a different database or to queue transactions until the database comes back up). It needs to insulate other components of the system (including the middleware itself) from local failure.

In turn this separation has spin-off developmental benefits. Aggregating different pieces makes it both simpler to develop each piece as well as offering opportunities to swap out, and in, component elements.

Additionally, the middleware itself should never go down or have to go down — not even for bringing up new versions. It should also:

Figure 5.2: Keeping the model in sync. with the code



- **be fault-tolerant (as well as running on standard platforms — UNIX, NT, etc.)**
- **not require much attention, if any, from systems administrators.**

Self-healing software may be a relatively new concept. Yet we have found that there is much to be learned from the experiences of the fault-tolerant hardware vendors. There is no fundamental reason why software should not aspire to the 'five nines,' or 99.999% reliability that is expected of telecommunications networks, power grids and certain defense systems.

What is clear is that middleware at this level (which almost certainly will need real-time characteristics) has yet to be implemented in real-world systems. That being so, the direction for the future is clear:

- **customers need this kind of robustness**
- **middleware must provide this if it is to ameliorate, rather than compound, the problems experienced by IS (and organizations as a whole) across the globe.**

Ease of use

Over the past few years, GUI tools have helped to cut down the application backlog for client/server systems. Our experience has shown that businesses would like to harness this productivity for middleware as well — not least because IS managers are increasingly aware that projects built around hand-written code essentially place the entire system at the mercy of the weakest programmer on the team.

Today's GUI-based tools, by contrast, create reliable and reasonably efficient code that can be modified and carried forward as requirements change. Few people would willingly return to the 'mountain of COBOL' approach that until recently dominated development management thinking.

That said, it is fair to say that thus far middleware has not taken advantage of the improvements in GUI tools. Too much middleware today still imposes:

- **primitive programming environments**
- **difficult-to-use functionality**
- **an excess of APIs**
- **a fundamental lack of intuitiveness which makes even UNIX look user-friendly.**

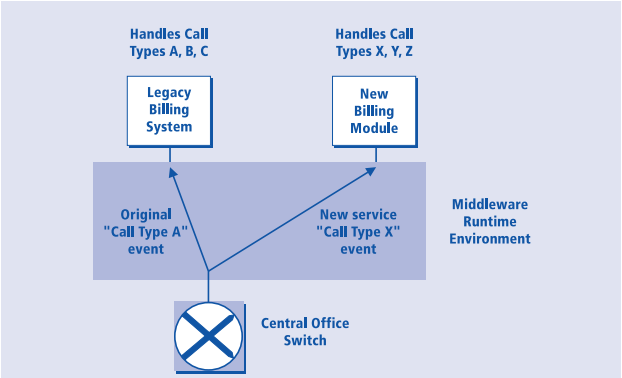


Figure 5.3: Example in use

Although PowerBuilder and similar tools cannot yet provide the requisite functionality to generate middleware-style code, there are a number of suitable tools on the market (for example Project Technology's Bridgepoint Analyst object modeling environment). Even Visual Basic should quickly reach the point where it will be an appropriate tool for middleware applications. One consequence of this is, as argued by Brett, accelerated support for off-the-shelf development tools which leads, in turn, to customers adopting middleware sooner.

In addition to enabling use of off-the-shelf tools, middleware should also provide the kinds of simulation, audit, debug and performance tuning capabilities we expect from good development environments. As middleware becomes ever more critical to 'bet-your-company' systems, it must provide the developer with those services that help ensure that deployed code is up to its task.

The old (traditional) approach, of using models to generate non-performant code, and then hacking that code into shape, is both inefficient and one-way. Once the code has been 'hand crafted' it is almost always no longer mapped to the original model. That model became useless.

The goals of efficiency and reuse imply a code generation architecture that provides:

- **the ability to create the code you want**
- **while maintaining the one-to-one relationship(s) with the object model.**

In our view, development architectures need to look something like that shown in Figure 5.2 which lends itself well both to:

- **using off-the-shelf GUI tools**
- **providing object templates for those GUI tools to use.**

For example, a code generator which links to the GUI tool could provide:

- **an 'SNMP Publish' object**
- **'Persistent Store' objects (variations mapped to particular vendors' databases)**
- **an HTTP interface**
- **etc.**

These templates must be thoroughly debugged and performance tuned. Properly introduced, these will grow in complexity over time to become templates for simple and then more complex applications, from Internet publishing to stock price monitoring to CDR parsing, and so forth.

This approach makes obsolete the object-oriented battle between:

- **compiled code, which is fast**
- **interpreted code, which is more flexible.**

In the code generation model suggested above (Figure 5.2), compiled code delivers high performance without sacrificing flexibility — because the model can be modified and new code generated for insertion into the runtime. Such middleware must, therefore, permit:

- **on-line versioning**
- **automatic instantiation of new code**
- **old versions to fail-over gracefully in a seamless hand-off.**

Being open

The dangers inherent in a closed, proprietary system are widely known, and so is the solution: open systems. However, many people define 'open systems' to mean 'running on three flavors of UNIX'.

To us at Softwire, a more useful definition of open systems is 'interoperability', especially if that in turn means being fundamentally independent of IS fads. Those companies that have bet their systems on DCE or OSI, for example, are incurring huge ongoing costs, both in absolute terms and in foregone opportunities. To us middleware should insulate the customer from fads, not exacerbate them.

Ideally, middleware should, therefore, not be affected by the protocol or type of interface the customer is using. It should act as a backplane, providing reliable connectivity. Companies should then be able to plug interfaces into that backplane, irrespective of whether what they plug in today or tomorrow is a CORBA engine, a SQL engine, an HTTP engine, and so on. Furthermore, this should be achievable without requiring rewrites or upgrades.

Ideal application architectures should support many (interfaces) to one (middleware), and one (middleware) to many (applications, data stores, hardware components, etc.). By maintaining the logic of connectivity (when this event appears, do X activity, Y process or take Z action) in one place, multiple complex applications can be tied together through interfaces without the laborious redevelopment which is required each time something changes in n-to-n component architectures.

For example, in the telecommunications world, such middleware might sit between the network and legacy operational support systems in order to accept events from the central office switches. When a switch event occurs that is associated with a call type 'owned' by the legacy billing system, the middleware passes it to that system just as though it were not mediating at all.

However, when an event appears that is associated with a call type that does not belong to the legacy system (for example, a 'new' quality-of-service based data traffic element) then good open middleware should pass this event to a new modular (probably client/server) system that has been specifically designed to handle this type of event. This delivers the flexibility to be able to change — by addition or subtraction rather than total replacement.

Middleware may also be involved in the ultimate reconciliation between disparate billing systems to aid creation of a single customer bill. In principle, there is no limit to the degree of modularity that this approach can support — a critical advantage for businesses which can no longer accurately predict their systems needs several years in advance. A simplified schematic which shows this type of functionality is provided in Figure 5.3.

Management conclusion

As companies introduce applications to support new services, developers enhance the middleware object model to extend the linkages between old and new systems.

Middleware's role here is to enable companies to add these new application modules quickly without having to rewrite existing systems. Just by modifying the event handling logic in the middleware (using the existing

object model created in the GUI), companies should expect to offer new services in a matter of hours or days — rather than weeks or months.

Ultimately, middleware products are poised to offer one key benefit: systems integration in a soft(ware) box. The alternative — large, expensive teams of programmers — is increasingly less and less viable as companies need to offer their new services more quickly and cheaply.

Monopoly and quasi-monopoly businesses might, in the past, have been able to afford to delay the introduction of new services. In today's increasingly competitive environment, the time delays associated with traditional project implementation are no longer acceptable. Nor can steep development costs be passed on to the end customer.

Middleware is poised to step into this breach but, in order to do so, Charles Brett's rules must be taken to heart and reflected in off-the-shelf products. If that is one central challenge for tomorrow's middleware offerings, then a second is that middleware must be a bridge between what exists (and works — generally known as legacy systems) and what is new and/or forthcoming. If middleware does not satisfy both of these, its utility will be modest.

Real time middleware at PG&E

Douglass Campbell
President
DC Systems

Management introduction

DC Systems is a consultancy and systems house located in Pleasanton, California. It provides software solutions to the electric utility and data acquisition markets — specifically in the context of SCADA (Supervisory Control And Data Acquisition) systems.

In this interview Mr. Campbell discusses the introduction of middleware into Pacific Gas Electric (PG&E) when it redeveloped its SCADA systems in regional control centers (PG&E is the largest utility, by territory size, in the US). The project, which originally began in 1990, turned out to be more complex than PG&E initially realized — because of the multiple needs to:

- ***control sub-stations and electrical distribution and transmission within each independent regional area***
- ***communicate between operational centers***
- ***share data with other systems — legacy and new — within the utility***
- ***make snapshots of real time data available for users on PG&E's corporate Intranet.***

Looking for a turnkey solution

In 1990 PG&E contacted us at DC Systems about leading a project to develop a new SCADA system for them. We began taking a look at all the turnkey solutions available in the marketplace at that time.

PG&E was thinking not only in terms of an Intranet for sharing information (at the time called TCP/IP-based networking). It was also thinking in local area network terms for the systems architecture itself. It wanted a system which could scale in size:

- **from a very small operations center where maybe there might only be one operator**
- **up to larger switching centers where there might be five or six operators at any given time**
- **which would make use of existing embedded controllers.**

There was one additional key element that had to be present in any solution. This was guaranteed delivery of data. Even though TCP/IP is an inherently reliable delivery mechanism, it is not perfect. It does not deal with situations such as:

- **a process going down and restarting**
- **a TCP/IP socket connection which breaks.**

Furthermore, SCADA systems are most needed in adverse conditions — as in storms or other variable network conditions. We had a lot of trouble on the PG&E WAN, plus data coming in from field devices was typically brought in on an exception basis; operations centers only received data that changed.

One consequence of this was if, say, an operator acted to close a circuit breaker — and when the circuit breaker closed the fault still existed and therefore popped right back open — then the traditional data acquisition scheme would generate two events:

- **when the circuit breaker closed**
- **when it opened back up.**

Time tags were applied to those events at the field device — so we would have two events with individual time tags with perhaps as little as a milli-second between them. It was, and continues to be, imperative that nothing is lost as information is distributed:

- **from the field device**
- **to the point of receipt and through subsequent processing.**

That is why some form of guaranteed message delivery was a key requirement. It is this that ensures that events are not lost and thereby that operational problems can be actively managed.

The importance of PG&E's data points

It turned out that addressing the data points issue was critical. In a SCADA system, a data point can be:

- **a single measurement from a transformer**
- **the position of a switch**
- **the state of an alarm system**
- **something similar (but not generated by a computer, rather generating signals upon which a computer or person could take a decision).**

Broadly speaking the bulk of these data points are concentrated in sites spread across PG&E. Each operations center is responsible for monitoring and controlling anywhere from 20 to 100 remote sites depending upon the particular geographical location of the operations center and whether it is:

- **transmission related (handling high voltage transmission); there are relatively few of these sites but many data points at each such site**
- **distribution related — dealing with many more remote sites (often out on power poles or in much smaller sub-stations) but with a smaller number of data points per remote location.**

What was needed was a system able to:

- **bring all the data being generated into a regional operations center**
- **then distribute that data out to those applications which needed it.**

In a traditional architecture for a system like this there would be a data acquisition task which would go out and acquire all the necessary data on a periodic basis. Typically such traditional systems are delivered using:

- **a continuous scanning approach**
- **a lease line phone connection between the operational center and each remote site (or some sort of a radio system) using a low — 1200bps — data rate**
- **a data acquisition task which pro-actively goes out to obtain the remote data and then stores the returned result in a centralized repository**
- **a collection of programs which used the values created by the data point(s) by querying for values from the centralized repository or database.**

Although workable, this is — in many ways — an inefficient scheme as PG&E had realized. For instance, assume that one of the remote processors is a user display program and it displays a sub-set of data points; it might have two dozen data points on the screen. If, every 1-3 seconds, it has to ask for all values from the database, only a very small percentage of those will change during that 1-3 second period. The result is to obtain the same data over and over again.

Enter RTWorks

In order to deliver on a new SCADA system, it was necessary to define an architecture based on a LAN. About the same time that we were looking at turnkey solutions and determining that there really was not anything available to meet PG&E's needs, we came into contact (through other engineering work being done within PG&E) with a company called Talarian Corporation (of Mountain View, CA).

Talarian had a demonstration product called RTWorks (and, within that, a real time middleware messaging approach which is now a separate product called SmartSockets). Although its genesis was in the Hubble space telescope, jet fighter cockpits and financial trading, we realized that the underlying 'middleware' approach was exactly what we needed. It was:

- **real time**
- **easily scaleable**
- **network aware (individual modules could be run on one or distributed across multiple machines, on a LAN or WAN, without requiring modifications to the programs themselves)**
- **able to take care of inter process communications**
- **capable of guaranteed messaging delivery**
- **possessed of something which we didn't really realize the power of (until later), an expert system.**

All this added up to offer a middleware foundation (Figures 6.1 and 6.2) which would allow us to:

- **concentrate on the applications (RTWorks at this point in time has well over 120-130 man years of development in it)**
- **avoid, or at least minimize, the interactions between the multiple constituent platforms that had to be part of our solution**
- **deal with the vast amount, and variety, of field data coming from a large number of geographic areas — each maybe the size of two counties and with upward of 5,000-6,000 data creation points within each two county area.**

Publish and subscribe

Part of what RTWorks brought to the table was the middleware ability for what we term intelligent data distribution, but Talarian prefers to call 'publish and subscribe' (Figure 6.3).

That is, rather than the user display program having to request updates to data values, it simply says that it is interested in a particular subject or particular set of specified data values. Then, whenever one of the specified values changes, that change is automatically propagated from the database program to that user program. In this way:

- **the user interface program is only receiving updates as they occur (unlike the wasteful, refresh-based traditional approach)**
- **there is no time lag in between updates**
- **only the data needed is published.**

Another way to describe this is that the data goes into a real time database first. Once there, it is published to those applications subscribing for that data. The data comes straight from the acquisition device but is only picked up by those subscribers that are interested.

To achieve this RTWorks incorporates what Talarian calls a data acquisition function. In reality this is a misnomer. It really does no data acquisition on its own but instead is a real time database manager which keeps track of:

- each 'arriving' data point
- what subjects each data point belongs to
- whenever a change occurs, that the change is published to that subject
- writing the received changes to the formal historical log (in effect an event data warehouse).

In the PG&E context, what is so neat is that the real time database manager is only concerned with:

- the current value of any particular point
- the subjects which it needs to publish
- the number of processes that are subscribed to any one of those subjects.

If it receives a change for a point for which there are no subscribing processes, no action is needed. In turn this makes efficient use of the bandwidth of the LAN.

The value of the inter process manager

But this is not all that we achieved. Because of the publish and subscribe paradigm, each process has only to be aware of where the interprocess communication server resides — rather than having to connect to all the other processes from which it is interested in receiving data:

- the real time database manager publishes data to the interprocess communication manager
- it only makes a single connection to the interprocess communication server.

It is the interprocess communications manager which knows the IP address of the processes which are connected to it and the subscription details. For example, assume I have a subject called 'Substation A' plus a dozen data points; if the value for one of these changes the real time database manager will simply publish that new value to the interprocess communication server. The interprocess communication server knows, for the five (say) processes subscribing to that subject, the routes and recipient IDs for distributing the information.

With such 'middleware', it becomes easy to distribute the system across a LAN or WAN — because there is really only a single connection about which you need to be concerned. In turn this allowed us to broadcast data.

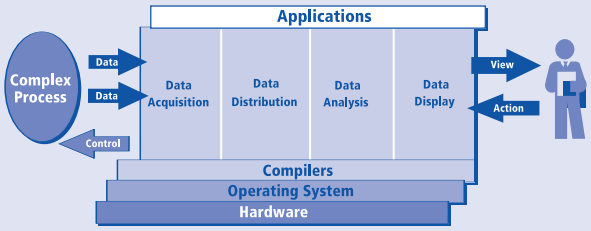


Figure 6.1: RTWorks fundamentals

Figure 6.2: SmartSockets

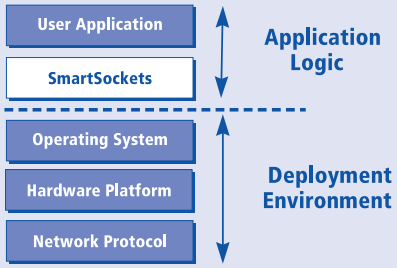
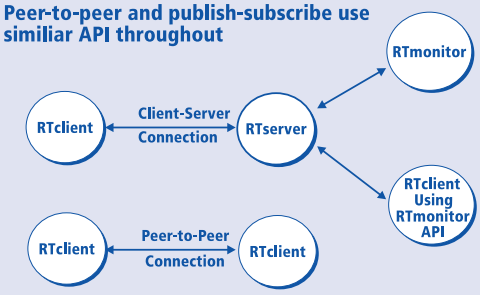


Figure 6.3: Publish and subscribe



When PG&E wants to send data out to a wide number of processes on different processors, it can choose a subject which is subscribed to by either some or all processes. The interprocess communications manager will then complete the distribution. For example, if there is a default subject called '_ **all**', then all we need to do to send something to everybody is simply to publish it to the subscription list '**_all**' and the interprocess communication manager looks after the rest.

Guaranteed message delivery

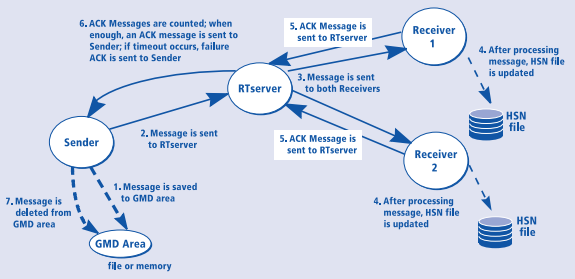
At this point the importance of guaranteed delivery as a middleware service asserts itself. I described why guaranteed messaging (Figures 6.4 and 6.5) is so important in the context of a SCADA system — when a circuit breaker has to be closed but pops open again. In the traditional approach the data scanning process would ask the micro-processor in the field device what had happened since the last scan. What would be received back would be the two events saying:

- **the breaker had closed**
- **the breaker had re-opened.**

What RTWorks enables is the central PG&E location to receive each of the two events, which are individually acknowledged to the field device (which can now forget those events). This is possible because the real time database in RTWorks now owns those pieces of data (once handed over from the field device) and assumes responsibility for them (from publishing to interested subscribers through to writing to the event warehouse records). RTWorks (or, more accurately, the SmartSockets component) guarantees this delivery.

What happens if the instructions to close this circuit breaker are received — and it pops — but the connection line goes down at that moment? In this instance, RTWorks has issued the instruction to close — and then the connection failed. The micro-processor in the sub-station, the field device, still has the events stored in memory with the time tags (it keeps this information until RTWorks acknowledges receipt).

Figure 6.4: Guaranteed message delivery



At the same time, via RTWorks, we have separate logic running in the expert system which keeps track of the fact that a control action (the 'close the breaker' command) has been issued and what should follow. An internal timer is automatically started. If it does not receive the change back at the control point within the specified period of time, it will issue an alarm message indicating a 'control failure'. Now remedial action can be initiated.

What was so attractive about this was that we obtained, with a minimum of development effort:

- **what is effectively transactional reliability**
- **guaranteed messaging from the field device to RTWorks — with ongoing guaranteed messaging once in the SCADA system as each event hit the real time database manager.**

We had a minimum of infrastructure coding to do in order to deliver this. That paid off:

- **in simplifying development**
- **enabling existing field devices to be integrated into the new SCADA system without wholesale replacement of those 'legacy' field devices.**

Middleware should ease the path forwards

All of what we delivered, we could have written ourselves. But that would have meant developing at the sockets and protocols level — which takes time and is expensive.

In the PG&E situation we needed to get a system up and running as quickly as possible. PG&E was running, in some instances, Intel 286 PC-based systems with seven or eight users. As you can imagine these were quite over-taxed and PG&E wanted to move to the new system as quickly as possible.

By using RTWorks we were able to get a new system up and running in under 18 months. In terms of effort there was about three man years of effort — two people for 18 months — who were to develop the basic system and make it operational. What RTWorks allowed us to do was:

- **'forget' the messaging middleware infrastructure — that was already written by Talarian (in what is SmartSockets), with the additional modules which comprise RTWorks**
- **concentrate on writing the layer on top of RTWorks which provided the customized features needed by PG&E's new SCADA system — which also made it look like a traditional SCADA system (where this was needed).**

We actually built the whole system using RTWorks because of the additional facilities/components within RTWorks:

- **a graphical user interface module (used to write the applications which extract the data out of the real time database)**
- **the inference (expert systems) module (keeping track of actions and consequences in addition to other management functions)**
- **the real time database module (as described above, for receiving the data points).**

When we started the GUI was geared towards aerospace type of applications (for which it had originally been developed). The requirements of the utility industry are somewhat different.

We customized the GUI to make it look like that which a utility operator should expect. This was done by using the GUI's drawing editor because not only are the individual data displays built using that, but a template is also built. We built a custom template which had specific buttons tied to functions that were meaningful to utility operators.

We also had to build some of our own data message types on top of those pre-defined data message ones provided by RTWorks. One of the aspects that is different about a utility application (compared to a satellite application) is that in a satellite application you receive a big 'blob' of data or frame of data onto which you put one time stamp for the whole data frame. In a utility application, time stamps are specific to individual data points.

We had to build an object that had:

- **the data point, its value and its time tag**

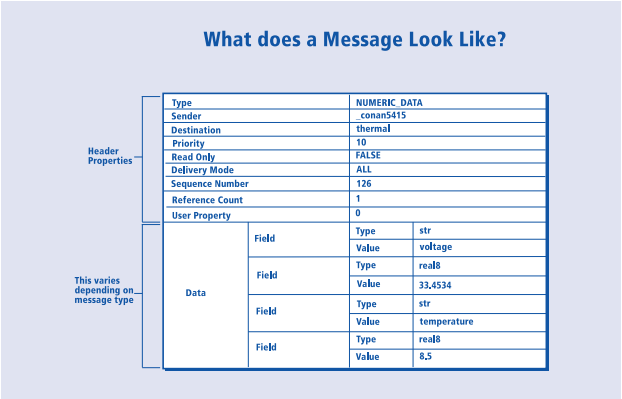


Figure 6.5: Message 'format'

- **state information, data quality flags, etc. — because we had per point information.**

This was not a problem. We built our own RTWorks objects using the tool kit and then encapsulated that in what Talarian refers to as a 'user defined message'. We probably created 50-60 such PG&E 'user defined messages' — for doing different types of communication between processes. By using these we were able to distribute that encapsulated data throughout the system via the SmartSockets middleware.

We also wrote about a dozen or more of our own process (again exploiting SmartSockets' interprocess communications middleware) for:

- **the historical data recorder**
- **the historical playback server**
- **separate file and print managers**
- **an audio program (because any alarm in the system also carries audio information)**
- **the external data server (which is a way for other systems to get data from the real time system without taxing the throughput of the real time system).**

We used each one of these pieces and then ran the new system in parallel with the old system until all the data points were transferred over to the new system. Then PG&E simply removed the old system.

Developing to minimize retraining

There was one other requirement of the new system. This was to minimize the re-training and transition pain for PG&E operators.

Operators at PG&E are not computer people; they are utility operations personnel. PG&E felt that throwing

them directly into a new system would cause problems — especially during the transition period where, with two systems side by side, the possibility of operational errors had to increase.

For the new system we needed to write conversion programs to convert the databases and screens from the old system to the new. There was no overt tool to do this. It needed straight code re-writing of the two existing platform bases, the PC-based one and a VAX-based one.

What proved unexpectedly helpful in RTWorks was the command file driven drawing editor. When we needed to convert existing screens from the old to the new, we were moving from character-based graphics to true vector based graphics. In the conversion process we were able to:

- **scan intelligently through the character graphics**
- **figure out what larger objects they were making (from the smaller character graphics)**
- **create a command file**
- **feed this file into the command line drawing editor which would build the vector graphic displays for us.**

This saved a monumental amount of time. Using this, plus the expert systems module, we were even able to mimic the keyboard and mouse interactions of the two older systems. In turn this minimized the re-training — until the operators had decided they were comfortable with the new interfaces. The inference module was able to provide the various interface 'personalities'.

In a traditional SCADA system this would have been done with hard coded logic in a programming language. But, because the inference module sits so close to the middleware layers within RTWorks — and is real time — we could provide real time response at the interface. In effect the middleware enabled us to:

- **set options in start-up files (to tell which interface the user wanted to use)**
- **make subsequent changes rapidly to the user interface simply by changing the rules and the knowledge base.**

It was not necessary to re-compile the user interface. This resulted in a code base which was both smaller more compact. In turn this made maintenance both more reliable and easier.

Our development approach was vindicated by operators choosing to move quickly into the new mouse-based user interface for 75-80% of functions within only a couple of months.

Connecting SCADA to other parts of PG&E ...

In 1986 PG&E installed a massive central energy management system to cover all of PG&E. That had its own direct communication lines. The regional approach came about after this was implemented — in the late 1980s and into the early 1990s and grew up underneath this energy management system.

In practice the energy management system only needs about 10% of the data that is available overall. But this is the data necessary to make the macro type decisions — allocation of power flows, brokering with other utilities, state estimation, load flows, etc.

In contrast, regional centers were more concerned with traditional SCADA type of activities — remote monitoring and control — rather than the big picture. At PG&E the central and the regional are about to meet. The 51 local operational centers are more or less self standing as regions. In the past there has been no real linkage:

- **between each other**
- **to the central energy management system.**

We are now looking at improving this. Today the new SCADA system — at the regional centers — is operational. The next potential enhancement is inter control center communication and sharing of data — using RTWorks as the middleware base.

This has been aided by additions to RTWorks (while our system has been maturing) for independent systems automatically to communicate with each other. Now multiple RTWorks servers or interprocess communication managers can:

- **communicate directly with each other**
- **share the data with each other.**

We plan to use this to control sharing of data:

- **between operational centers**
- **up to the central energy management systems**
- **externally, to major customers.**

... and to outside customers

One of major business factors which has come into play in the utility industry in the US is the competition

which is being stimulated by energy deregulation. The very large transmission customers of PG&E — the refineries and similar organizations — are able to buy their electricity from anybody they wish, or will soon have the option to do this.

With major customers possessing the ability to buy power from anybody they want, PG&E is looking for value added services to keep those customers. One possible service is to share the real time data that is generated out of the data points and regional centers — but to do this some connection (middleware) is needed. What attracts large customers is that we can do much more for them using what we have.

Take, for example, a Safeway or other large store with (say) a refrigeration plant. Logically PG&E can use the SCADA system to tell if it has dropped a feeder going into that site. Or it (PG&E) could issue an alarm if power consumption dropped on the refrigeration plant thereby indicating a problem. PG&E could then automatically alert the store management which could take remedial action. If the power goes down you can warn customers, so they can do something to save the (say) frozen goods from warming up and being ruined (at huge cost).

We think — as does PG&E — that the middleware enabled infrastructure used for the new SCADA systems can be extended to address both sharing of data within the utility as well as externally with customers. The real time middleware which RTWorks provides is pretty straightforward to put out — and underneath — the applications which exist on other platforms, irrespective of the owner of that platform.

Lessons learned

One of the biggest lessons learned was why it was correct to go with RTWorks. RTWorks allowed us to exploit what PG&E had already. The new SCADA could co-exist with past investments.

In this sense, middleware is truly attractive. It allows developments to proceed without having to hard code an irrevocable future. In addition, using what had already been developed by Talarian, enabled us to focus on SCADA (our expertise area) and avoid or minimize infrastructure development costs.

A second lesson is typical of projects of this magnitude. The project is always bigger than you thought it was, and will inevitably take longer than you thought. While middleware is not going to solve either of these it can assist if it allows you to break the component elements into separate pieces which can be assembled later. If middleware allows one to divide and rule — rather than attempt to build the monolithic monster — it has real value.

I would add to this that we learned a lot about how to use middleware in this the first project we have done that used a middleware component. What we found was a mind set issue.

With middleware you should concentrate on the application rather than worrying about the nuances of middleware-type functions. This opens up incredible freedom and permits a focus on the issue in hand rather than on substrates. It is kind of like being a passenger in a plane. You should not have to worry about the flying once on board; you can just get on board and do some work ... or whatever.

Management conclusion

Middleware — particularly message oriented middleware — is often thought to be the semi-exclusive preserve of asynchronous or decoupled systems. As Mr. Campbell demonstrates at PG&E (and as Allan Lees discusses on page 32 of this Report), the inverse is a truer representation.

However, what is abundantly clear — and common to most real time systems — is that you have to do much more to achieve your results or you have to have the real time infrastructure provided. In the PG&E instance it was SmartSockets and RTWorks which provided the necessary substrate which enabled DC Systems to concentrate on the SCADA issues.

CORBA, DCOM and the future of ORBs

Keith Jones
IBM Consulting Group

Management introduction

In late 1995, 1996 was widely expected to be the year of the Object Request Broker (ORB):

- *relevant standards had been agreed for more than three years*
- *vendors had produced new products that either met or exceeded those standards*
- *yet surveys consistently show that ORB-based applications were and are being deployed in relatively small numbers.*

What went wrong in 1996? What will happen in the future? Has CORBA (the Common Object Request Broker Architecture) passed into an irrecoverable middleware orbit?

State of the art

For several years now the attractions of object oriented programming have motivated vendors to produce a wealth of new tools and attracted users to the concept (but not necessarily the act) of building applications from reusable componentry.

Object analysis, design and modeling tools have complemented compilers, debugging tools and team development environments for nearly all popular platforms. Yet applications built to date using this technology have been focused largely on the client side of client/server systems. Too often they have not reached out to server domains.

Furthermore, while a growing number of object implementations certainly exist on users' systems, they exist as 'object islands'. Something appears to have been missing.

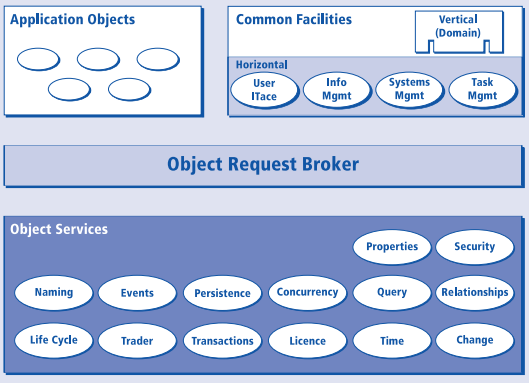
CORBA

The first commercial ORBs were made available in the years following the CORBA 1.1 specification (1992). Lightweight and based on C/C++ programming models, these ORBs flowed object requests between like platforms over TCP/IP transport implementations. The speed at which the CORBA specifications and their initial implementations were developed inevitably lead to proprietary improvisations which precluded:

- **ORBs from inter-operating**
- **objects from being ported from one vendor environment to another.**

These factors inhibited acceptance. With growing support the OMG, therefore, produced an Object Management Architecture (Figure 7.1) combining specifications for an Object Request Broker, associated Object Services and Common Facilities — all in support of distributed object applications. This architecture has become richer in detail since its introduction but has (remarkably) not appreciably changed its basic form during subsequent years.

Figure 7.1: Object Management Architecture



During this period, most of the leaders in OMG produced their own ORB products including:

- **SOM/DSOM from IBM**
- **ObjectBroker from Digital**
- **ORB Plus from HP**
- **Orbix from Iona Technologies**
- **plus a multitude of others.**

An ORB is, however, of little value without substantial Object Services. In addition to basic life-cycle and naming services, real applications require:

- **security of access**
- **persistence of state (data)**
- **transactional integrity**

if they are to be deployed across a network of heterogeneous systems. It was not until December 1994 that the OMG agreed upon both Persistence and Transaction specifications. Even so, the appearance of commercial implementations of these Object Services and others has been relatively slow.

Nevertheless, there was an expectation (with these critical services now 'described') that 1996 would be 'The Year of the ORB'. To understand why this did not turn out as predicted it is necessary to consider:

- **what is an ORB**
- **what are the essential characteristics that should be expected in this new class of middleware**
- **what failed to happen, and what did happen.**

The anatomy of an ORB

The essential elements of the Common Object Request Broker Architecture are shown in Figure 7.2. An Interface Definition Language is the means by which client and server object classes and their interfaces are defined at source level. Currently (although more languages will come) CORBA specifies the mapping from IDL scripts into the following:

- **C/C++**
- **Smalltalk**
- **COBOL**
- **Ada**
- **Java (under consideration).**

Language bindings produced are used to create client object 'stubs' and server object 'skeletons':

- **the stubs enable client objects to request method invocation on server objects using the correct interface details**
- **the skeletons enable server object implementations to be constructed using the defined interface details.**

In addition to static interface bindings, CORBA also supports runtime:

- **discovery of interface signatures for dynamic invocation by clients**
- **recognition of incoming requests by signature on the server side.**

While much more expensive to execute than static bindings, this dynamic support enables CORBA tools development and ORB-to-ORB interoperability. This latter capability was added to CORBA in its most recent modification (2.0) dated August 1995.

In order for a client object to invoke a method on a server object using the Object Request Broker it must have an object reference. Object references are the means by which a client distinguishes one object (instance) from another (instance).

The Object Adapter in CORBA is the component that generates and recognizes object references for a given object instance. This component is also responsible for taking whatever action is necessary to activate/deactivate an object instance. The ORB itself does not understand the structure or content of object references but provides convenience services for handling them. Other ORB services deal with location transparency and marshaling of method parameters as object requests are brokered. To complete the basic picture:

- **the Interface Repository is where interface definitions are stored for dynamic lookup as object requests flow**
- **the Implementation Repository is used dynamically to locate server object implementations in local environments.**

The basic CORBA architecture described is then supplemented by a set of Object Services.

Most agree that this technology has now reached a level of maturity sufficient for real distributed object applications development. That said, the key question remains — why are ORBs not being developed (in any appreciable numbers)?

DCOM, the Microsoft villain

Early in 1996, Microsoft announced plans to deliver its Distributed Component Object Model (DCOM) based on its earlier COM/OLE/Active technologies. The intention was that DCOM would soon be able to construct object applications from reusable 'components' for deployment cross networked Windows NT Server machines. But Microsoft went even further:

- **first it announced that Software AG and other business partners had committed to deliver implementations of DCOM on non-Windows platforms — for example MVS and various flavors of UNIX**
- **in the summer of 1996 it 'offered' ActiveX to the Open Group (successor to X/Open and the Open Software Foundation)**
- **in October 1996 it proposed DCOM for adoption as a new standard for object communications to the Internet Engineering Task Force (IETF)**
- **in December, 1996 DCOM was made available to Windows 95 users in a beta program.**

With such initiatives, Microsoft acted to position DCOM as its backbone for object communications for both the workstation and the enterprise marketplace.

DCOM and CORBA differences

The definition of the Component Object Model made public by Microsoft in 1995 uses much of the same language used by CORBA but with some different attendant meanings. Figure 7.3 shows how COM client objects interact with COM server objects.

As with CORBA, interfaces are defined using a definition language. The Microsoft IDL (MIDL) is an amalgam of (the old OSF) DCE IDL and a new Object Definition Language (ODL). Unfortunately for CORBA protagonists, this is very different to its CORBA equivalent. In addition, MIDL definitions are not used to generate bindings to popular programming languages as occurs with the CORBA IDL. Instead

MIDL definitions are used to create Microsoft's equivalent of the CORBA Interface Repository — called a Type Library — for dynamic reference. To justify this, Microsoft claims that COM defines a language neutral binary standard which in turn 'allows' vendors to map their languages to this new standard in their own creative ways.

COM clients invoke requests upon COM server objects by:

- **nominating an interface**
- **obtaining a reference to that interface from COM services.**

What is important is that the reference returned is a pointer in memory to a table of pointers that locate individual member functions in the nominated interface. This is entirely different to the object reference used by clients in the CORBA architecture because it:

- **does not identify a specific instance of an object**
- **instead provides a pointer to an interface — a class of objects having well-defined behavior (any given client may in fact need several interface pointers on a single COM server to achieve its business purpose).**

The tables of pointers (vtables) are believed to be derived directly from those generated internally by Microsoft's Visual C++ compilers (sic). In the case where both client and COM server objects are local to each other, within a single process, the vtable points directly to interface entry points. The result is that performance should be comparable to C++.

COM interposes levels of indirection with local/remote procedure calls in this pointer reference network when client and server objects are across process or machine boundaries from each other:

- **a client-side proxy is generated to represent a (remote/out-of-process) server object**
- **a server-side stub is generated to represent the client in the remote process.**

The interposed COM proxy and stub components are transparent to application designers and implementors. This provides tremendous flexibility because it allows Microsoft and its host of supporting third party vendors to interpose additional function such as workload balancing, concurrency and transactional control

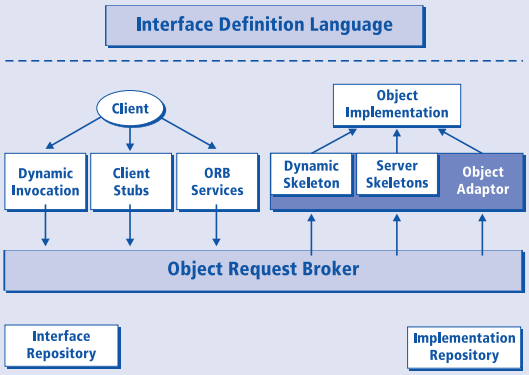


Figure 7.2: CORBA 2.0

Figure 7.3: Microsoft's Component Object Model

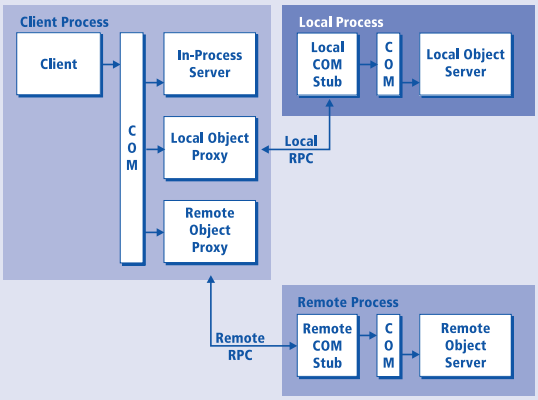
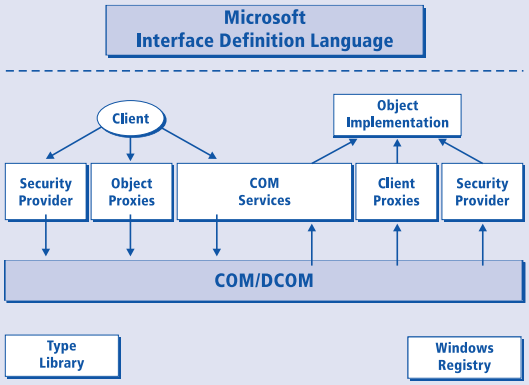


Figure 7.4: Distributed Component Object Model



over COM objects/components. The performance implications, however, of this approach are unknown at this relatively early stage in their evolution.

Figure 7.4 shows the most recently published DCOM architecture in a form easily compared to CORBA. Note that Security plays a prominent role. This is based upon DCE Security — as implemented on platforms supporting DCOM. The Object Request Broker equivalent is strongly based upon the DCE RPC, albeit with Microsoft optimizations for performance and detection of transport outage conditions.

Currently, however, the DCOM equivalent to CORBA Object Services are in the main unspecified. Those that are specified are implemented as COM Services on DCOM supported platforms — such as:

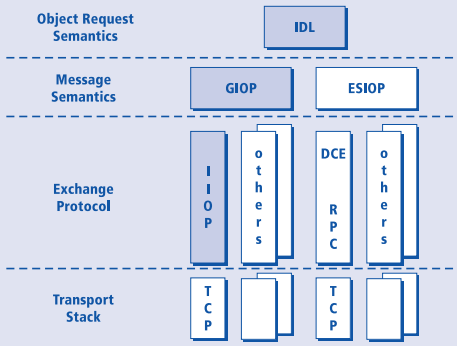
- **persistent storage**
- **universal data transfer**
- **intelligent names.**

The latter (also called Monikers) are the basis for naming and location of structured storage for persistent data and other essential constructs. In practical terms this means that to complete a real business 'object' in a COM scenario, interfaces have to be associated with persistent data by the client using multiple pointers; monikers play a key role in this association.

The role of Object Adapter in CORBA is also subsumed by COM Services which is responsible for locating and launching a DLL or EXE file in the local environment that implements a COM class (defined as a package of closely related interfaces).

The equivalent to a CORBA Implementation Repository in COM is the Windows Registry. Since COM components are often both client and server in real application scenarios it becomes obvious that every platform participating in the supporting DCOM network must have implementations of COM Services and synchronized (Windows) Registries in order to function correctly. This is not true of a CORBA implementation.

Figure 7.5: CORBA Inter-ORB collaboration



Inter-ORB collaboration

Both CORBA and DCOM ORBs were available in 1996 on a selection of popular platforms. Unfortunately neither ORB flavors yet support full interoperability. Building a heterogeneous ORB network has, thus, remained the preserve of extremely ambitious object application implementors.

Instead, the question being asked is when can commercial solutions to this important problem be expected? After all, Microsoft dominates the Windows platforms while others offer ORBs where Microsoft has chosen not to operate.

The work of the OMG in defining the CORBA 2.0 specification has demonstrated how ORBs produced by different vendors can inter-operate in a heterogeneous world (see **MIDDLEWARESPECTRA**, August 1996, page 47). Figure 7.5 shows the main elements of the CORBA 2.0 Inter-ORB Architecture.

The messages which must flow between ORBs adhering to this architecture are once again defined in a language independent IDL and:

- **provide for vendors using different implementation languages**
- **ensure a strong contract between them.**

This is mandatory if inter-operability is to be achieved.

The General Inter-ORB Protocol (GIOP) defines the protocol for exchange of six basic message types used to communicate object requests between one ORB and another. Common Data Representation is also defined for inter-operable data-types and inter-operable object references are defined to ensure collaboration.

In support of the GIOP, the Internet Inter-ORB Protocol (IIOP) is an extremely thin layer of functionality on top of TCP/IP protocols. This layer (sometimes referred to as 'TCP/IP for objects') is designed to be the most basic requirement which ORBs must satisfy if they are to claim compliance with the CORBA 2.0 standard. The architecture, however, allows for other exchange protocols to support the use of GIOP message types over a variety of transport stacks.

In parallel with GIOP the OMG defined a set of Environment Specific Inter-ORB Protocols (ESIOP) to

allow for co-existence with ORBs based on other communication mechanisms. This is significant because it explicitly allows for federation of ORBs based, for example, upon (OSF) DCE Remote Procedure Call technology.

As this facility has been built into CORBA 2.0 specifications and DCOM is based upon (OSF) DCE RPC technology, it is questionable as to why Microsoft has decided to follow an independent course of action. This apparently denies DCOM users of the opportunity to inter-operate with CORBA systems at the ORB Level. But that does not seem to matter much to Microsoft — which may be in danger of repeating others' mistakes.

Growth in messaging

Whilst the adventurous have been building object applications, still more have been realizing the benefits of messaging oriented middleware (MOM). There is a strange parallel here.

Messaging has been seen as the answer to the problem of connecting 'islands' of procedural application code deployed on systems widely separated by either geography, administrative control or architectural design. Messaging systems, like IBM's MQSeries or Talarian's SmartSockets (see page 38), offer a robust channel for messages types to flow from:

- **one system to another**
- **one application to another.**

The question that then raises itself is — why not connect islands of objects with messaging software? If the messaging middleware is in place, why not use it — especially to bridge between CORBA and DCOM?

The honest answer is that, although for many years the OMG consortium of vendor and user corporations has invested some of the best intellectual resources that the industry can muster in the concept and realization of the Common Object Request Broker Architecture (CORBA), it has never made CORBA a pressing issue. While the central tenet of CORBA is that (when constructing distributed application systems using object technology) it must be possible to avoid all together the complexities of communication between islands of implementation on disparate platforms, it (the OMG) has made many of the same 'mistakes' that it accuses Microsoft of making.

For the same reason that 're-use' of (OSF) DCE Remote Procedure Calls and Services was not 'regarded as appropriate' to communications between objects, so the OMG 'set aside' compounded its problems by 'avoiding' MOM APIs and related services. If Microsoft can reasonably be accused of failing to exploit 'what is there', so can the CORBA community be accused of a similar 'myopia'. The result has been confusion. With confusion comes delay and understandable purchasing uncertainty. That certainly reflected itself in the poor volume of ORB successes in 1996, brought about at least as much by the purity concerns of CORBA protagonists as the 'impurity' of Microsoft pragmatists.

Management conclusion

Is 1997 the moment to introduce ORBs? Microsoft's announcement of DCOM in 1996 clearly:

- ***raised the profile of Object Request Brokers for some***
- ***promoted confusion to others***
- ***paralyzed the market.***

While DCOM appears to provide many of the benefits of CORBA, the proprietary nature of its interfaces and their underlying architecture concern those who prefer to implement distributed object applications using a multi-member defined standard. Balancing this, to some degree, is Microsoft's support for a broader ability to exploit languages and approaches which are in popular use today — Visual Basic and PowerBuilder being only the most obvious examples.

CORBA is an architecture backed by many industry vendors. It describes an open architecture for co-existence and inter-operability of object solutions. By comparison, DCOM has only just emerged under the umbrella of Microsoft's recently announced Active technologies, albeit with similar objectives — to promote construction of software from reusable, interchangeable components.

If in the 'Year of the ORB' we expected large numbers of users to be deploying ORB-based applications, then 1996 clearly was not that year. Given the competition between cross-industry ORB standards and proprietary ORB architecture it is hard to believe that 1997 will be the year in which large numbers of new users deploy ORB-based applications — unless the ORB becomes a relatively invisible part of a larger assembly of software in distributed scenarios.

The most interesting question might well be 'whose components/object solutions do I buy and which servers do I deploy?' rather than 'which ORB do I buy?'. Unless the ORB vendors make progress in this area, the already lively 'market' for acquiring DCOM objects may supersede CORBA — with numbers and acceptance (DCOM) winning over elegance (CORBA) — much as TCP/IP won over OSI.

MIDDLEWARESPECTRA is published and distributed worldwide by:

USA and Canada:
Spectrum Reports, Inc.
Subscription Center
PO Box 301368, Escondido
CA 92030, USA
Telephone: 1-800-933-5997
Fax: (619) 432 6560
email: spectrum@interalpha.net

UK and Rest of the World:
Spectrum Reports Limited
Subscription Centre
St Swithun's Gate
Kingsgate Road
Winchester SO23 9QQ
England
Telephone: +44 1962 878333
Fax: +44 1962 878334
email: spectrum@interalpha.net

Editorial Offices:
Spectrum Reports, Inc.
PO Box 301368, Escondido
CA 92030, USA
Telephone: (619) 747 8327
Fax: (619) 432 6560
email: spectrum@interalpha.net

Spectrum Reports Limited
St Swithun's Gate
Kingsgate Road
Winchester SO23 9QQ
England
Telephone: +44 1962 878333
Fax: +44 1962 878334
email: spectrum@interalpha.net

Internet: spectrum@interalpha.net
or World Wide Web at:
http://www.interalpha.net/middlewarespectra/

ISSN 1356-9570