MIDDLEWARESPECTRA
Volume 11, Report 4

Contents — MIDDLEWARESPECTRA — November 1997

Middleware makes Financial Exchanges work

Mats Anderssen
Senior Vice President and Chief Information Officer
OM Group

Management introduction

The OM Group is the derivatives exchange in Sweden. It has two main lines of business:

- *a 'transaction business' which runs exchange and clearing houses, as well as providing facilities management services (a subsidiary provides 25 banks and brokers in Sweden with back-office services)*
- *a technology business which builds, designs, supports and operates the exchange and clearing house technology which is used to run the OM Stockholm Exchange and which has since been adopted by some 10 other Exchanges worldwide, including those in Hong Kong, Sydney and Milan, as well as the American Stock Exchange in New York.*

In this interview, Mats Anderssen — the CIO for the OM Group — discusses how middleware technology (primarily RTR) has been developed and deployed to support financial exchanges where transaction integrity as well as real time processing is required. In addition, he talks about how the OM technology is being adapted for non-financial trading — for example for trading in Norwegian electrical power futures and a similar deployment occurring on the Californian Power Exchange.

Planning for the next generation exchange

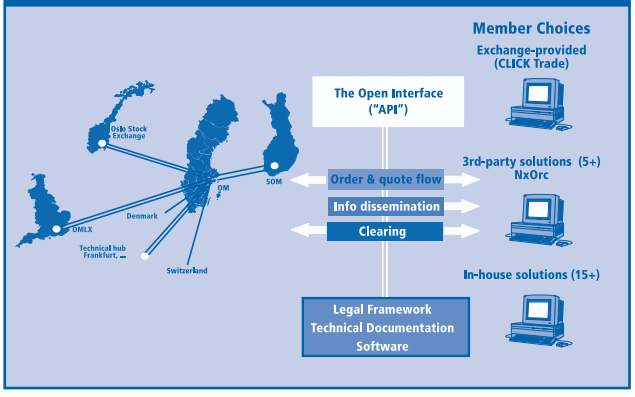
The genesis of our latest generation of technology started in 1989 when we realized that we needed to provide fully electronic trading. Quickly we understood that the demands of such automated matching and clearing required a completely new structure for the future.

Although the exact reasoning is in the past, we chose to use what we believed would be a highly flexible client/server-oriented architecture. We went out to the market to look for an existing software product solution because we understood that our business knowledge domain — where we add the real value — is in providing the know-how about how to build futures and derivative markets and supporting systems rather than middleware per se.

That said, our heritage was in using Digital's VMS, on a homogeneous basis. It made sense to move to OpenVMS as our back end server technology because we already had extensive experience with standard products like RDB — so long as we could meet the key requirements for an Exchange, primarily:

- transaction integrity
- reliability (to 99.9999% or 99.99999% uptime)
- stability
- manageability.

Figure 1.1: Open API structure



In the Exchange business, credibility — as in banking — is all important. Those trading — whether investment houses or individual investors — must possess confidence that their trades and actions will be executed as ordered. Any degree of uncertainty or unavailability can cause tremendous damage in a very short space of time, as well as the loss of transaction income to an Exchange.

What did assist us back in 1989/1990 was that availability, in all senses, had 'only' to be provided for a relatively short period each day — typically a six hour trading day. That trading day is gradually becoming longer. We saw, even then, developments in other markets which pointed towards a need to provide round the clock mission critical services.

For example in California today, where electric power provision is being deregulated, we see a much longer trading day being necessary for the Californian Power Exchange (PX). This is not yet 24x365. But we are edging closer, especially as Exchanges look to link themselves together to provide out-of-hours trading across multiple time zones.

All this mandated that we have a crystal clear strategy for disaster recovery, whether the crisis is:

- damage by fire
- an explosion
- terrorist activities
- simple system failures
- etc.

To achieve this we had to build for full scale and fast failover. When you have to be able to give full recovery for financial trading applications 'running in place', the attraction of using a client/server basis becomes clear. Not least it offered multiple server support.

In addition, we decided to add what we now refer to as an open API (Figures 1.1 and 1.2). With this we aimed to make development easier both for ourselves as well as for the customers of an Exchange — the banks and brokers. Indeed, one of the old dependencies which had hampered us was that these customers often needed our assistance to implement and link our applications to theirs.

By providing an API — with security embedded — on a number of client platforms, these banks and brokers could integrate our trading flows into their internal back office and internal order routing systems without human intervention. By minimizing the dependency on us, we have been able to focus on constantly improving our trading products and services.

Choices

In retrospect we made a number of decisions that were either clever or lucky. Our timing was good because client/server technology was clearly just arriving. We also understood that building our own home-created middleware for reliable transaction serving capabilities was something that would demand man years of our effort and at least as much for testing and proving.

Using these as principles we decided that it was preferable — if at all possible — to buy a product rather than build in-house. Therefore, out we went to see what was available.

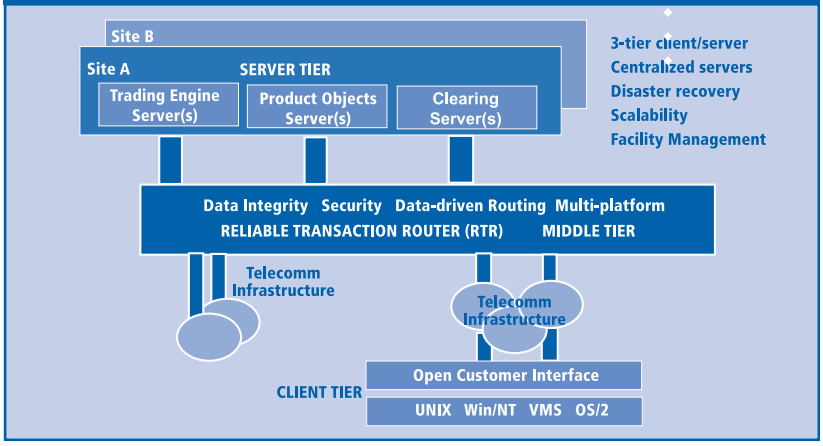
When we looked at the market, Tuxedo was the obvious candidate along with a number of products that barely exist today. Then, almost by accident, we stumbled on Digital's Reliable Transaction Router (RTR) — even though we were already a Digital user (RTR was well hidden, even then). RTR had the attraction that it:

- was transactional
- did not tie closely into a database.

If this last seems odd — particularly in a client/server environment — you must understand that we wanted a clear cut, reliable and secure communication channel between:

- a number of servers (typically 10+, to cover failover, etc.)
- multiples of clients (hundreds, even thousands, spread across many different organizations).

Figure 1.2: Architectural Model



RTR gave, and still gives, us:

- reliability
- availability
- failover

in software at a time when hardware fault tolerance was expensive. This mattered because — if the total solution was costly — then the per transaction cost would need to be expensive which would deter trading activity and diminish underlying liquidity.

RTR, from day one, gave us the capabilities we needed. We could operate as a small Exchange and yet grow tremendously fast without pain. We could run all the Exchange financial products in one server to start with and, when volumes outstripped that server, we could partition the applications and data across multiple servers.

For instance, taking a simplistic example in an equity trading system, equities A to E could be on one server, F to N on a second and the rest could be on a third. The nature of RTR facilitates this — and the associated backup and recovery — because RTR acts as our middleware software system infrastructure.

The database dimension

Now returning to the database dimension, we need high performance transaction throughput. This means that we need to hold large amounts of data in main memory. We are typically seeing order rates of 50-100 order transactions per second.

We keep the order book, which is the main 'data dispenser' in a mainstream trading system, in memory so that we can offer very short turn-around times. As Figure 1.3 shows, on a typical day we have around 250,000 transactions per seven hour trading window with an average response time of around 0.2 of a second. This is measured from:

- the bank or broker client — where our interface receives the original buy or sell order

- London (for example) to Stockholm into the central server where the trade is matched

- Stockholm back to London (again, for example) — when it offers the response (to the interface at the Exchange Member's site) about whether the order closed, was partly filled, became a trade, etc.

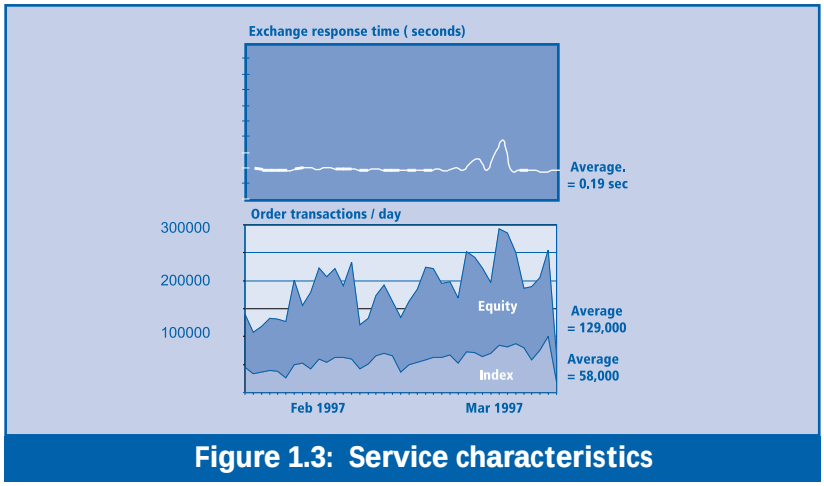


Figure 1.3: Service characteristics

In effect, RTR is our database and it supports a pan-European wide-area network. That indicates why we need RTR's level of sophistication for handling failover as well as multiple sites. If something falls over then we depend on RTR because it can automatically continue from servers located elsewhere (Figure 1.4).

Of course we had already put in place — not least for regulatory reasons — an audit trail log. We could have always re-played from that log, although this would always have been slower and infinitely more cumbersome. Instead, the basic facilities of RTR offer us:

- flexibility, especially across multiple systems (for example, we have data driven routing so transactions can go to different servers depending on content)

- high throughputs and short turn-around times

- connections to a high speed memory database (developed in-house)

- automated failover and recovery

without our having to develop the basic middleware to enable all this. With RTR we could build our applications on top of an existing transactional infrastructure.

Development and deployment issues

We use an object oriented database — on top of Oracle's RDB — to hold the definition of financial products, user characteristics, market rules and regulations. That is used as a source by our main memory, high performance databases which:

- hold the details of the marketplace and make these available for dissemination ...

- provide market data to the traders.

We develop primarily in C and C++ for performance although we have many new clients being developed in Java (and we will shortly have NCs connected to a single client NT server which will run our API into our RTR servers). We expect much of our future to be developed, especially as we move to accommodate browser-based clients without losing our transaction capabilities. We also see that, over time, we will probably introduce more of the server products based on Java.

In this context it is, I think, worth mentioning RTR's location independence, particularly at the transaction level. Our developers can issue a transaction and not worry where it will be processed. In the context of client/server application development, this is a real bonus. Removing the need for developers to have to know which processes work where in the RTR network is a real productivity boost.

In consequence, we have been able to extend transparently the OM Stockholm Exchange with a real time link to our Exchange in London (OMLX). It is to this infrastructure that our client API talks.

Of course as with all client/server implementations, testing was and remains complex — especially with our set up consisting of:

- multiple servers
- hundreds of clients
- thousands of financial instrument and settlement contents.

Throughout we had to be rigorous with all the transaction implications as well as fail over scenarios. The deployment advantage that RTR brought us was:

- we did not have to test RTR itself (as we would have had to do if we had built our own middleware)

- the ability to separate the different pieces enabled us to structure application testing in ways which would probably not have been possible in a host-type solution, although client/server introduces its own complexities.

Over time what has consistently helped us is that we have come to know RTR inside out. We are past the threshold of viewing it as complex. It has become part of our culture and we only wish that it was more widely sponsored by Digital as being one of the best — in our view — middleware products available in the market place.

At the user end

While we use the facilities of RTR extensively, we do not need to present these to our users — the Members of the various Exchanges that use our software. As I mentioned earlier we developed an API which effectively:

- hides much of the RTR complexity

- adds security between client and server (we envelope the data, including authentication)

- offers an interface which is financially-oriented.

This interface is what presents the financial products, prices, trade details, etc. to the user (trader). It is very much application and industry-oriented and it is this that we now support on several platforms, including:

- UNIX (HP-UX, Solaris and AIX)
- Windows NT
- OpenVMS
- etc.

Using this approach we actively try not to lock Members into our choice of platform or screen. In the past it was a terminal into the Exchange. Today, exploiting client/server, Members can:

- choose their platform(s)

- write to our interface

- integrate their selected platforms into their existing and future systems in-house (without our involvement).

In turn, this enables Members to introduce 'straight through processing' — from trade to settlement. This can hugely cut costs because the human element has been reduced to the bare minimum. Therefore, while Members can build their own applications, part of our strategy has been to promote the creation of a portfolio of commercially available applications software which add functions and features above our API.

Currently, there are at least 15 third party-sourced applications and we provide several others, including Research and Trade, a small OM-majority-owned company with products which are popular in the market. It provides highly advanced market maker portfolio analysis workstation software.

Transaction implications

In our environment the transaction is sacrosanct. We must be able to deliver it with full integrity. The way we ensure this from the client is that we always commit — using RTR — at the server. As part of this, the application which issued the original order always obtains a commit status back from the server.

If the transaction commits at the server but the client 'has been lost' (for whatever reason), the originating application will look for the status

when it returns. If a would-be transaction fails on transmission between the client and the server, it has not acquired 'transactional status'. This is only obtained once the server accepts the transaction.

In the transaction flow, the links from our central servers to clients are real time. It has to be this way to maintain context. We monitor RTR all the time. If a trader puts an order in and the client 'disappears' but meanwhile the market moves, we have to act. The trader may want to delete the transaction but we have rules that can be set up so that orders which 'belong' to clients that have disappeared are automatically deleted from the order book.

A further, if background, check comes with our demographics. We tend to have a relatively small number of clients — despite the high number of transactions. In Stockholm we have around 250 clients from about 40 organizations. Similarly, in the various Exchanges using our software — Sydney, Milan, Hong Kong, etc. — we have roughly between 800 and 1000 clients connected (in total) or about one to three screens per organization. This means that it is reasonably straightforward to resolve any problems which do occur.

At the same time, one of the attractions is that we can support any size of organization — from tiny to massive. This is reflected in the way in which we charge Members. We do not use a per-transaction charge. Instead we charge per API — meaning that all our client software has a definable upper limit on the number of transactions per second it can put through. This means that we are charging for potential peak capacity rather than actual use.

This has one specific benefit. It enables us to plan and configure our systems to the peak level which the total of Members want. On the other hand we have to remember that in our type of market we have automated trading behavior — more or less expert systems which are triggered on prices. One buy order can spawn dozens if not hundreds of smaller transactions.

That is not all. Rules can be set up to trigger price changes in related instruments on the buy side or on the sell side of different strike prices — all of which is handled by RTR's transaction facilities.

At yet another transaction level, our interface provides the transaction flow, including the destination, prices and information for clearing and

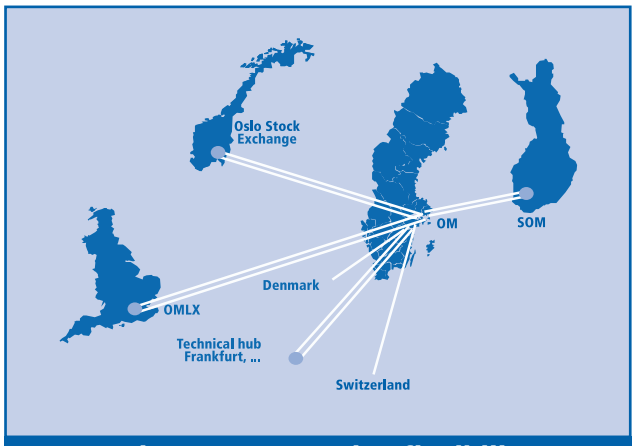


Figure 1.4: Location flexibility

Figure 1.5: Exchange and Clearing

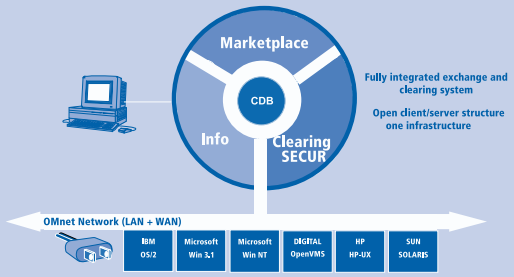


Figure 1.6: Transaction services

	Exchange			Clearing			
	Business Operations	Facilities Management	System Dev't & Support	Business Operations	Facilities Management	System Dev't & Support	
OM	X	X	X	X	X	X	
OMLX	X	X	X	X	X	X	
PULPEX	X	X	X	X	X	X	Started 5/97
NordPool		X	X		X	X	
EL-EX		X	X		X	X	
Oslo SE		X	X				
ÖTOB			X			X	
AMEX			X				
Milan SE			X				
HKFE			X				
ASX			X				Start 10/97
PX			X				Start 1/98
TOTAL	3	6	12	3	5	6	

settlement information. We have integrated the functions of the OM Exchange and clearing house so that the same transaction structure holds the flow:

- from order
- to trade(s)
- through clearing
- to settlement.

This is wholly paperless. It is handled through the same API. What is also often overlooked is that this is more than just software and its documentation. As with other Exchanges this embraces a legal framework covering security, method of operation — and liability.

Multi-location, inter-enterprise capabilities

A dimension which I have not discussed so far is the capability which RTR offers for multi-location — even inter-Exchange — operation. As I have discussed above, RTR gives us automated fallback and failover. Necessarily, to make this workable, we have multiple sites with multiple machines running. This is what provides the speed, resilience, recovery, etc.

If you follow the implications through, this enables us to link to other instances of RTR. For example — although not yet implemented — the various Exchanges using OM Group's software could link all their systems and provide trading between markets. Sydney might be accessible to Stockholm and/or Milan — with a minimum of additional technical effort (the legal and other implications would be more difficult to solve).

In one sense we are already utilizing this. In Stockholm we provide the facilities management for the Oslo Stock Exchange and the Norwegian market. In Norway they have a full terminal and client infrastructure, but the servers are located here — physically distinct from our Swedish market.

In another variant, we are providing the services for the Norwegian, Swedish and the Finnish Electricity Exchanges. Electricity is a commodity, in the same way as oil or pork bellies. In terms of trade processing, electricity is little different from other futures.

Scandinavia is one of the first places where electricity has been fully deregulated. With multiple sellers and buyers, it is possible to establish a fair market. In such circumstances — as either consumer or producer — you can offer prices to create a transparent market.

This is not limited to Northern Europe. It is, for example, also happening in the UK and in the USA. In California, we are building the systems to support the California Power Exchange.

In effect, we are using our financial expertise plus knowledge of middleware and highly reliable infrastructure to create an integrated and automated market for those buying and selling power. We saw the opportunity to deliver an expertise which was unfamiliar to the power industry. We have the know-how to build automated exchanges and to provide the infrastructure to make these markets liquid. But, at the end of the day, the technicalities and procedures are much the same as for a financial market — or even internally within an organization which needs 'internal trading'.

Lessons learned

Early on we decided, as policy, not to mix the middleware structure with data content. We have stuck to this — to our advantage — and have kept the network (what we call OMNet — RTR plus our security solutions) separate from, and ignorant of, data content.

This gives operational flexibility in that we have been able to add servers and applications without requiring code changes in the infrastructure. We have also seen that infrastructure has a longer life span than server applications themselves.

On a less positive note we had to learn the hard way about RTR and its complexity (it is not intuitively simple). That we use almost 100% of the functions and features means we are very much dependent on it as a supported product. If we ever wanted to swap out RTR, this would be technically possible because of the separation of middleware and data/applications. It would also be a major undertaking, for nothing else in the market place — after a feature-by-feature comparison — can provide the same real time transaction processing which RTR gives us.

My last lesson is that we have learned that middleware needs overt management facilities. You cannot leave it alone. You have to know the status of transaction flows, of the middleware itself as well

as the applications and the underlying operating system(s). To us, the middleware management domain should be able to be integrated with traditional network and system management tools.

As a business, we have standardized on BMC's Patrol and have integrated network system management as well as all application events and failures into one colored map/console. RTR does not fit into this because it has yet to be integrated with SNMP alarms. This is a feature we feel users should demand, especially if you are running high exposure — or enterprise — middleware.

Management conclusion

There can be few such high pressure environments as an automated futures and options exchange. Not only is there financial complexity in the extreme (multiple instruments, swaps, derivatives, etc.) but individual transactions can be for many millions of dollars. None can be lost.

At the same time, such markets depend on real time access to information and execution. Without execution there is no liquidity.

As Mr. Anderssen describes, building a dependable and trusted environment to deliver all this is not simple. It needs middleware, especially if it is offered seamlessly, to provide location and process independence with transaction integrity. The OM Group has achieved this — and spread it out to 10+ Exchanges world-wide. That is no mean achievement for a piece of middleware that Digital barely acknowledges.

Does BQM have a role to play?

Charles Brett
President
C3B Consulting

Management introduction

BQM is 'Business Quality Messaging', where the word 'messaging' has strong links to the types of messaging associated with 'email'. Three concepts underpin BQM:

- *the desire to extend electronic mail, which is perceived to be cheap and easy to administer, in order to address critical business processes*
- *the wish to see application-to-application messaging become as easy to manage and use as electronic mail*
- *the assumption that there exists a class of distributed applications, called 'business critical', for which a suitably enhanced electronic mail infrastructure would be an appropriate transport.*

Whereas electronic mail moves mostly unformatted (text) data from person to person, 'message-enabled applications' entail moving formatted data from application to application (or application to person) in business processes which involve a mixture of automated and human actions. Those interested in BQM have suggested a handful of applications which meet a the 'business critical' criteria — such as calendaring, demand chain management and workflow.

This analysis examines BQM. It looks at whether these basic assumptions are sustainable. It also assesses the value of BQM, at least in its current guise.

Origination

BQM is the invention, in April 1997, of Intel, IBM and Microsoft. From analysis of the various original and subsequent statements, it appears that IBM and Microsoft are orienting themselves to supply most of the message transport software infrastructure via:

- IBM's MQSeries
- Microsoft's MSMQ (code-named Falcon).

For example, IBM's Web-site white paper 'Business Quality Messaging with MQWare' describes BQM as "*the convergence of transactional messaging (once and once only assured delivery) with interpersonal messaging technologies, giving organizations a new way to reliably run shrink-wrapped business critical applications on corporate intranets. ... BQM ... provide[s] assured delivery of information for networked applications.*"

BQM has, as an initiative, been pursued through the vehicle of the Electronic Messaging Association (EMA). The EMA was chosen in order to create awareness and educate the electronic messaging community (users and vendors) about issues relevant to this desired convergence of transactional and interpersonal messaging and to foster the introduction of solutions by that community.

It would seem that the three prime BQM movers hope that others will join them in providing solutions for businesses (what they refer to as 'shrink-wrapped business critical applications'). Toward this end, BQM is being presented and promoted through a Special Interest Group (SIG) within EMA which has attracted participants from organizations as diverse as:

- Pfizer
- MCI
- Texaco
- Lockheed Martin
- Boeing
- the U. S. Treasury Department
- Exxon
- several vendors (including AT&T, Hewlett Packard, ISOCOR, the MESA Group and RedBox Technologies)
- Meta Group
- plus the founders, Intel, IBM and Microsoft.

The functional specification

At the September 1997 EMA meeting held in San Jose, the three sponsors presented to the SIG a 'Functional Specification' for BQM. This three page document ('specification' is generous) describes in the most general terms the requirements for a reliable transport infrastructure to assure the secure and exactly-once delivery of messages between applications.

In brief, the document or 'specification' calls for the following:

- connectionless, asynchronous communications between applications — using messaging
- messages consisting of a header used by the infrastructure, plus content not interpreted by the infrastructure
- message properties (carried in the message header) including unique ID, application-defined type, priority, delivery mode (two specified), acknowledgement mode (optional, three specified), length, security and where to send replies and acknowledgements
- queues whose names are unique and location-independent for staging messages — and supporting a list of capabilities (twelve items in the specifications list are related to queues alone)
- assured, exactly-once delivery of messages
- basic support for handling undeliverable messages
- basic support for journaling of messages
- support for integrating message sending and receiving within transactional units of work ('transactional messaging', see below)

- a number of capabilities and 'hooks' to support security
- separation of the application end user from the infrastructure — by enabling any and all installation, system definition, and operation of the infrastructure to be effectively embedded within the application.

Reading the BQM document, it is immediately apparent that it:

- is a specification of the required capabilities of the transport infrastructure
- does not address a specification for the transport itself
- fails to address other issues relevant to the overall problem — for example the application interface.

The transactional messaging connection

Person-to-person electronic mail is effectively 'supervised' by people. If you do not receive an answer to an important message you follow up with a telephone call. In effect, the person is the transaction manager — assuring that whatever is important is actioned.

Business transaction applications lack such personal supervision; their volumes and response time requirements preclude it. Instead, error conditions have to be automated through programming.

Indeed, much of the associated complexity — and dependability — of traditional transaction monitors (CICS, Tuxedo, Top End, etc.) exists to deal rapidly and automatically with error conditions. These high performance utilities are optimized around transactions which fail only a small percentage of the time but which raise havoc in high-volume and/or short-response time situations.

The transactional messaging envisioned in BQM is somewhat similar. BQM is supposed to prevent a message from being lost if a receiving application retrieves it and then just crashes; or from being sent twice by the application that originated a message, crashes and sends the message again upon being restarted.

Generally speaking, in the BQM view of transaction systems, the 'BQM' middleware co-operates to move a message reliably in three steps (Figure 2.1):

- the sender puts the message on a queue within a transaction scope (so the message becomes visible only if the transaction completes successfully)
- the message middleware moves the message from queue to queue (as an ACID transaction in its own right)
- the receiver reads the message and removes it from the queue within the scope of a third transaction (the message is actually removed from the queue only if the whole transaction is successful).

The EMA SIG has put forward what it perceives to be a fairly elaborate functional 'specification' to satisfy this. That specification, however, undeniably reflects the reality that two competing middleware vendors already possess existing products that meet the 'agreed' requirements.

MSMQ and IBM's MQWare (an NT-only product derived from MQSeries and simplified for situations like BQM) both use queues and transactional messaging reliably to deliver messages from queue to queue without involving special logic within the applications at each end. In this sense, both meet the SIG's objectives. However, the substantive issue is — does this go far enough?

Yet the problem is real

There is no doubt that what the BQM initiative wishes to address is a business problem. On this basis, BQM at a high (stratospheric) level is real:

- already the Internet provides a highly desirable global network infrastructure that businesses would dearly love to exploit

- some users are, today, already doing considerable work to integrate people-intensive steps with automated steps in an overall business process — by moving information through one or more workflows (and often involving multiple companies)

- others are 'front-ending' transactional applications with substantial workflows — for example, integrating an existing airline reservation system with a system that brings both travel agents and corporate travel departments together in complex workflows that conform to each corporate customer's travel policy.

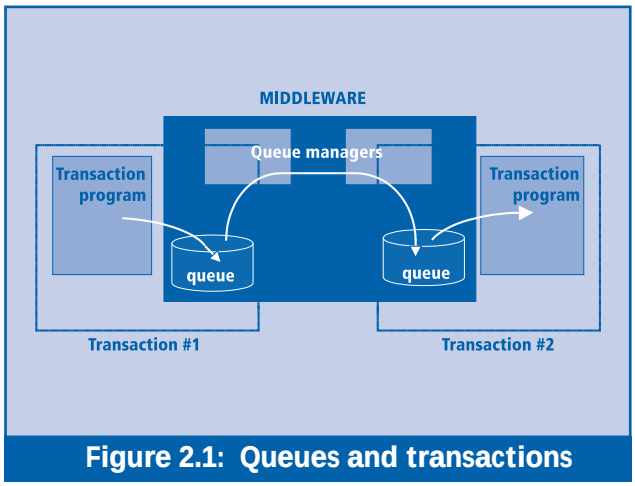


Figure 2.1: Queues and transactions

All three of these are considered to be 'business critical' but are not truly mission critical — even though the front end processes soon start to look as demanding as a traditional transactional back end. In addition to these, however, there are other applications which can more readily be seen as 'business-critical' — most often because they involve no transactions or management of assets:

- one typical example can be found in a pharmaceutical company which must periodically distribute updates to clinical trial instructions

- another similar example is a manufacturer that must occasionally send recall or product update notices to its customers

- a third is internal calendering.

Figure 2.2: Subsets and supersets

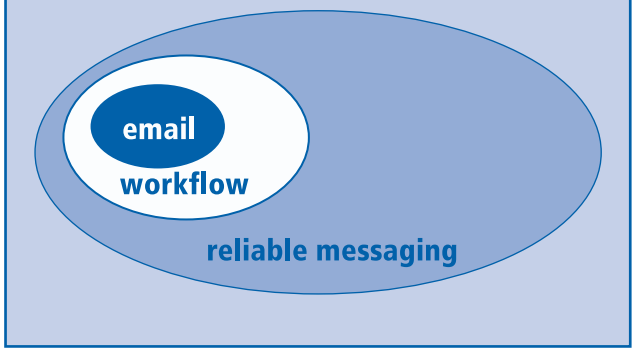
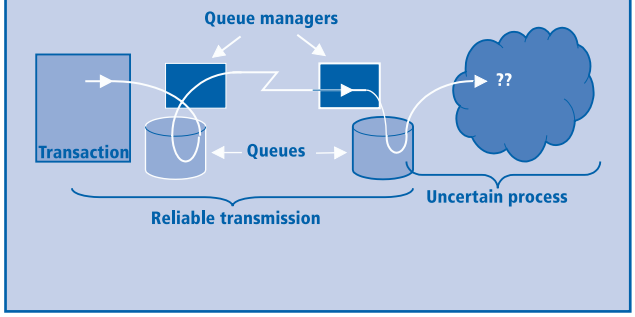


Figure 2.3: Ensure business processes



The common element about each of these is that they need need reliable, authenticated and non-repudiable ('business quality') delivery of communication between (as well as within) organizations. Furthermore, with many initiatives increasingly being Intranet or Internet based, ready-made opportunities exist for message passing — because the base infrastructure exists on which messaging could easily be deployed and managed. From a high level perspective, if users are going to mix automated steps with people-driven steps in a

business process, the process as a whole logically should be supported by something adequate for the most demanding part.

If you accept these principles, it seems reasonable for the SIG to argue that:

- the BQM concept, of a business critical workflow, should indeed rest on reliable messaging middleware
- electronic mail — wherever it is involved in formal business processes — should be implemented as a special (simple) form of workflow (Figure 2.2).

This said, it is not obvious that the SIG has really stated the problems either clearly or fully. Indeed, one persistent concern is that the three original proponents are more focused on defining their own opportunities than on addressing the user requirement. Supplementing this feeling is the paucity of documentation and description available — which manifestly does not address the basic issue of whether, or how, application-to-application messaging can ever be as simple as the email model which is the apparent objective.

Clouding the issues still more is the realization that BQM does not claim to be a substitute for the high end message queuing role which both IBM and Microsoft see organizations needing. These two vendors seem intent on distinguishing BQM from mission critical, high function middleware (where both have a presence already) which supports data:

- capture
- transformation
- movement
- routing
- publication
- tracking
- management
- archiving
- analysis
- disposal.

The BQM problem description

As has probably become apparent by now, the problem description lying behind BQM is confusing. It is ambiguous, plus detail is lacking. And, as if to complicate matters, BQM is (arguably) too like TQM (as in Total Quality Management) which only submerges any substance still more.

Phrases such as "*the convergence of transactional messaging (once and once only assured delivery) with interpersonal messaging technologies*" are constantly frustrating — especially when:

- transactional messaging (as BQM describes it) is a technology but interpersonal messaging is a class of application
- convergence is not what BQM is about.

It would appear that mission critical transactional messaging and email are designated as separate categories so BQM can apparently occupy a vacant middle ground. What would have been far preferable would have been a claim like 'BQM is the application of certain robust characteristics of transactional messaging to improve the operational viability of mail-enabled applications'. This would have been accurate, precise and clear.

The failure to define the problem carefully (which should not have been terribly difficult), means the current functional specification does not look like the answer (or an answer) to most users' business problems. Instead the offering to the EMA smacks more of an initiative seeking a problem.

Instead, the specification goes deeply into implementation, almost as if it is trying to make it a rule that MQSeries or MSMQ are mandated. Similarly, the initiative demands shrink-wrapped applications on top of a layer of network infrastructure (middleware) as if to satisfy Intel's desire for a horizontally layered market as well as to assure a market for Intel (or Intel-based) servers. Surely it was not necessary to convey these as lasting impressions of BQM is (or should be).

In short, the picture left of BQM is that the original proponents seem less interested in the total problem — what it is and how to solve it — than in being part of the solution for whatever the business problem really is. This is unproductive, especially when a real problem exists — one that is substantially larger than that currently being discussed by the BQM SIG.

The larger problem

It does not take much analysis to see that the problem which BQM broadly pointed to is much bigger than what the SIG is discussing. BQM should be about:

- eliminating transmission failure
- handling any failures in an automated way that ensures data and process integrity
- making this as accessible as email (rather than as inaccessible as traditional transaction processing or even message and queuing)
- ensuring middleware independence (all middleware should be as replaceable as any other hardware or software, as argued in past **MIDDLEWARE-SPECTRA** Reports).

Automation of reliable and predictable handling of errors is vitally important. Measured on this basis, the EMA's SIG should be concerned (it does not seem to be) about failure modes beyond the scope of mere message transmission.

As Figure 2.3 shows, a message could — in principle — be delivered with extreme care to a part of a business process that is not able to take similar care in terms of getting its task(s) accomplished and continuing the process. An EDI transmission, by contrast, typically runs in a 'production' environment and is under rigid control and audit as it:

- is assembled
- leaves the sending organization
- arrives and leaves a service provider running a Value-Added Network (VAN)
- arrives and is subsequently processed at the receiving organization.

In other words, transactional messaging is meaningless if the applications at either (or both) ends do not themselves have a notion of a transaction (a series of actions that must either all be completed or none completed). Transactional messaging means that not only is the message moved reliably but also any required actions requested or implied in the message must be performed as well.

In non-transactional environments, it is equally reasonable to assume that the sending or receiving application (or both) should take as much care to perform the business process step as the transport infrastructure takes in delivering the message. Thus, the truly interesting problem (to users, but perhaps not to the BQM proponents) is:

- not how to transport messages between queues
- but how to design and provide readily usable and reliable functionality at the source and destination, and perhaps elsewhere, within the infrastructure.

This larger problem encompasses supporting the receiving site, including helping it to recover from any of its own errors in handling messages received. In effect, the BQM initiative needs to follow up its initial thinking with a substantive explanation of what the scheme expects or requires of application vendors (and of users) — if it is to contribute ....

Simplicity

The other area of deficiency concerns simplicity. IBM has demonstrated its interest in simplicity by producing MQWare. That is a significant step forwards when compared to the many complexities found in a full-blown MQSeries implementation (and it remains to be seen how simple MSMQ is in practice).

Despite initial good intentions, the BQM document all but ignores the issue of simplicity, except in the context of handling of messages in transit. This is inadequate. Messages traveling over a connectionless, asynchronous messaging infrastructure do not simply appear in a recipient's in-basket the instant they disappear from a sender's out-basket. An asynchronous messaging system — using queues to buffer messages when a receiving application is not running and whose messages individually have lots of attributes and may be running time-critical applications — demands onerous management, with multiple tasks and obligations.

As if to underline the relative omission of simplicity, there is the related issue of management simplicity. Managers of electronic mail systems are, of course, accustomed to working with in-transit mail. The concepts described are, therefore, not completely strange.

Yet the BQM documentation makes no mention of a management interface — only that the end user should not need to perform operational functions ('operation of the BQM provider'). This is another apparent deficiency that the BQM initiative must correct if it is to be taken seriously.

Was involving the EMA a positive step?

The choice of the EMA, at least at first glance, as a launching pad is curious. It looks almost as if an audience — any audience — which knew of messaging would do.

Irrespective of this, the BQM effort has identified a real problem area. A 'functional specification' now exists which establishes a capability level in terms of reliable delivery without involving users or applications. It is aimed at an organization (the EMA) with members who should be able to understand — and care about — the underlying issues. If interested, these members can work to evolve more complete requirements for themselves.

Through the 'specification', the initiating vendors have presented ideas on how the problem of reliable delivery should be addressed. Their attitude seems to have been that this was needed in order to educate the industry and users (which may have been correct). What is now significant is whether the EMA's membership will seize the initiative and drive BQM where users need it to go.

Unfortunately, even if the EMA takes notice, what those members should be demanding goes well beyond the implicit compromise which Intel, IBM and Microsoft have agreed. The danger is that BQM will go the way of the Message Oriented Middleware Association or the Open Application Group — founded by groups of vendors with little in common other than a collective desire to sell something to someone.

If the EMA SIG really takes hold of BQM it will need to define the full extent of the problem area, and only accept the initial BQM specification as an initial proof that an acceptable mechanism can be built. It needs to address the larger problem, not necessarily the one proposed by IBM, Microsoft and Intel.

Amongst other elements, it would produce its own API or other standard. But even this would (likely):

- be at odds with the founding proponents of BQM
- almost surely never be implemented (it is rare that the business of users is to write utility software for sale)
- open the door to additional competitors.

Not all is lost

In recent weeks two different announcements have heralded possible short term progress:

- Oracle 8 with its Advanced Queuing (AQ)
- Candle Corporation's Roma.

Oracle included AQ within the latest version of its database as a facility not only for staging or buffering messages (traditional message and queuing) but also to support other functions which might be needed to support a reliable business process, for example:

- recording an audit trail
- tracking messages
- archiving them
- recovering from business process interruptions.

Such a queue facility, placed in a database which is already run in a secure production environment and exploiting existing skills, could make sense for other purposes as well — including many of the areas being 'covered' by the BQM SIG. Oracle may well have opened up one alternative path forwards.

Addressing broadly the same basic problem, but in a different way, Candle's Roma initiative includes a software layer that developers can use in place of the APIs provided by either MQSeries or MSMQ. With Roma, application developers do not need to choose between MSMQ or MQSeries. They can

write to the Candle messaging API knowing that the choice of MSMQ, or MQSeries, as the underlying transport is irrelevant.

If additions are made to Roma (security, for example), applications developers may well decide that the flexibility of choice offered — where you deal with only one API and where multiple platforms are supported — is more attractive as a product than what is still only a discussion within the EMA.

Management conclusion

The BQM initiative touches several important issues — but too lightly. It raises numerous questions, including:

- *can middleware vendors provide a messaging infrastructure that can be deployed easily?*
- *what is the full extent of automation support needed to support global business processes (see also pages 30 and 46)?*
- *can application-to-application messaging and electronic mail be used together in large business processes involving both applications and people-intensive processes?*
- *should users deploy both a more robust infrastructure for electronic mail and extensions thereto, as well as a powerful message brokering plus a transport infrastructure for production applications (and if not, what should this strategy be)?*

Integration of people and applications in reliable business processes is a real and general problem. BQM is barely a first stab at addressing this problem area. Furthermore, because underlying need is so understated, there is lots of confusion in the BQM announcements and 'specification'. The desire to avoid proponent conflict is all too apparent.

Being a vendor-sponsored activity supplemented with user involvement, BQM runs the risk of becoming another talking shop with good intentions and minimal output but paper. It will need to take a tougher user stance and acquire teeth if it is to deliver anything worthwhile.

Is the BQM initiative a worthy effort? That it has touched a significant problem is reasonably clear. On the other hand, the current effort is not nearly sufficient. What has been done so far is only a beginning and — if not soon supplemented — will probably be too modest to warrant further user attention.

Deploying messaging 'in extremis'

Eddie Hall
IS Planning and Development Manager
British Gas Services

Management introduction

Eddie Hall is the IS Planning and Development Manager for British Gas Services which is part of the Centrica Group, the holding company spun-out from British Gas plc. The Company, which installs, repairs and services domestic central heating systems and other gas appliances, includes:

- *British Gas Services Ltd.*
- *British Gas Trading Ltd.*
- *Energy Centres Ltd.*
- *Accord Energy Ltd.*

Mr. Hall joined British Gas Services in 1994 (from another part of British Gas) to oversee the creation of new IT systems at a time when the new company was having to assimilate systems inherited from other parts of the original British Gas, as well as to create a systems strategy independent of other parts of British Gas. In this interview he discusses:

- *the issues he had to confront in planning systems for a newly demerged company which was inheriting a mix of systems developed by others*
- *how he used middleware, in particular VCOM queuing, to solve application integration problems*
- *why British Gas Services took the decisions it made.*

Different strategies and systems

When I joined British Gas Services, each of British Gas's four business units had a share of the twelve regional gas computer systems. As you can imagine this was a complex environment in which to work.

Adding to this, the twelve regional systems were based either on ICL or IBM (with one exception, which had an ICL back end and a Digital front end architecture). Two years earlier in the old British Gas, the strategic decision had been made to join all the IS organizations in the Company together to use one overall systems structure (based on ICL). We had been moving towards this for 12-18 months. We were in a period of great change.

No time

In April 1994, when I joined British Gas Services, I had been with British Gas for 20 years. My job was to come up with a systems approach which would move us forward and which was based on a business strategy that was most vividly painted in two 'day-in- the-life' stories (which would now be considered politically incorrect):

- Sid, a service engineer, who would be field based, would not report to an office and would receive information and respond electronically from handheld devices

- Sarah, who was office based, dealt with customer calls and administration — plus provided both the front office and back office support.

At this point my knowledge of British Gas served well. Amongst the activities that the Company had been good at were data analysis, process analysis and modeling activities. All this was at our disposal as we grappled with developing a vision of what the British Gas Services systems infrastructure would need to look like.

Our first step was to look at what options existed:

- one option was to take the existing portfolio and expand or enhance this to satisfy the integration requirements as well as electronic data transfer and document transfer

- another option was to start completely from scratch — especially as we already possessed good models of our various business activities as well as development tools like TI's IEF

- a third was to buy-in 'best-of-breed' applications, off the shelf, and make these interwork.

Irrespective of the technical considerations, one particular pressure drove us. This was the timescale given to me. Within two years we had to have the entire British Gas Services systems up, running, implemented and rolled out.

In this context, when we looked at the options, the only one that seemed to meet the timescales was the third option — to buy anew. Yet I did not have any experience of buying in such package systems. My career had been mostly about building applications internally within the regions.

To cut the story short, we decided to approach several systems integrators for assistance. In the end IBM and Digital joined together with ourselves to meet the objectives.

Besides the terms and conditions associated with the agreement, we made it crystal clear that solutions which needed only IBM or Digital products were not acceptable. Their contribution was to help us pick the best solutions and supporting technical architecture.

On the 31st July, 1994 we presented our strategy. The functional areas by which we proposed to support the business were rapidly agreed.

Then we discussed our view that what we needed was to implement a technical architecture which gave us the largest selection of packages from the marketplace. We suggested:

- UNIX for the applications

- Oracle on MVS for the main data server

- a variety of LAN, WAN, telephony, radio, mobile, etc. for communications.

The use of MVS/Oracle surprises some. Our reasoning was partly that the size of our intended database demanded such an approach and partly that a single image would enable a greater degree of integration which in turn would permit greater flexibility with geographic distribution of offices and people. In addition, use of a very large data server at the back end made sense because we could use an existing mainframe in the Mitcham Data Centre which was already running Oracle.

All this was presented to the Board with all the risks and estimates of times and costs (around $100M over several years). This was agreed. Now all we had to do was deliver what we had described.

Making progress

In selecting applications, the depth of prior British Gas modeling of data and activities greatly assisted us. In effect we had a formal description of the functions which we needed to support. Between ourselves, IBM and Digital, we built a blueprint of:

- the business functionality required
- a description of the interfaces we would need to flow across the functions
- the sorts of relationships that we would need between packages (even down to where we would need real time or pseudo real time integration or could wait overnight or over a weekend, etc.).

Once this was in reasonable shape (it was constantly being refined) it emerged that both IBM and Digital possessed excellent databases of packages which were searchable by key factors. We used these to work through the choices that were available in the areas in which we were interested — and there were 15 of these.

We created a gap analysis process and went out to tender to a maximum of 7 vendors for each of the 15 areas. In the event we actually did the high level gap analysis, on average, about five times per product — all in parallel because it was not conceivable to implement one package without delivering the others at the same time.

At the same time I established a small integration group to consider how we would fill in the middle part — the integration of the various packages. This 'middleware arena' was where we had our greatest uncertainties — both technically and about costs. But we had to make some decisions or our strategy would have no chance of delivering.

The first job, therefore, of this integration team was to define:

- the specific ways in which each of the systems areas would talk to each other
- what mechanisms were available
- the volumes involved
- the levels of integration required.

Having considered the above, we decided to use standard middleware products — and several of these, capable of working in parallel — as our primary integration mechanisms:

- Oracle View
- Oracle SnapShot (so that the users could see data at specific moments in time)
- Oracle Shared Tables (so that our SAP financial applications could retrieve transaction data from a series of shared tables held on the MVS/Oracle mainframe)
- plus a messaging product.

The need for messaging

We knew that we needed messaging if only because of:

- the extreme level of integration we had identified
- the need to be able to keep systems separate (but working together)
- a desire for flexibility (we had to have the ability to pull something out, discard it and plug something else in).

By February 1995 we had selected everything, including the MVS/Oracle storage system. At this stage we had virtually decided that IBM's MQSeries was going to be our messaging solution.

But, when we selected the stock control system, based on Digital's VMS, we had to face the fact that MQSeries did not support VMS. We had to go back to the drawing board. VCOM was chosen because it was available on MVS, AIX and Solaris as well as VMS.

One aspect of our implementation philosophy was soundly proven here. Middleware should be as replaceable as any other part of the system. When VCOM displaced MQSeries, we demonstrated this.

Using the messaging middleware

To understand how we use messaging (Figure 3.1), let me explain our systems. We have:

- MVS with its primary Oracle database
- multiple application servers which run dedicated applications on UNIX (either Sun with Solaris or RS/6000 with AIX)
- an FDDI via Interlink connection from the application servers to the MVS/Oracle server (which, as discussed above, is doing all the SQL data serving).

Users connect to the application servers by LANs, WANs or radio. In so doing, several forms of middleware are used, including:

- VCOM messaging for the connections between our Work Management Systems (WMIS), UCAD (our in-day work issue system), SCMS (the contract and the customer system) as well as the logistics systems
- SnapShot for providing views to any system/user that needs them.

We use VCOM to:

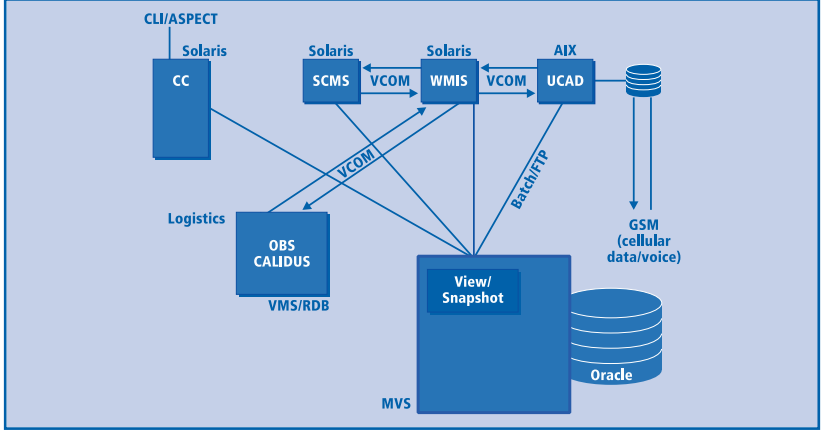
- tell each system that a job has arrived to be done or that a job has been completed, etc.
- refresh data overnight — on a daily upload/download basis.

The nightly downloads are updated during the day using VCOM messaging — as events occur. Thus it is VCOM which bring us our near real time messaging.

It enables us to avoid 'holds' where the updating of one system is dependent upon communications or availability of another system. Instead, should a system or comms. link be unavailable, we make use of VCOM's transparent queuing on the sending system — until the recipient system is available. Then the queues automatically clear down, message by message.

We have had this working for over a year, since May 1996. Basically the messaging runs continuously in background but is suspended during batch processing. VCOM is restarted as soon as the batch finishes and the target resource or application once again becomes available. If there has been a build up of messages (and we can watch the queues on the Web), it is amazing to see how fast VCOM drains those queues as it passes on the accumulated work.

Figure 3.1: VCOM links



Operational experiences

Looking back, our initial implementation expectations were somewhat simplistic. I think we got our original design wrong. We did not really know enough about messaging products as a whole to understand at the nitty gritty level how they work.

What we discovered was that considerations involving management and distribution play a much greater role than we expected. These challenges arose with issues like the way in which messaging products can:

- clear queues
- produce bottlenecks
- consume processing power
- create new nodes and queues.

Let me give you some practical examples. Our original, over-simplified, design meant our queue lengths were often too long. The result was that this encouraged messages to stack up. With shorter queues we have mostly cured this.

Another example of over-simplification occurred when we originally implemented the Utility Computer Aided Dispatch System (UCAD). We put it on four AIX machines with one queue with the one RS/6000 sharing this out. In the light of experience, we have implemented VCOM with eight outbound/inbound queues spread across all the machines — which is far more efficient and faster.

But it is not just about spreading the load. It is also about spreading the volume of updates in a coherent pattern.

For each of the seven local Area Service Centres (ASCs) we have the equivalent of what we call a single UCAD. We also have two UCADs running on a single RS/6000 and another five ASCs run on clustered RS/6000s using high availability software. Because our customer records are divided geographically, we can optimize and still cope when we have adverse conditions as happened last winter when Scotland was hit by a major storm.

At that time, call levels in the Uddingston Area Service Centre skyrocketed. If our old approach had been in place, the messages from that one ASC alone would have produced a bottleneck for all the other ASCs. By spreading the volume, the other ASCs kept operating as normal and, equally, did not hold back the ASC supporting Scotland.

Let me offer a third example — which occurs if a VCOM queue is waiting for an online user to release a record. If this happens, VCOM just sits and waits until that record is free. No other messages are processed from that queue. With too few queues, this is quite likely to occur.

Now that we are becoming familiar with the practical operation we are also becoming more sophisticated. We are breaking up the types of records so that — in the depths of winter when we are busiest — we can use the prioritization of job classes to ensure the jobs that need to be done are processed first. Using defined message types we can (for example):

- have the 30% of the transaction workflow that is in a priority category processed immediately
- hold the remaining 70% for subsequent or overnight processing.

This triage is done through VCOM.

Another trick we have acquired is to use queues as a practical predictor of activity. For example, at 4pm, supervisors can look at their UCAD screens to see what workload remains that day, before saying to engineers that they can finish after their last job. But looking at UCAD may hide the fact that there are yet to be processed jobs in the VCOM queues. By allowing the supervisors to see the relevant queues (via a browser over our Intranet), they avoid being caught out should 100 jobs be about to arrive at (say) 4.10pm.

A last trick — negotiated with our users — is that we can, at peak periods, shut down applications for half an hour. This can have miraculous effects. Where we might be having major queue application delays (and therefore resource deployment constraints), as in winter, shutting down the applications for just that half hour means that we can clear the processing backlog and queues. We have this working. It is amazing to watch how fast VCOM drains its queues. We then restart the applications and we have more than recovered the half hour shut down.

Today we are running at about 400,000 messages per day between some 18 queues. This will probably expand as we learn more. In implementing, what we have found is that there are better gains to be obtained from adding more queues rather than trying to use a few big queues.

Also we have some work still to do on re-balancing loads, because of other factors, like system occupancy and application performance. Solaris makes it easy to move applications and systems resources. Moving the associated middleware — whether when using VCOM or something else — is significantly more complex.

Lessons learned

I think the first lesson is that you need people who understand the practicalities of messaging as opposed to people who understand the theory and technology of messaging. Second, if you want messaging to work, you must obtain information or estimates (in advance) about volumes and timings.

A third lesson is that I would not worry about mixing the middleware or messaging tools. Initially, for example, we were reluctant to have both VCOM and MQSeries. But each has its strengths and there are at least one or two places where MQSeries would have been more suitable than VCOM. But we initially decided that if we were going for one messaging solution we might as well go whole-heartedly for VCOM.

My last lesson learned is undoubtedly the most important. It is that messaging does work and does allow you to separate systems yet still link them together in real time. The British Gas Services VCOM implementation was my first experience of messaging (my only other experience was in CICS and some LU6.2 activities between MVS and 8100s). What I have learned is — do not forget, or underestimate, the importance of understanding in advance your likely system and application interrelationships.

Management conclusion

When Mr. Hall arrived at British Gas Services he faced everything — from pieces of other people's systems based on at least two different mainframe platforms to being caught in the middle of an effort to standardize the whole of British Gas Services on one enterprise-wide approach.

Presented with an all-too-close deadline for introducing systems to British Gas Services, he chose to buy-in packaged applications and connect these with middleware. Yet, by his own admission, the initial understanding of middleware — and messaging in particular — was imperfect.

Nevertheless a mix of VCOM (and other middleware) was introduced. Perhaps miraculously, this succeeded. Today the different systems inter-work — and messaging middleware has been shown to do what was required, if not quite as expected.

Why PC servers will not overtake mainframe servers anytime soon

Wayne Duquaine
Principal
Grandview DB/DC Systems

Management introduction

In recent years a common argument has been that PCs will become the servers for all. The PC market now seems to believe that it:

- *has reached that august status*
- *will quickly become the 'enterprise server' of choice.*

In this analysis, Wayne Duquaine discusses why current PC server hardware and operating systems are still years away from kicking mainframes out of their Fortune 1000 server role. He examines the evolving role of servers in the context of what is — and should be — expected.

The relevance of this is that middleware is oriented primarily to solving the distribution of processing. This is needed because there are many systems — composed primarily of PC servers and UNIX machines. However, if these lack certain features, it is arguable that central host processing still has a role to play — which could even obviate some forms of middleware.

Setting the scene

Four years ago, conventional wisdom was that UNIX was coming on strong, that UNIX had legitimized itself as the new server and that large numbers of Fortune 1000 corporations were installing UNIX servers rather than upgrading their dinosaur-like mainframes. On this evidence, it seemed clear that:

- mainframes were doomed
- rapid advances in UNIX-based hardware and operating systems, databases and transaction processors would quickly overtake whatever niches the mainframe world still owned.

More recently, conventional wisdom has evolved to a point where PCs — using Windows NT Server — are coming on equally strong and are arguing that PCs have legitimized themselves as servers. The evidence appears to be that large numbers of the Fortune 1000 have been (or are) installing NT Servers instead of upgrading more expensive UNIX systems.

On this basis it is often argued that it is now UNIX as well as the mainframe which are doomed (both will get eaten by NT Server). The expectation seems to be that the rapid advances in PC-based hardware and operating systems — allied to databases, and transaction processing — will rapidly overtake whatever niches the UNIX world managed to grab. The presumption seems to be that, while the mainframe will continue to hang around for some time, it will ultimately be replaced by (Wolfpack) clustered PCs where the superior ease of use and ease of scalability will eventually win the war.

However, before PC folks start crowing too soon, it is instructive to analyze why UNIX failed to dislodge the mainframe. The same Achilles' heel that clipped UNIX's aspirations to displace the mainframe is shared by PC servers. These weaknesses — each of which is discussed subsequently — can be listed as follows:

- CPU-centric design
- I/O starvation under load
- poor memory and I/O disaster recovery
- weak clustering (compared to the mainframes)
- feeble system tuning under load
- lack of sustainable 99.999 % uptime.

PC servers: kings of 1970s mainframes

Today's PC servers are — by all metrics — equal to or superior to the mainframes of the 1970s. Unfortunately, it is not 1970 or even 1979 anymore. The essentially CPU-centric design of today's PC servers reflects the state of high-end mainframe design of the 1970s — not that which is available (for example) for OS/390 today.

Furthermore, the PC server's CPU-centricity leads to:

- fundamental reliability issues
- questions about sustainable throughput.

Since the early 1980s, mainframes have abandoned the CPU-centric view of the processor. Modern mainframe designs are memory-centric, coupled with redundant system buses. The CPUs (typically 4, 6 or 8 per system) are nothing more than fancy peripherals hanging off buses. Additional engines — for example I/O processors, clustering processors (a.k.a Sysplex coupling facilities) and service processors are also hung off the system buses — replicated, as needed, based upon throughput and/or reliability needs.

Mainframe operating systems (like MVS, or OS/390) exploit this view of hardware. If a CPU crashes, MVS Alternate CPU Recovery (ACR) just 'shrugs off' the problems. ACR can:

- re-schedule an interrupted program in progress on a different CPU
- back-step the PSW to just prior to the failing instruction
- re-start the failed program from where it left off.

If a CPU fails while executing inside the bowels of the operating system, MVS Function Recovery Routines (FRRs) can:

- clear up any CPU-specific locks

- re-schedule the operating system service call on a different CPU
- re-execute the service.

To date, no PC or UNIX operating system (including NT Server) is able to provide such sophisticated levels of recovery. In consequence it is difficult — if not impossible — to achieve the 99.999 % reliability which businesses increasingly require for mission critical applications.

PC servers: masters of I/O starvation

Part of 'Amdahl's Law' states that — in order to provide well-balanced system performance — there should be a minimum of 1Mbyte of sustained I/O throughput for each 1 MIP of processing power. Judged on this metric, PC servers fail miserably.

Current PC servers easily achieve 200+ MIPs of processor performance. Unfortunately they possess I/O architectures which have a hard time reaching even 40MB/second of sustained throughput. For reference this approximates roughly to 1970 mainframe I/O performance. In contrast, a typical mainframe — today — can push sustained I/O throughput rates in excess of 500MB/second.

Because of the consequent system bottlenecks, I/O throughput has become a limiting factor when determining how large a database — and how many users — a PC server (or clusters of PC servers) can realistically support. While it is certainly possible to put a physical terabyte of data on a PC server, it is quite another to get high throughput against it with an update-intensive and/or I/O intensive workload. Benchmarking such a PC workload against a mainframe — which has more than 10 times the I/O throughput — is a losing proposition. In addition, backing up, restoring and logging also end up being hamstrung by the woefully inadequate I/O architecture which exists on today's PC servers.

While games can be played (caching large parts of the database in main memory, using battery backed up RAM, replicating parts to clustered servers, etc.), a key limiting factor in PC scalability continues to be its poor I/O architecture. Large databases dictate large I/O throughput requirements — for updates, logging, etc. Building large databases and clustering on top of today's PC I/O architecture (which has feet of clay) will not produce acceptable scalability.

Poor memory and I/O disaster recovery

As if this was not bad enough, current PC servers have significant reliability problems when dealing with memory and I/O failures. Most current PC servers rely upon single-bit error detection and correction (SEC/SED) techniques for handling main memory errors.

In contrast, mainframes utilize double-bit or even triple-bit error detection (DEC/TED) techniques. This is highly significant from a reliability (and hence uptime) perspective because of the dramatic reduction in size of RAM memory storage elements.

1970s style SEC/SED techniques are acceptable for memories of 64KB-256KB of RAM. But above this, such 1970s techniques are inadequate for large collections of 4M, 16MB, 32MB and above memory sizes.

If this is one dimension to the memory issue, another lies in background radiation. Older memory technologies were rarely afflicted by background radiation effects — such as alpha particles — because RAM storage elements were (by comparison to today):

- loosely packed
- as large or larger than an alpha particle.

An 'alpha hit' would — at most — knock out one memory cell. However, with more modern memory technologies, an alpha particle hit can typically take out two or more memory cells. On PCs, this results in a 'crash and burn' situation — because the hardware cannot correct such errors.

Since alpha particles are generated multiple times per day by everyday surroundings (bricks, concrete, plaster, etc.) this represents an on-going reliability concern. Amdahl Corporation (purveyor of mainframes) learned this lesson the hard way in the early 1980s when it had to undertake an expensive retrofit of its I/O processors on its 580 mainframes. They were 'crashing and burning' about once a month — due to alpha particle hits that were knocking out the I/O processor's 1MB RAMs, which lacked the more powerful error-

detection/correction techniques of main memory. Hence, if alpha hits could cripple mainframes, it seems only reasonable to assume that they could cripple (in a high availability production environment) the more dense yet less corrected memory of PC servers.

Mainframes also have sophisticated memory recovery techniques — to keep complexes running even when 'uncorrectable' memory errors (for example, large alpha hits) corrupt the memory pages that are running the operating system kernel. On a mainframe, if a memory error affects the OS kernel, the service processor:

- reloads the page to a different part of main memory
- changes the address registers to use the new page
- restarts the CPU processor that was affected by the error.

In contrast, NT Server and UNIX systems crash and burn in such situations. In a production environment, this is not acceptable — especially when a proven alternative already exists.

I/O errors are another area where PC and UNIX operating systems are significantly less reliable than mainframes. In a typical large server scenario (terabyte size databases, etc.), some form of serious I/O error typically occurs every other week (a controller going haywire, a disk drive going off the rails, etc.).

While PC servers have RAID arrays to handle disk surface media errors, RAID is useless for preventing errors such as hot I/O or missing interrupts. Because of the large amount of DASD and its continual pounding, most large data centers see hot I/O conditions at least once every two weeks. [A hot I/O condition occurs when an I/O controller or its microcode becomes confused and rains I/O interrupts on the main CPU. It does so at such a rapid rate that it can cause the main CPU to become swamped or overloaded with I/O interrupts — to the point that it cannot effectively do any other useful work.]

MVS is different. It has built-in facilities to detect such conditions. When these occur, it (MVS) automatically tells the service processor to 'fence off' (disable) the offending controller — by turning off its hardware interrupt gates.

In contrast, PC and UNIX systems end up in a mess in such situations. Worse yet, rebooting the system may not clear up the condition. The PC server just keeps being driven to its knees by an aberrant I/O controller. So much for enterprise level reliability at the PC server level.

The opposite of hot I/O errors is the lack of a return acknowledgment from an I/O operation that was initiated (such as a disk read) because no reply was ever received back from the device or from its controller. These are referred to as 'missing interrupts'. Where MVS possesses the facilitates to retry automatically (when such errors occur) — including using alternate I/O paths if necessary — most PC and UNIX operating systems just hang when such conditions happen.

Weak clustering

While all 'know' that Microsoft's Wolfpack and related PC clustering technologies are about to be the greatest thing since sliced bread, their sophistication pales in comparison to IBM's Sysplex clustering facilities for the mainframe world. While the PC clustering techniques hope to scale up to support a dozen or so PC servers, the mainframe today can cluster together over a hundred CPUs, providing aggregate support for over 800,000 users on one Sysplex complex. Suffice it to say, given the current design, it will be some time before Wolfpack and NT scale up to that number of users.

Mainframe software has been enhanced so as to use the Sysplex architecture for fully exploiting:

- parallel database operations
- flat file (VSAM) record sharing
- parallel logging across an entire Sysplex (consisting, for example, of dozens of copies of the DB2 database server and the CICS transaction server).

To offer a feeling for this, the new 'Sysplex-enabled' version of CICS 5.0 eliminates on-the-fly logging to disk. The only time that logs are written to disk is at system (or CICS) shutdown. During normal operation, all logging is now directed to

the Sysplex facility — which caches the logs in large, shared Sysplex memory. Since the Sysplex itself is serviced by a separate set of processors (part of the coupling facility), the main CPUs are no longer burdened by logging I/O overhead.

Lastly, new components in the mainframe operating system — for instance, the 'Workload Manager' — are able to monitor utilization of the Sysplex in a rules-driven fashion. These Workload managers can automatically:

- start new (server) processes on any CPU in the complex as the workload on the system increases
- terminate server processes as the workload drops.

Thus the system can dynamically increase or decrease its multi-programming level (MPL) on the fly. Comparable system workload management capabilities are lacking for PC servers as well as UNIX based operating systems.

Poor system tuning under load

Today's mainframe MVS has extremely sophisticated system tuning capabilities embedded within its System Resource Manager (SRM) component. This can tune the CPU utilization, memory usage and I/O demand of database servers, transaction servers and flat file programs — even down to individual users (for example, favoring throughput or response time for one user or group of users for high priority operations).

In contrast, PC server systems are almost totally lacking such tools. While NT nominally offers some tuning knobs, they are:

- uneven in their impact
- difficult to predict
- under heavy loads and are almost useless (even moderate loads can cause unpredictable response times).

For example, try starting up SQL Server on NT Server and cranking up a suite of different clients accessing it. At the same time go and start up a compilation on the same system — and for good measure try running a few DOS *.bat* files at the same time.

What you will find is that the system noticeably slows down (even screen re-painting suffers). Attempting to tune the system, to favor the database server under load, usually results in sluggish 'foreground' (GUI user) response, even when the system is no longer under stress.

The reason for this is that the current NT tuning capability is fairly static, rather than being self-adaptive, based on workload, as in MVS's SRM. As numerous enterprise studies have shown, erratic and unpredictable response times tend to frustrate end users.

No 99.999 % uptime

The current PC RAS (reliability, serviceability and reliability) philosophy is basic: it is crash and reboot quickly (also known as 'fail fast'). To be fair, both UNIX and PC operating systems suffer from this 'crash and burn' mentality for dealing with CPU, memory and I/O failures. Because the hardware is cheap, the operational philosophy seems to be 'replicate and cluster' — which is somehow supposed magically to increase reliability.

This 'replicate and cluster' approach — using lower reliability PC server operating systems — has yet to be proven in a 24x7, 365 days per year enterprise environment. Furthermore, building a replicate and cluster architecture based upon a slow I/O architecture with poor recoverability means you have to address and solve a number of non-trivial challenges.

What killed UNIX as a central enterprise server is that it could not provide the same level of 99.999% uptime that MVS can. Typical UNIX systems can run 1-2 weeks round-the-clock between crashes (about the same as NT Server).

For Fortune 1000-sized organizations, this is unacceptable. They expect much higher levels of availability. They are used to MVS levels of reliability which can run for months on end without a crash. Because of this huge reliability mis-match, the great downsizing movement of the early 1990s has significantly slowed. Indeed, in some corporations it is reversing itself.

Several major banks and financial corporations which four years ago were crowing that, within

five years, all of their key mission critical applications and systems would be running on enterprise level UNIX servers, are implementing a 180 degree turn — primarily because of the reliability issues raised here. There is little sign that PC servers — essentially offering the same levels of hardware and OS reliability as UNIX — will fare any better at the enterprise level.

Management conclusion

Today's PC architecture is inadequate for enterprise processing which demands maximum availability and manageability. I/O, to mention only one of the factors, will continue to act as a drag on how effective a multi-programming level can be sustained. In turn, this limits the enterprise role which PC servers can play (as happened with UNIX before) in supporting large numbers of users running I/O intensive workloads such as database updates.

Furthermore, it will be significantly harder for PC servers to replace mainframes at an enterprise level than most PC advocates want to 'fess up' to. Fundamental hardware constraints, especially when combined with recovery weaknesses in the operating systems, indicate that this will be significantly longer in arriving than most PC pundits admit. As such, PC servers suffer from many of the shortcomings that UNIX servers possess.

It is these operational shortcomings which have IS people backing away from using either PC or UNIX servers as high availability and throughput enterprise servers. PC (and UNIX) servers will be used as great departmental and branch office servers, where 'crash and reboot' is practical. Until, therefore, analogous PC hardware and operating systems reliability — and their recoverability deficiencies — are cured, PC servers will encounter the same fate as their UNIX brethren.

This has various implications for implementers of middleware. One such implication is to consider whether 'crash and reboot' is acceptable for linking departmental and/or branch servers. Another is whether 'returning' to the mainframe is a way to minimize the complexity which middleware inevitably seems to introduce. A third is relevant to distributed mission critical systems; from the above analysis, these may be particularly sensitive to the failure of any one component (unless fundamental asynchronicity has been designed in from the start). A fourth implication concerns middleware itself: does it magnify or decrease the relevance of the issues raised here?

All the above issues are highly relevant to any organization with mission critical requirements. In practice, these considerations are all too rarely addressed with sufficient attention.

Enterprise integration, information flow and middleware

John E. Mann
Vice President, Research
MIDDLEWARESPECTRA

Management introduction

The middleware market is experiencing rapid growth and change. This is being driven by multiple market factors, including the pace of business change, the scope of integration projects being attempted today and the decentralization and heterogeneity of computing brought on by open systems.

At the core of this change is 'Enterprise Integration'. This is integration over multiple applications as well as across such organizational spans and technological heterogeneity that a new integration approach is required — with messaging middleware at the core.

Integration through this approach will appear in two overlapping forms: one is applications requesting services of other applications, and the other is information flow, originating in one or more applications, perhaps merged or transformed, and published over the network for 'subscribing' applications to use.

This new approach presents enormous opportunity for users and vendors alike. Yet a complete solution to the enterprise integration problem entails more elements than are present in most of today's middleware products, which were designed primarily to construct large complex applications.

What is 'enterprise integration'?

'Enterprise integration' is emerging as a distinct and important business problem in its own right. Representative examples are easy to find:

- a U.S. telephone company implements a distributed computing infrastructure in order to give itself a flexible base for innovation
- an insurance company — known for its quality of customer service — seeks new approaches to integration as its customer demographics undergo major change
- an SAP customer looks for a practical way to integrate its non-SAP applications with R/3.

Looking at these and similar scenarios, one sees executives looking at integration from a cross-organizational perspective and making it an objective of high importance. Information technology now so pervasively supports business operations that enterprise integration (or EI) is coming to be seen to be more about running a business, than about technology.

Market drivers

Signs of interest in EI abound. Users are saying that they want current, up-to-the-minute information on the state of their business, across conventional management towers.

Equally telling is the willingness of companies to invest tens of millions of dollars to implement business process re-engineering around R/3. SAP may be fashionable, but with $50M worth of smoke in some single implementations there must be a fire (a business problem) somewhere.

Indeed, user interest in EI is being driven by many, many different pressures:

- the simple fact is that most of the easier IT jobs have already been done
- initiatives to improve customer service are often forcing integration across previously isolated applications
- the furious pace of software innovation, enabled by ever less expensive systems, is providing users with a plethora of novel ways to apply technology to business problems which, in turn, create many of the integration challenges
- a surge of decentralized IT decision-making, again enabled by inexpensive computing hardware, is increasing the complexity of the integration challenge
- the extraordinary acceptance of the Internet as an additional business medium — whether for communication or support or electronic commerce — is pushing organizations to make whole new investments
- rapid change and short time frames are forcing users to integrate and reuse (rather than reinvent) applications
- renewed recognition of the value of existing transaction processing is encouraging its reuse through integration
- the data warehouse concept is enabling business (rather than IT) to shape what it expects from information.

Defining enterprise integration

EI is the integration of highly complex and mission critical applications in an environment of:

- short timeframes
- autonomous organization units
- heterogeneous platforms and applications
- dynamic change and uncertainty — in both technology and business.

What is apparent is that traditional integration approaches — such as sharing databases or choosing and conforming to abstract standards — are largely infeasible at this level. Instead, a different approach is needed — and integration at the enterprise level requires new methods.

In this context the word 'enterprise', therefore, implies an enterprise-wide, strategic perspective.

Enterprise integration supports high level, business-oriented organizational issues, such as:

- dealing pro-actively with wholesale change
- balancing autonomy and empowerment with sufficient control
- iteratively improving critical business processes (including obtaining information needed for that improvement)
- extracting ever more value out of data.

Because it looks at an organization's entire information technology base — the applications, the communications and the infrastructure — it (EI) necessarily takes an architectural view. It considers strategies rather than technologies: specifically, instead of looking at middleware, it looks at the style of integration that middleware enables.

Defining elements: change

Users now realize that systems must be built for flexibility. A large insurance company, known for its customer service and exemplary IT systems, stated in 1995 that its traditional integration methods were insufficiently flexible to deal with new requirements driven by a changing customer base. Years ago a systems integrator at Digital Equipment Corporation complained that the foundation pillars for a good application design — the business problem to be solved and the technology available to solve it — were 'firmly footed' in shifting sand.

Moreover, the large scale of current projects guarantees change and the complexity of these projects threatens to turn these changes into nightmares. It is certain that by the time a large project is finished, some part of the requirements — frequently not always fully understood by the business in the first place — will have changed. Accordingly, one of the specific objectives of EI is to withstand, even to exploit, the impact of change.

Defining elements: complexity

Faced with competition, users have attempted ever more difficult projects. Complexity is a frequent and well-known hazard in such cases; failure on one part of a large scale, tightly integrated project can be capable of dragging down the whole effort. Furthermore, the co-ordination of huge monolithic projects is difficult.

Complex systems are also difficult to change. ERP application vendor SAP is talking constantly about revising its applications and 'breaking' these down into components in order to meet customer demands for more flexibility. Enterprise integration, therefore, must necessarily address the challenges of complexity.

Defining elements: organizational scope

Increasingly, applications involve multiple autonomous departments, and even other firms. As such, the systems environment is one of increasing diversity with different standards, platforms, applications, infrastructures and even assumptions.

Even if it were possible to begin with a common base (say, implementing R/3 across a global enterprise), for example, time, change and organizational autonomy will likely deliver increasing diversity as local groups seek to improve their systems to meet local conditions. Enterprise integration, to succeed, must deal with the reality of diversity that accompanies large scale business projects.

Defining elements: existing investments

From the enterprise point of view, an organization's applications range in age from brand new to decades-old. This will always be the case. One of the realizations of those who focus on EI is that any enterprise-level strategy must deal with this reality in a long-term, not an ad-hoc, manner. To business executives — often brought up with brand and similar considerations that span decades (Coke, Mercedes, etc.) — this is common sense. With enterprise integration, IT must necessarily take on the same long range perspective.

In summary, enterprise integration — whatever it is — must deal with factors that were either less severe or not present at all in smaller-scale integration projects. In order to do so, solutions must face the issues of change, complexity, organizational scope and legacy integration head-on. These are business issues, not technology issues.

There is indeed a solution

There has been a solution appropriate to EI available for years. This has been applied primarily in

larger projects where hiring expensive systems integration expertise could be justified. However, the issues of change, complexity, organizational scope and legacy integration are becoming so important to more and more organizations, that this is encouraging the market to produce more robust off-the-shelf solutions even as users push to find justifications.

As with any potentially valid solution, this one is firmly rooted in the underlying business problems and therefore appeals not only to traditional IT staff but also to business executives. The technological core of the solution lies in message-oriented middleware, a technology that has been deployed for over a decade for integrating applications that were not meant to work together.

What is more, this is a proven approach. Thousands of person-years of experience have been accumulated in this kind of work. Over the past ten or fifteen years, dozens (perhaps hundreds) of skilled systems integrators who have had a choice of integration methods have chosen a message middleware-based approach to integration, even though that choice required them to first build a message-oriented middleware component.

Indeed, most of the currently available message-oriented middleware products originated in integration projects. In fact, in important respects middleware-based solutions look like the way a company or a team of people is organized:

- the solution is composed of autonomous components
- applications are assigned responsibility to perform each function separately (implementation details within each are left to that application)
- applications perform functions either upon direct, explicit requests from another application (or user) or on receipt of information that the application is assigned to monitor (or respond to)
- information movement is usually 'almost immediate' (real time or near real time) depending on the particular business need.

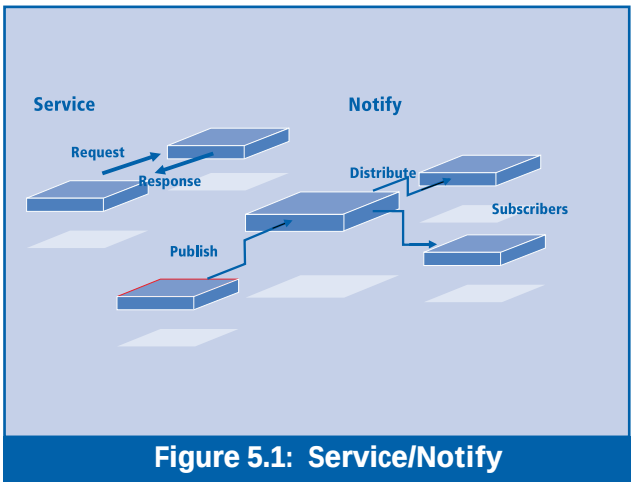


Figure 5.1: Service/Notify

Figure 5.2: Transquest

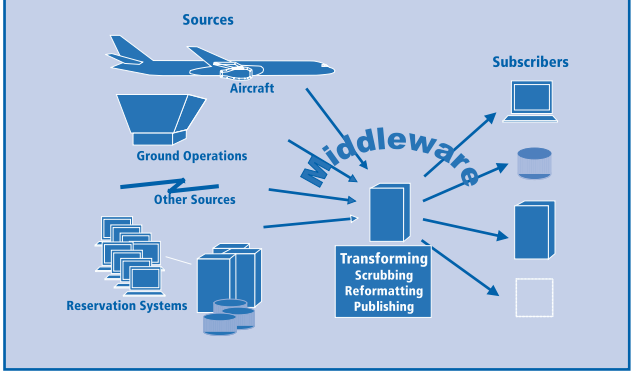
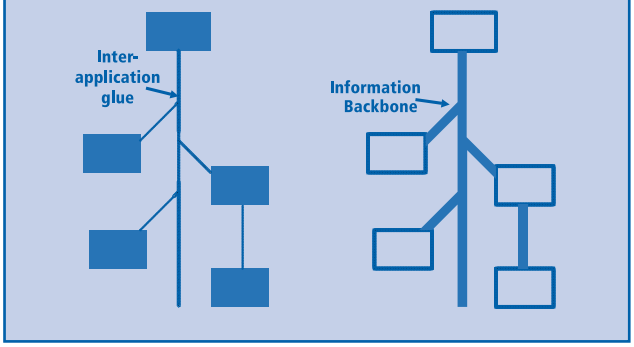


Figure 5.3: Glue and backbone



- information and requests are moved between applications using a reliable network-based communications system

Thus, a large integration effort entails:

- assigning responsibilities to individual applications, for performing specific functions and for sending and receiving specified types of information
- putting the applications to work undertaking those tasks
- specifying how information and requests will flow between applications
- defining what interfaces and message formats should exist.

Piece-by-piece

The result is that information flowing between applications is usually — although not always — moved one piece at a time instead of in batches:

- a piece of information or a request is the 'message'
- applications interact with other applications by sending, receiving or responding to messages which may contain requests, replies or information
- an application can react to an incoming message by performing an action (but this does not necessarily require responding to the source of the initiating message)
- applications not originally designed to receive or respond to messages can be revised or fitted with a 'wrapper' to enable them to do so (this is often simple, but not always).

This approach has a number of important benefits:

- it tolerates heterogeneous implementations, since application implementation is left to the application
- it adapts well to change, even change that occurs within a project, because it focuses on interfaces and communications rather than application implementations
- it allows a system as a whole to evolve piecemeal.

Messaging all by itself does not comprise the complete solution — there are many other implementation issues. But messaging can characterize the overall strategy, or integration approach. Two models stand out in this arrangement (Figure 5.1):

- one model consists of applications explicitly requesting services (such as return of information) from others
- the other consists of information flow between applications that publish that information (such as orders) and others that receive and utilize the information.

However, messaging supports virtually any possible communication flow pattern.

Information flow

The second model — information flow — indicates a possible important future direction of enterprise integration. A key feature of many operational large-scale middleware-based integrated systems is a mission critical flow of information, as a number of user organizations have recognized.

For example, MCI has supplemented purchased commercial middleware with a layer of its own software that adds security, management and other facilities not provided by its choice of middleware product(s). Having developed this, MCI's IT organization has been able to deploy its middleware widely and now encourages its use by many applications. Called the Registry, it carries millions of messages per week and supports dozens of applications.

In another industry, Transquest provides information technology services to Delta Airlines (and others). It has taken a different direction. It focused on application and data problems instead of infra-

structure per se. Nevertheless it ended up creating an information backbone (Figure 6.2).

This arose because one of the factors about which Delta is naturally concerned is operational quality — the percentage of on-time departures, arrivals, and so forth. Because Delta operates hubs in a number of cities, several times a day several dozen planes arrive to fill all available gates as they unload one set of passengers and load the next set. As at all hubs, many of the passengers are making connections.

Obviously such an operation needs to run smoothly if it is to work at all. The status of each flight on each leg — and of each aircraft and each gate — are all important. For example, if a pilot discovers something needing repair prior to the next takeoff, the sooner this fact is known the quicker a solution can be put in place. No passenger likes to connect to a flight that is delayed or cancelled. Yet, the information that is needed to reduce delays comes from several sources:

- ground operations is the source of information about gate departures and arrivals
- the aircraft itself transmits events such as touchdown and wheels-up
- flight scheduling details come from the reservation system.

All this information is moving between systems using message-oriented middleware. The information must not only be collected from multiple sources, it must also be distributed to multiple destinations. For example, executive staff want to know today's percentage of on-time departures as well as review past records, often using data analysis or mining tools to look for patterns.

Between source and destination, the information must also be transformed. At Transquest, reservations data is moved off the reservations systems onto other computers before being accessed, because the former systems are both heavily loaded and difficult to program.

More generally, all data is changed into formats that are convenient and consistent for the receiving applications. Then the data is 'published' using a publish/subscribe model. Developers at SCG Partners, Inc., the systems integrators that did much of the work, describe how they can watch an aircraft take off in Atlanta and within five seconds see the on-line flight status spreadsheet — which subscribes to the flight information as it is published — reflect the wheels-up event.

From inter-application glue to information backbone

What is so relevant about the Transquest story is that, rather than being some ad-hoc integration of a collection of applications (using middleware as glue), it is the construction of an information flow. Now that this flow has been established, groups within Transquest can — and do — write additional applications, extracting even more value out of the same information. Applications that can utilize given data simply 'subscribe' to that data. The role of the middleware-based infrastructure is to move, transform and deliver information between applications — to be an information backbone (Figure 6.3).

The concept of information flow, therefore, retains the desired characteristics of an enterprise integration solution as described earlier in this paper. Moving from interapplication 'glue' to information backbone may be an intellectual distinction — a new way of looking at integration — rather than a substantial difference in implementation. But it has several implications for the future of middleware:

- middleware's role in enterprise integration clearly stands out, as the transportation system that moves data
- integration involves transforming and distributing data as well as transporting it; middleware by itself does not completely solve the integration problem — as NEON (New Era of Networks) and Active Software recognize in their products
- an information flow offers value beyond moving messages, as Oracle is offering via its Advanced Queuing (AQ is part of Oracle 8): via AQ, the information as well as the flows moving through a corporation can be archived to yield additional value as data warehouse material.

In summary, the solution to enterprise integration is to organize applications and infrastructure in a way remarkably similar to the way organizations structure themselves — by assigning responsibilities to largely independent units and facilitating communications between those units. Because this middleware-based approach closely parallels the organization it will have to support, it promises to be an extremely durable and effective solution.

Alternatives: big 'ERP', sharing data and standards

There is currently much discussion about integration. Some observers feel that middleware-based solutions remain too difficult to implement — which, perhaps, explains why so many custom solutions are still offered to integrators to complete. At the same time attention is also being paid to other ideas and market realities, which must be integrated or absorbed into any realistic enterprise integration approach. The question is, how do these "alternatives" fit into enterprise integration?

The most easily accepted alternative — primarily because it readily appeals to rapid decision making — is 'Big ERP'. SAP, Oracle's Financials, PeopleSoft, BAAN and others dominate this market. The attraction is that 'Big ERP' appears to solve all — without the need to think about middleware — in one purchase and implementation.

This is a misconception although there is one good justification. The sheer scale of R/3 just about proves the contention that building an ERP system in-house is all but impossible today.

In manufacturing, SAP's R/3 is being introduced with enthusiasm. Large companies (including IBM, Digital and Microsoft) are spending upwards of $20M each on R/3 on efforts which include business process re-engineering. Others of the ERP providers offer systems that encompass similarly large segments of a company's operation.

However, most users who have implemented R/3, or anything of similar magnitude, probably do not want to go through that effort again. They want the current implementation to bear fruit for years, perhaps decades, to come. But even the largest of such applications cannot do everything.

Some users view the role of 'Big ERP' as a backbone application to which other applications will be integrated — with information flowing back and forth between the core application set and the outlying applications. The principle of 'backbone application' applies equally well to companies such as transportation companies, financial institutions, and others that have written their own major systems.

Thus, the implementation of a large, core application is not an alternative to building a middleware-based information flow but a starting point that defines a great deal of the information flow immediately which should make subsequent work easier.

A second, more traditional, approach to integration is to move and distribute data — not through networks but through shared databases or batch job streams. The problem is twofold:

- first, applications that directly access data are dependent on that data format, which entails a loss of flexibility
- second, when a data format needs to be changed in a database, all programs that read or write to it are affected.

One of EI's goals must be to escape this kind of inflexibility which data sharing usually produces. Conceptually, data sharing is attractive. In practice, it is not. It imposes a rigidity which often does not complement the business objective.

Standards are often claimed to be a third alternative. While standards are often helpful it does not follow that EI can be accomplished simply through standards. By definition, the scope of an enterprise integration project is too large to expect success trying to impose standards.

The big point about messaging is that it works, and will continue to work for an indefinite time, without requiring adoption of a contemporary standard. It can even integrate multiple solutions, each of which were built around a different standard.

In this context, object technology is an interesting case. It offers many concepts — for example, encapsulation — that are constructive in enterprise integration as well as an essential part of middleware. It is important to distinguish the valuable principles, methods and even some of the standards of object technology from the notion of using standards as a basis for EI.

Overall, the middleware-based enterprise integration approach encapsulates and accommodates existing and new systems even if they are themselves huge in scope, implemented using obsolete methods or even based on new techniques. This is what 'real' EI should be about.

Does a complete solution for EI exist?

If there is a nascent market ready for middleware, is middleware ready for the market? The answer is — not entirely. There are major pieces missing. Part of the problem is that most middleware vendors seem to want to sell only parts of the solution.

Users do not like fragments of solutions. Despite Andrew Grove's beliefs (or perhaps desires), most users do not buy chips and components to make their own PCs. People want to buy the whole computer or the whole car. SAP owes much of its success to its ability to provide an integrated solution. Secondly, transporting data is only a small part of the problem. Users need products to help them:

- develop applications
- model the information flows (which model will be used also to report status at run-time)
- monitor the information flows from a business / application perspective
- operate the network, adapting to outages, etc. (with help from assigning priorities, classes of service, etc.)
- store and manage object interfaces, contracts, etc. (probably using repositories)
- transform data formats in real time
- implement security
- analyze security risks based on data access and origination rights.

The 'middleware' industry is not standing completely still. Some vendors — such as Candle, Active Software and NEON — have begun to broaden the scope of the 'middleware' product category. There are vendors of data transformation products, such as TSI, that provide pieces and are partnering with other middleware firms including IBM. The Big ERP companies are also interested in infrastructure — and will probably influence the market if not actually supply elements of the solution.

The area that users wish vendors to investigate is the notion of seeing EI as building an information flow — as exemplified by the Transquest and MCI examples. This is easy for users to communicate as well as justify and even implement. Part of its attraction is that it borrows from some earlier ideas — such as data warehouse — and seems to enable users to 'start small' and grow. It is why information flow has so much potential.

Management conclusion

Enterprise integration is a business problem, easily defined in business terms. It is a new problem, brought on by the unprecedented scale, scope and complexity of the applications being attempted today — as well as by the dizzying heterogeneity of existing and future components (out of which systems are and will continue to be built). This problem occupies the attention of most businesses today.

There is at least one solution for this problem. This is based on message-oriented middleware. One advantage is that it can be described by analogy to organizational structure. This facilitates large scale implementations by breaking the challenge into manageably sized pieces which can then communicate with each other using messaging. Indeed, a closer look reveals a promising conceptual approach, that of viewing integration as implementing information flows through the applications in the organization.

Other methods of integration do exist. Several older ones (such as sharing data from a common database or imposing organization-wide standards) are appropriate at scales below the enterprise. At the same time, modern approaches — like large application suites or 'Big ERP' — inevitably entail middleware, although this may not have been realized at the time.

Enterprise integration demands an integration approach that can address unprecedented scale, scope, complexity, change and heterogeneity. Adjusting for this requires new thinking, but its solution is business-oriented which affords an important opportunity to users.

The 'New Age Enterprise': developments and middleware

Tom Curran
Principal
TC Management

Management introduction

Is the Internet middleware? This is a question often raised. On first sight the answer is 'No'. It lacks the sophistication and reliability needed to be able to undertake commercial transactions.

At the same time, many will argue that enterprise applications from vendors like SAP, BAAN, Oracle and others are acting more and more like middleware. They are to be found at the heart of most large enterprises (see also page 30) and other applications increasingly have to be hooked to R/3, Triton, etc. In other words, enterprise software has almost become a form of middleware in its function.

Tom Curran has called the result the 'New Age Enterprise'. Such an enterprise exploits the obvious opportunity of the Internet, and uses communications and defined interfaces not only to integrate its own applications but also to exchange information — including business transactions — with other organizations.

In this analysis, Tom Curran looks at the shape of the New Age Enterprise application and how such applications — and enterprise vendors and middleware — may evolve. Using SAP as his primary example, he discusses how the New Age Enterprise application is going to have to change as it increases its dependence on the Internet as part of the middleware equation.

New horizons

On the horizon for enterprise computing is a new generation of standard software. Allegedly this will enable organizations to bridge proprietary transaction processing across existing enterprise application software and database systems. In all but name this has middleware running through its midst.

The resultant hybrid software environment — connected by Web-enabled middleware — will permit companies to go beyond the functionality that enterprise application vendors have traditionally offered. It will integrate specialized components which, in turn, will fuel productivity as well as satisfy the desire to have a company specific enterprise computing platform. At the same time, advances in the area of open transaction management and business process blueprints will likely spawn new types of inter-company collaboration.

By relying more on third-party software developers, SAP — to use one leading enterprise application vendor as the example — is attempting to deliver the functionality, integration and requirements that many of its customers desire. SAP is moving to enable third-party software developers to interact better with its infrastructure and interfaces through add-ons and so forth.

This 'market enlargement' will not only help achieve the goal of a more seamless integration with the Internet. It will also encourage more tools and software developers as well as more BAPIs for building Internet applications.

The seamless integration of SAP's technical infrastructure with the Internet is, however, just one step. The next and perhaps more intriguing step concerns how SAP and its third-parties plan to exploit this infrastructure in practice. To succeed, vendors will need to hook various middleware technologies together — a complex prospect.

Intranet development

The Intranet is swiftly becoming vitally important for business transaction processing, largely because it does not necessarily make explicit use of the public communication system (the Internet). Every major enterprise software player has announced an Intranet solution but the main 'development' is, so far, usually only the addition of a web client.

Using Intranets, businesses can deploy Internet technology internally (within their organizations) to enable employees to take advantage of technological progress and better communications. For example SAP's Release 3.1 Intranet scenarios cover support for:

- service calls (for example, customer service on-line logging which allows customers to identify product problems by choosing from a catalog of pre-identified potential problems)
- who is who (for example, for communications between employees of large companies using the company's Intranet as an extension of SAPoffice)
- OLAP serving (for example, consolidation of information for analytical and reporting purposes)
- purchase requisition handling (for example, simplifying internal purchase requisitions)
- reporting 'actuals' (for example, using an ad hoc Web browser-style interface to post activity allocations remotely from one cost center to another).

In this context, Intranets are playing an early and distinctive role in the New Age Enterprise. For this reason, SAP — and competitor application software — are making large commitments to a 'middleware' infrastructure which links enterprise customers to the Internet .

Necessarily, this involves middleware — although many might not think (or may prefer not) to call it this. Indeed, one characteristic of the 'New Age Enterprise' is that there are many innovative products lining up to fill this 'niche' that is a mix of the Internet and middleware. Take supply chains as an example (and Figure 6.1).

Supply chain

Continuing with the use of R/3 as a role-model, SAP is planning to make its Reference Model more integrated with the Internet, and with Internet scenarios. The goal is to show how additional

processes and scenarios can run outside R/3, including:

- obtaining order confirmations through Internet mail
- ordering from product catalogs on the Internet
- etc.

With the heightened emphasis on the Internet and implicit market enlargement through third-party software development, SAP has decided that more and more it cannot ignore value-adding processes which currently lie outside R/3. Equally it knows it cannot move fast enough to include everything inside R/3 (like most software vendors, this was probably its first desire).

Instead it is gradually acknowledging that it will have to map processes to R/3 that are essentially outside of R/3. While one could argue that toolsets — such as ARIS Studio, Visio Business Modeler or IntelliCorp's LiveModel — already fulfill such a role, SAP is clearly looking to extend its influence — supplying the reference point where all these integrate.

To achieve this, SAP will need to improve the structure of its Reference Model so that the models may be better understood by people outside of the data processing community — such as finance professionals and front-line managers, who not only know the business better but are closer to the processes themselves and who may even be the process owners. Improvements in the Reference Model's notation are, therefore, under way.

Facilitating software construction

The next important area of R/3's Reference Model development involves improving software construction from models. The goal for process model configuration and automated customization is to enable users to work mainly from business process models.

SAP's concept is that, for example, a consultant should be able to deliver customization right from a model. This goal will only be realized once models are developed which are based on specific industries — in the jargon — via 'industry-specific' templates.

This leads naturally to a third area of Reference Model development, the creation of vertical market blueprints or templates. While the Reference Model has proved useful for improving the understanding of organizational structures, the process models now cover enough areas to tackle entire industries.

Special applications for specific industries enable business engineers to start with an industry template and build from there. For example, the intention is that someone in the publishing business can draw from a specific Reference Model which is oriented to publishing and configure an individual solution accordingly.

Traditionally SAP has been strong in petrochemicals for 25 years. It is now looking to expand in other areas and countries. In the U.S., for example, SAP seeks deeper contact with the high tech, chemicals and consumer package industries. But to deliver this objective, R/3 needs to be aligned with its chosen specific industries. Evidence of its efforts

While SAP is a pioneer in the creation and publication of business blueprints, others in the software industry have banded together to work on a larger, more generic business process model. Called the 'Supply Chain Operations Reference Model' (or SCOR), this model is backed by the Supply Chain Council which is:

- a non-profit corporate special interest group based in Cambridge, MA
- oriented to improving supply chain process and related descriptions
- mapping the results into software purchasing decisions and the way these are handled.

SCOR has the potential to become the industry blueprint of choice since it is not specific to any one vendor (but that has also been said of other similar groups). This makes sense for the New Age Enterprise: a multi vendor approach will be required for both intra-business as well as extra-business collaboration.

Figure 6.1: SCOR

will be seen in R/3 Release 4.0 — where a large set of new functionality will be built around retail (for example).

Internet applications

In R/3's 'implementation' of the New Age Enterprise, two primary Internet application scenarios stand out:

- consumer-to-business
- business-to-business.

In the consumer-to-business relationship, the customer or potential customer uses a Web browser to access the vendor's business system to review a catalog, place an order or inquire about a product or service. Consumer-to-business applications which use the Internet benefit from offering products and services to consumers on a geographically global basis.

To support this, the Internet Application Scenarios (in R/3, Release 3.1) for consumer-to-business, include support for:

- employment opportunities (for example, enabling direct recruiting through the Internet)
- electronic correspondence (for example, issuing automated Internet reminder notices)
- interactive requests (for example, offering on-line customer information giving customers access to information such as current bank information or account balances, billing status, etc.)
- order entry for variant products (for example, supporting on-line ordering).

Business-to-business

While consumer-to-business is interesting, it is not seen as the killer app. A 'killer app.' is the term used to describe a product which takes a particular software product, market or platform into the mainstream. Business-to-business applications look as if they will become one such killer app. which will spawn whole, new commercial opportunities.

In business-to-business relationships, integrated business systems co-operate with each other by using common standards to deliver business transactions (using, in SAP terms, BAPIs — Business Application Programming Interfaces). Such applications, once in place, create an electronic marketplace that facilitates ad hoc business-to-business matching, including synchronization of purchase requisitions and sales orders from different enterprise systems through a common transaction — or such is SAP's aspiration.

Such scenarios will normally involve a supply or value chain:

- bank data transfer (for example, on-line banking — which will allow for the exchange of banking, accounting and bookkeeping data through the Internet while supporting different country-dependent standards)
- Kanban (for example, enabling checking and replenishing of just-in-time product inventories)
- special stock enquiries (for example, supporting on-line stock enquiries for commission, contract and job processing, work-in-process inventory, etc.).

This is, in most people's terms. middleware. It is also an unfamiliar way to describe middleware — which will, no doubt, some cause confusion.

Electronic commerce

After today's first wave of Internet use, the second will dominate — driven by business-to-business electronic commerce. From 1994 to 1995, sales over the Internet grew from $60M-$200M. Estimates for the year 2000 project sales to range anywhere between $70B-$200B worldwide. Most of this will come from trading between organizations which map themselves to New Age Enterprise principles.

Electronic commerce essentially represents consumer-to-business and business-to-business communication. Utilizing digital communications and

technology, electronic commerce will facilitate the buying and selling of goods and services including the transfer of funds. It will embrace such common inter-company and intra-company processes as:

- procurement
- order handling
- production
- marketing
- sales
- distribution
- etc.

Electronic commerce will also have many off-shoots in areas as diverse as:

- electronic data interchange
- electronic funds transfer
- bank card operations
- digital cash.

It (electronic commerce) is, therefore, the prime target for those interested in growing via the Internet. While some electronic commerce is already undertaken over the Internet now, this is primitive by the standards of what will happen in the New Age Enterprise.

Today, most is still based on e-mail and simple messaging. It lacks the sophisticated capabilities for enabling complex transactions.

As a prospective 'New Age Vendor', SAP proposes to play in this arena. It plans to enable customers and third-party developers to use and develop the mass of business logic already built into R/3. It hopes to deliver 'superior' applications to enable electronic commerce, by:

- making it easier for third-parties to work with its Business APIs
- offering the opportunity to incorporate third-party modifications and improvements into the R/3 standard system
- enabling R/3's basic infrastructure to be exploited as the 'connecting' middleware.

The larger, more market shaping activity, however, will take place in the field of New Age Enterprise application components. These software products will:

- tap into the mass of information contained in an SAP Business Blueprint
- drive process-oriented electronic commerce between companies.

The benefits for New Age participants will be:

- more fluid information exchange
- common process definitions around which business transactions can be based.

As part of this evolution, two new entities will emerge which will further refine (and define) the New Age Enterprise :

- autonomous business agents
- learning agents.

Autonomous business agents

In the New Age Enterprise, software will take a more active role in everyday work. With the emphasis on business processes, software vendors like SAP are already introducing new midddleware technologies — like autonomous software agents.

The autonomous software agent is a form of middleware which can act independently as well as be the hands and eyes for software components. Agents are adaptive. They can make decisions for the user rather than, as in the case of current software programs, depending on interface designs which require direct, personal manipulation.

Researchers are currently developing software programs that understand how the user interacts with a software program. At MIT, for example, researchers have built agents which continuously:

- watch a user's actions
- monitors his or her habits
- automate any patterns that the agent detects.

This has many possible applications. For example, an autonomous software agent can be used for searching for information on the Internet. Such an agent will learn the user's habits and then go out and use search engines on the Internet. It will learn what kind of information the user likes and will consistently search for and automatically deliver the relevant information to the user.

Because the Internet has created too much information for any one person to handle, business agents will likely have a direct and immediate impact on use of the Internet. Drawing on the information about business processes in the Business Blueprint, software agents will evolve as interpreters of how to harvest information related to any business process. They will do this by understanding the process model plus the repository meta model.

An agent will look at the meta model and interpret the relationships between business processes and business objects — thereby providing an information consumer (say a decision maker) with a wealth of information associated with that business process.

For example, assume that a company wishes to provide its customers with a new capability to do 'rush orders' over the Internet. From the enterprise software — in the SAP instance, R/3 — this would mean changing or configuring the model to accommodate rush order handling. After the appropriate changes are made, the model would serve as a blueprint for an autonomous software agents:

- the agent would be able to query the process model and understand what information is relevant
- this data would then be contained in business objects such as 'Customer' or 'CustomerOrder'
- the agent would make proposals about what to look at and how to prepare the information.

Companies like SAP intend to exploit actively the possibilities offered by business agents by using them to make certain decisions in the R/3 business environment. For example, a possible business agent scenario might focus on commodities trading. Companies could use business agents to keep their competitors from knowing what they are buying or in what quantities.

A manufacturer, for instance, might run into a shortage but might want to keep this information confidential. It could make use of business agents dedicated solely to conducting on-line 'enquiries' to purchase many small amounts of the necessary supplies — without tipping off competitors or suppliers about its commercial position.

Learning agents

Another area of autonomous agent research revolves around learning agents, which are essentially electronic teachers. Learning agents can teach the user according to the problems being faced. They can teach, for example, about:

- a particular software object
- software's new functionality
- what functionality the user should be interested in, based on his or her problems.

If the user encounters a repeated set of problems regarding the use of a certain function, the agent will point to the areas that the user should know. It will base its instruction not only on the specific problem, but also on the level of skill of the user. The learning agent improves the skills needed to perform a specific function.

Like other Internet agents, learning agents are especially adept at choosing the information that the user does or does not need to know:

- if the user is new to a certain functionality, the agent will recognize that and get him or her up to speed
- if the user is experienced, the agent will filter out the information that the user does not need.

Learning agents can be used as a teaching tool. This is especially relevant for companies that have to train large numbers of employees. Companies

do not have to move the employee to a training center and incur training costs, but can train them on the job — which is particularly relevant to enterprise applications and new ways of doing business (including accelerating deployment of enterprise applications like R/3). In this way, learning agents help to accomplish the goal of what companies call 'learning while earning'.

Moreover, given that today's companies are leaner, the typical employee now requires a much broader set of skills than ever before. Learning agents will be useful because they:

- assess the employee's existing skill level
- broaden it where this is necessary.

Furthermore, such agents will use the knowledge content in business blueprints. These may even go as far as to propose organizational changes or optimize informal communication channels. Germane to this change are insights about how people — and organizations, collaborate: in other words — middleware.

Enterprise software vendors have been slow to understand this phenomenon. Niche players — such as IBM's Notes and Radnet's WebShare — understand the importance of enterprise collaboration and provide products that enable collaboration.

Indeed, Radnet (a small Cambridge, Mass. software startup) seems to be betting its business on the concept that enterprise collaboration will be such a killer app. in the New Age. Its architecture is, arguably, completely New Age. Furthermore, similar companies understand the relevance of the Business Blueprint as an approach for automating products as well as introducing software agents for enterprise collaboration.

Applying agents to R/3

For example, one vision of using such collaborative agents would be to combine R/3 and its interfaces together with these learning agents. This would assist in accessing information better on the Internet. As tools, learning agents will likely help solve many of the problems which occur today during many (possibly most) initial R/3 installations.

In the past, SAP has had difficulty informing business engineers about relevant products, functions and relevant data. While the IMG does filter out much of the information (that installation and deployment consultants do not need to know), there is still far too much information to digest.

In the future, better information tools must be generated for better productivity. Ultimately, learning agents may well serve as an information 'release' tool for R/3 — and even its data. This should act to:

- drive down the price of enterprise software implementation
- change radically the way companies deploy and then use enterprise software products.

SAP, BAAN, PeopleSoft and Oracle are all moving towards some form of autonomous agent as a logical future step. As such, autonomous agents represent a field of the organization of human knowledge that has never existed before. They feed from a knowledge repository about business practices and processes which was never, previously, available. But they do need some middleware in order to operate, as the infrastructure.

Agents as middleware

As intriguing as this new technology is, there are still major obstacles to overcome. Any autonomous agent needs a stable infrastructure — middleware — that is not proprietary (if it is proprietary, then an agent may be prevented from working as intended — because it cannot talk to others).

Today's leaders seem to have decided that the Internet is such a middleware infrastructure. They are building their environments to work seamlessly with the Internet and its cousins, the Intranets and Extranets.

In the near future, Microsoft will have numerous products to enable any Windows-based PC to communicate with any type of Web site, software or scripting language which runs on more than one platform (such as JavaScript). This will provide autonomous agents with another part of the infrastructure they need to launch out and communicate with many different environments — not just Windows-based environments but systems like R/3 or Oracle or ...

Management conclusion

The Internet is a network with barely a handful of applications sitting 'natively' on it — FTP, Gopher, email, Usenet and the Web. While all these have business potential (support certain restricted kinds of activity), the notion of passing real business transactions over the Internet (and the Web) clearly represents a significant development. It is a space the 'New Age Enterprise' sees as an opportunity.

The New Age Enterprise envisioned by Mr. Curran promises many changes for corporate computing. These changes will be ushered in with the Internet and a host of innovative technology that will relate to the public domain infrastructure.

Not surprisingly, the Internet is a high development priority for SAP because the Web and the Internet can potentially:

- *serve as a low cost (in the context of R/3) alternative user interface for SAP's business applications*
- *open up new access paths and new business processes to customers.*

While technical obstacles still remain, the New Age Enterprise — along with advances in electronic commerce, third-party software development and autonomous software agents — is showing signs of adopting the Web and the Internet as the 'middleware infrastructure' for the future. As described, there are many areas currently under development. Indeed, the possibilities inherent in the New Age Enterprise are virtually limitless — but only so long as a middleware infrastructure exists which works.

Processes + transactions = distributed applications

Gustavo Alonso
Senior Research Associate
Institute of Information Systems
Swiss Federal Institute of Technology

Management introduction

In addition to the conventional operating system interpretation, the concept of process is increasingly being used to refer to complex sequences of computer programs and data exchanges controlled by a meta-program. The evidence is to be found in concepts like process-centered software engineering, business processes or process-based parallelism.

These ideas already have widespread acceptance in many areas, in particular where computation is based on clusters of PCs/workstations or on environments involving heterogeneous platforms and applications. Careful analysis — of areas such as business environments, software engineering or scientific data management — reveals a surprising number of problems which are both pervasive and common. Such pervasiveness may also explain:

- *the degree of attention being devoted to workflow (and today's best example of a process support system)*
- *why many researchers consider workflow management to be just a reincarnation of job control languages.*

This analysis argues that the notion of process, when augmented with transactional properties, has the potential to be a powerful tool for designing and developing distributed applications over clusters of PCs and workstations. It indicates yet another extension of middleware's relevance.

Top-down or bottom-up?

Today's computing environments have changed significantly since the conception of job control languages. In practice, the most widely available platforms for corporate computing are based on multiple standalone computers linked by a network. Such clusters offer the possibility of implementing truly distributed systems, which can be done in two ways:

- top-down where the entire system is designed from the beginning as a distributed system
- bottom-up, where existing applications are used as building blocks.

The former offers many interesting research opportunities. It has, therefore, attracted considerable attention. In contrast, the second approach is, above all, practical — since in most cases both the hardware and software infrastructures are already in place and cannot be discarded. While the most common notion of process — as described above — is still derived mainly from workflow management, it precisely targets distributed systems (Figure 8.1) built following a bottom-up strategy. Its relevance is that this is likely to be the approach of choice for future distributed systems.

Distributed applications

The type of distributed application addressed here is based on already existing, stand-alone tools located on clusters of PCs and workstations linked by a network. The desire is to provide a way to use these existing tools as the building blocks for higher level systems in which the process acts as the blueprint for both control and data flows.

Typical examples include business processes in which several office tools — for example spreadsheets, text editors, databases and human decisions —are combined into a higher level entity by encoding the business logic within the flow of control and data of the process. Among existing products, the ones most closely related to the notion of process (at least as used in this analysis) are workflow management systems. Existing workflow tools, however:

- are oriented almost exclusively to either business processes or imaging systems
- suffer from severe limitations related to performance and functionality
- are not easy to apply in other areas.

Nevertheless, workflow management should be seen a first potential step towards the support of development of complex applications over distributed systems using existing tools. Indeed, its concepts can be generalized by extending the notion of process to any arbitrary sequence of tool invocations (a script or a series of pipelined UNIX commands are simple forms of such processes).

Using this approach, future workflow systems may be able to act as high level programming tools linking heterogeneous, stand-alone applications. Integrated with additional technology — such as CORBA, queuing systems or TP-monitors — workflow management could yet play the role of distributed operating systems, particularly where it is possible to exploit the coarse parallelism and distributed characteristics of distributed processes.

Processes

A process can be seen as a description of an arbitrary sequence of application invocations along with the data flow between these applications. As such, a process acts as a meta-program governing the interactions among existing applications.

Each step within a process is an activity, which represents invocations of external applications. The flow of control within a process — what to execute next — is determined by control connectors labeled with transition conditions, usually boolean expressions based on data produced by the activities. Processes also include programming constructs that allow modular design, nesting and invocation of other processes.

During execution, the process engine navigates the description of the process determining which activities are to be invoked next (Figure 8.2). This procedure is very similar to that of executing any other program (albeit more like an interpreted program than a compiled one).

The description of the process is usually stored persistently in a database. The process engine consults the database continuously to find out what is to be done next. Any changes to the process are

also stored persistently (the status of executed activities, returned values, etc.), which opens up additional possibilities in terms of:

- recovery
- compensation
- overall reliability.

When an application is to be invoked, the workflow engine notifies an application agent located at the same node where the program resides. The application agent then executes the program and returns the results of the program to the workflow engine.

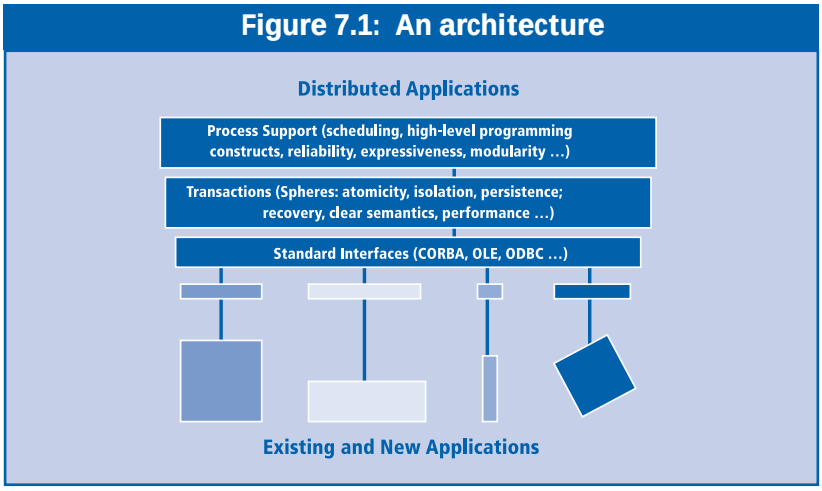
These ideas hint at the possibility of exploiting workflow engines as a distributed operating system — executing programs which have been constructed using existing applications as programming primitives. In practice, workflow engines already act as schedulers and resource allocators in distributed environments.

However, unlike in the case of centralized operating systems, workflow engines have to contend with the network. This immediately brings up issues such as:

- atomicity of invocations
- interferences between concurrently executed processes
- performance
- overall consistency.

This is where transactional concepts should play their part.

Figure 7.1: An architecture



Transactions

Transactions are the conventional form of encapsulating database operations so as to provide Atomicity, Consistency, Isolation and Durability. They provide not only clean semantics to the interactions between concurrent executions but also a powerful abstraction on which to base optimizations to the architecture of a database system.

The same ideas can be applied to processes so as to provide equally useful abstractions to:

- reason about the correctness of a system
- express properties of the execution of processes
- tune the overall architecture of process support systems.

It is likely that transactional ideas applied to process support systems will be as successful as when applied to databases (which, to a certain extent, has always been the goal of many advanced transaction models). However, in the case of processes, database-like transactions are too rigid. Some form of light-weight transaction needs to be used.

Thus, within a process — instead of using a unique construct encompassing all transactional properties (for example, Begin Transaction/End Transaction) — several separate constructs should be used to group activities according to the desired semantics. In this context, for example, one could consider spheres of 'influence' where these spheres can be naturally combined with the nested structures provided in most process specification languages. Such constructs could have the following semantics:

- spheres of atomicity (atomic units with standard all, or nothing, semantics)
- spheres of isolation (isolation units, much like critical sections in traditional operating systems)

- spheres of persistence (determining whether the activities in the sphere are to be made persistent or not).

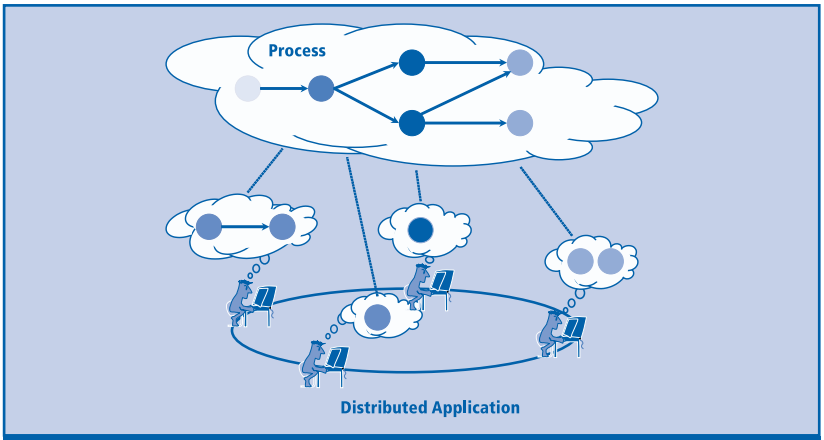


Figure 7.2: Processes to applications

Spheres of atomicity

Atomicity is the most popular property inherited from traditional transaction processing because it addresses a common co-ordination problem. There have been many suggestions regarding how to use atomicity in advanced transaction models and it has also been suggested as a basic mechanism for process navigation.

In fact, the prospect of process support systems may offer the only realistic scenario in which to use the many ideas related to compensation. For instance, activities could be declared to be Basic (non-Atomic), semi-Atomic, Atomic, Restartable or Compensatable:

- Basic activities are the default and correspond to non-Atomic applications — those for which the system cannot guarantee Atomicity
- semi-Atomic activities are those providing enough information to implement a roll-back method to be executed if an activity fails before completing its execution
- Atomic activities are those that preserve Atomicity by themselves, for example a transaction executed over an XA interface
- Restartable activities are those that can be invoked repeatedly, until they eventually succeed
- Compensatable activities are those that can be undone after they have finished, using a user provided method attached to the activity interface.

Note that these categories can be applied at different levels, thereby allowing you to group a set of activities into a semi-Atomic block or provide high level compensation for an entire sub-process by declaring it to be Compensatable. The same navigation mechanisms used for processes could also drive operations related to Atomicity properties.

Spheres of isolation

The semantics underlying spheres of isolation follow from the notions which point out that processes may require more synchronization than the traditional database concept of serializability (in the traditional operating systems sense).

For applications that demand a database-like approach, both spheres of isolation and spheres of atomicity could be used in a manner similar to that of nested or multilevel transactions. These provide a powerful mechanism to assist the reasoning about recovery in applications with a complex structure.

Spheres of persistence

Spheres of persistence could be used to avoid the overhead incurred by storing all process information in a database. Using databases for storage:

- provides significant advantages, in terms of reliability, monitoring and audit control
- but introduces a considerable overhead.

Within a process, a programmer could specify whether a set of activities is to be executed persis-

tently or not. If an activity or a group of activities are embedded within a sphere of persistence, every step of the execution is recorded in the underlying database. This guarantees forward recoverability in the event of failures.

The information related to activities not included in a sphere of persistence about the execution is maintained only in main memory. Upon completion — or after pre-determined time intervals — this information could be checkpointed to the database in an off-line fashion, thereby avoiding the I/O overhead.

Processes + transactions = distributed applications

In practice, the equation above should include 'standardization' on its left side. The amount of effort being devoted to CORBA, OLE, SOM/DSOM and ODBC — to mention but a few — demonstrate that there is a real and growing interest in being able to link together previously standalone tools.

In terms of products, the same goal can be ascribed to TP-monitors, persistent queuing systems, CORBA implementations and workflow management systems. These products are slowly converging towards a common point, as proven by the many ongoing attempts at combining their functionality. One has only to examine the following to understand the depth of interest:

- TP-monitors implementing execution guarantees in CORBA
- CORBA adding extensions for transaction processing and links to TP-Monitors
- the efforts being made to combine workflow standards and CORBA under the notion of business objects
- the addition of queuing systems to workflow management systems.

Once applications follow a given standard interface — and if these interfaces are widely accepted — the task of linking together standalone systems will be greatly simplified. But in addition to the standards there must a way to express interactions between standalone applications. Hence the importances of process.

Management conclusion

As users deploy more and more distributed processes consisting of applications driven by workflow engines, users will need more sophisticated process control than most of us seem to have realized. Certainly current workflow management systems will not be sufficient. Concepts borrowed from the transactional world will have to be added.

The environment described here will include all existing application elements plus a higher level workflow driver. The latter must provide new facilities for managing an environment in which multiple independent process streams will be running concurrently and whose interactions will be a concern. Conceptually, and in terms of its hierarchical structure, this is not very different from early transactional operating systems. But the problem of deploying distributed systems managed by workflow is relatively unexplored. It offers rich research and commercial opportunities. It also undoubtedly gives transactional concepts a continuing relevance as middleware and within information systems.

Take the muddle out of middleware; subscribe to: MIDDLEWARESPECTRA

ENHANCED INTRANET SUBSCRIPTION

The Enhanced Intranet Subscription provides you with multiple copies of each Report, for one year, plus additional middleware resources to put on your Intranet. It includes the following:

- ✓ 5 printed copies of each quarterly Report, published in February, May, August and November (these copies can be mailed to separate addresses)
- ✓ 52 pages of analyses and case studies in each Report, including diagrams
- ✓ a quarterly CD with a 12 month licence to publish the information on it internally on your Intranet containing:
  - ➢ the Reports of the Calendar year to date
  - ➢ the Reports of the previous Calendar year
  - ➢ the latest copy of **MIDDLEWARESPECTRA**'s Vendor Database (>200 vendors listed)
  - ➢ the latest versions of the 10+ volumes of the *Issues in Middleware Collections* (normally priced at US$ 3500)
- ✓ subscriptions: **USA/Canada US$3750; Europe/Rest of World GB£2500**

INTRANET SUBSCRIPTION

The Intranet Subscription provides you, for one year, with all of the following:

- ✓ 2 copies of each quarterly Report, published quarterly in February, May, August and November (these copies can be mailed to separate addresses)
- ✓ 52 pages of analyses and case studies in each Report, including diagrams
- ✓ each quarterly Report on CD with a 12 month licence to publish this internally on your Intranet
- ✓ a 30-day full money-back guarantee if you are not completely satisfied with the first Report/CD received
- ✓ subscriptions: **USA/Canada US$1250; Europe/Rest of World GB£795**

REGULAR SUBSCRIPTION

The Regular Subscription provides you, for one year, with all of the following:

- ✓ 1 printed copy of each quarterly Report, published in February, May, August and November
- ✓ 52 pages of analyses and case studies in each Report, including diagrams
- ✓ a 30-day full money-back guarantee if you are not completely satisfied with the first Report received
- ✓ subscriptions: **USA/Canada US$495; Europe/Rest of World GB£375**

To order your MIDDLEWARESPECTRA:

either
go to www/middlewarespectra.com
or call
(in the USA/Canada) 1-800-933-5997
(Europe/Rest of the World) +44 1962 878333

MIDDLEWARESPECTRA is published and distributed worldwide by:

USA and Canada:
Spectrum Reports, Inc.

Research Office
64 Hudson Street
Somerville, MA 02143, USA
Telephone: 617 628 8225
Fax: 617 623 4639

Subscription Center
PO Box 301368,
Escondido, CA 92030, USA
Telephone: 1-800-933-5997
Fax: (760) 432 6560

UK and Rest of the World:
Spectrum Reports Limited

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

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

Email and Internet

Email:
spectrum@middlewarespectra.com

World Wide Web:
www.middlewarespectra.com

ISSN 1356-9570