FINANCIAL MIDDLEWARESPECTRA

incorporating Enterprise Middleware

Contents

November 2001

Volume 15 Report 4

Cross border trade order routing and settlement using middleware

Marcus C. Consolini
Director of Operations
Tigerex

Management introduction

Tigerex is a privately owned company headquartered in Sydney (Australia) with operations in London, Hong Kong and New York. Tigerex is in the business of international order routing for securities transactions — essentially providing a single point of access to more than 40 markets, including the NYSE, NASDAQ, LSE, TSE, HKEX and SGX.

Marcus Consolini is Tigerex's Director of Operations. He joined Tigerex prior to it having an active business and his role was (and is) to:

- ***oversee the Operations of the company***
- ***oversee the design of the technology platform***
- ***institute the detailed back office procedures across the globe***
- ***once the technology platform was built, implement the technology into both clients and market execution brokers.***

In this case study he discusses the technology choices made, including changes made at the start and why these occurred. He also describes how the Sybase (New Era Of Networks) e-Biz Integrator message broker is used, as well as lessons learned.

Tigerex

Tigerex is a cross-border order routing company that was created and developed by two individuals who were the founders of the Trade Point Exchange in London — the largest private equity market in the world. After establishing the Trade Point Exchange (now known as Virt-X), they conceived the idea of building order routing conduits for international cross border securities trading.

Tigerex is the delivery of that concept. It provides two basic services:

- **the first is a technology-based service which is, in itself, a middleware system with links going out to clients and market execution brokers (MEBs — the likes of JP Morgan Chase, SSB and other such leaders)**
- **the second part of the business is introducing relationships between local brokers and MEBs.**

In practice what we are trying to offer is an end to end solution so that geographically local brokers can exploit the automated execution capabilities that Tigerex and the MEBs possess. It is the local broker which initiates a trade on behalf of its client. It is the MEB which:

- **receives that client's instruction**
- **executes it in the appropriate market.**

Customers

Our customers come in two flavors. There are the B and C tier brokerage houses that tend to operate in a local domain, rather than on a global basis. Their need is to be able to execute securities transactions on exchanges and using instruments that are beyond their traditional, and authorized, locale. For example, we have brokers in Hong Kong who want to fulfill their clients' orders against securities being traded in London or New York or Sydney. Without Tigerex, they would not be able to trade direct.

Previously, these local brokers, or brokerage houses — pre-Tigerex — have not had access to international markets, other than by the traditional tedious and long winded method of picking up the phone, sending a fax, going through transfers of funds between banks and other conventional processes. But this takes upwards of 20 days from initiation through to clearing and settlement. This is not good enough for the clients of those local brokerages. It leaves them (the local brokers) at a disadvantage compared to the Merrill Lynchs of the world, which have their own offices in most markets as well as the capability to drive straight through processing (STP) between their offices and markets.

The second group of customers are the MEBs themselves. Their interest is simple — additional trade volumes to exploit their investment in trading and settlement systems.

The Tigerex client is, therefore:

- **on one side — the B or C tier broker which wants electronically to trade securities on remote markets but has not had this capability**
- **on the other — the A tier brokerage houses (the MEBs) which can provide the execution and want the extra traffic.**

What we are not ... and what we provide

We are not, however, an exchange. Nor are we an ECN. Nor are we a broker channel.

What we offer is the technology to enable transactions to take place and the initiation of the relationships between the client and the MEB. The trade relationship — between the brokerages on each side of a transaction — exists outside of Tigerex, between the B and C tier brokers and the MEB.

Instead, what we do is:

- **'introduce' the parties**
- **supply and install the technology to assist the trades to occur.**

We provide electronic communication between the B or C tier brokerage — which originate the trades — and the A tier ones, which provide the execution. Both sides win. The B and C tier brokerage houses deliver more and faster services to their clients while the A tier brokerage obtains additional, automated trades, which they might not otherwise have received.

Finally, by using automation, we streamline the whole message control and trade transfer process. In effect, by plugging into the Tigerex service, our customers (at each end) introduce STP into their operations, thereby reducing their costs. Plus, we provide external liquidity outside local markets.

The Tigerex business model

Our business model is based on electronic communication using industry standard protocols, like the FIX message format. Orders are routed between the originating local broker and MEBs. They are delivered on an automated straight

through basis once the originating message has been appropriately formatted at the source. The advantage of this is that, on arrival at an MEB, the order (depending on the market) can be passed electronically into the market for execution.

As we are not a party to each transaction, we obtain our income by charging a fee per executed trade. So long as there is at least a partial fill of an order, we regard this as an execution of trade — and a charge is made. (All subsequent partial fills are not charged: we are paid by execution, not per message.)

Thus, if you are a broker in Hong Kong who has a client who wants to buy 1,000 Rolls Royce shares in London but had no previous access to London except by the old methods, you would sign up with Tigerex. Once you were established (and using the Tigerex Network) you would place the order through Tigerex which goes to a Tigerex MEB in London. The MEB would purchase the 1,000 Rolls Royce shares and pass the settlement and ownership details back to you, the broker in Hong Kong.

The transaction relationship is between you in Hong Kong and the MEB in London. The core of Tigerex's business is the movement and control of the messages which support each transaction from Hong Kong to London and from London back to Hong Kong.

But this is not all we do. We also establish the business forum whereby you, the Hong Kong broker, and the London MEB can meet and exchange mutually comprehensible and reliable communications (messages). The important factor here is that we are not just about technology. We provide a financial service which 'introduces' brokers from different parts of the global financial community to each other.

I guess another way to describe us is that we are a financial services company that specializes in technology. If you are a broker in Hong Kong which has never dealt with London or New York (other than by traditional paper-based means), we open the door to mutual electronic communication. This is different because:

- **financial technology houses do not generally make the business introductions**
- **financial institutions do not usually want to provide the technology.**

The reason we can do this is because of the diversity of relationships which our founders and owners have — and the respect they enjoy globally.

The technology challenges

Setting up the Tigerex technology platform has been a fast and furious operation. Before I came on board, in November 2000, I was living in New York City working on Wall Street for American Express Consulting Services. When I arrived here in Sydney, we had a technology partner who was in line to build what — at that time — the Company had thought to be appropriate for the services we would be offering.

My first step was to undertake an immediate review of the plans — primarily because I (and other recent joiners) were basically newcomers to the scene. Not only did we review what was planned but we undertook an immediate evaluation to determine which were the best vendors out in the market place that had solutions appropriate to our needs.

So, even though we joined up with a plan already in place, our evaluation suggested we needed an alternative solution. Our analysis indicated that the original middleware component was not ideal, and we were looking for a best solution. Equally, it was apparent that the intended provider's expertise was more concerned with front end integration than with message management. Yet, without message management, it was not clear to us that the front end integration would ever occur. Looking back, the initial plan was inappropriately focused rather than wrong — and we will probably use that original organization for a different, subsequent, part of our future systems.

The first challenge was, therefore, to figure out which vendor should assist us — outside of our own internal development — to build the foundation middleware messaging platform on which we could then implement the Tigerex system. We did nothing profoundly different to normal here. We produced an RFP. From this we created our own vendor matrices to determine, based on our requirements, which vendor that we had contacted best satisfied our needs. Those needs covered a variety of requirements, from the ability to support us internationally through to technology specifics.

Three vendors emerged from this process and we began negotiations with the top three to see what the relative price points would be. From there we were able to select the vendor that suited us best from both technological and business viewpoints. This turned out to be New Era of Networks (now part of Sybase) with its e-Biz Integrator.

Technological attractions

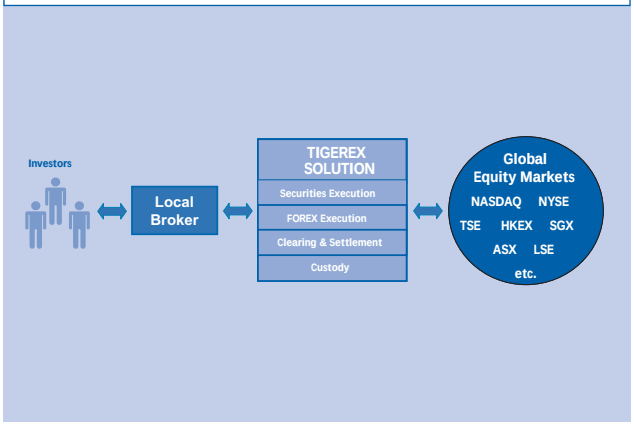
e-Biz Integrator had a couple of key technological attractions for us. Most importantly we had made the decision

that all messaging that we would be conducting would use the FIX format. (We had also decided that we would not cover, at least initially, post-trade communication — like that associated with dividends, stock splits, etc.).

Indeed, the original attraction to us of New Era of Networks centered on its FIX solution. At that time we were not actually looking at e-Biz Integrator as a middleware platform. Indeed, we were looking at another vendor to provide that part of the messaging middleware.

In our discussions with New Era we discovered the e-Biz Integrator and its capability to reformat and route messages. We then realized that New Era not only had the FIX capabilities but that these would work with the e-Biz Integrator message broker. It did not take us long to see that one vendor could supply both requirements — FIX and message routing/reformatting — and thereby provide the middleware foundation (Figures 1.1 and 1.2).

Figure 1.1: Middleware foundation I

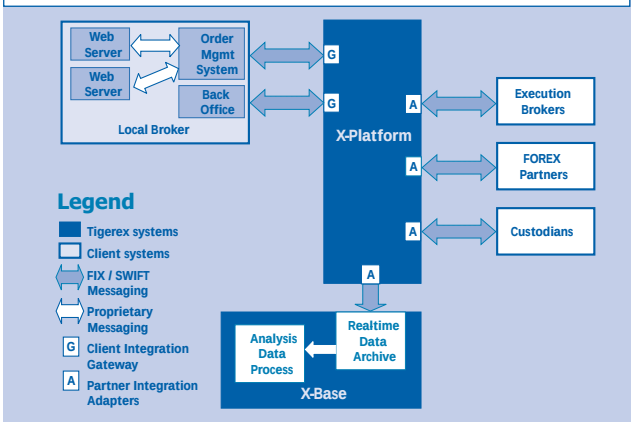


In addition, New Era's support for at least one industry standard messaging model — in this case IBM's MQSeries — was significant. The financial sector uses MQSeries extensively. It is familiar with it and it (MQSeries) is perceived to offer reliable, once-only transport of messages. By selecting a vendor which exploited MQSeries, we knew we were making adoption of our solutions by our customers easier.

So what we bought into were:

- **the FIX Adapter, from Sybase/New Era**
- **the e-Biz Integrator with its reformatting and rules/routing capabilities, from Sybase/New Era**

Figure 1.2: Middleware foundation II



- **MQSeries, from IBM, as the guaranteed message transport**
- **the Internet (of which more later).**

Using the messaging middleware and message broker

One of the great advantages for us of the New Era solution is that it enables multiple different shapes of solution, depending on the type of environment and customer. For example, we have the choice — going from the simplest to the most complex extreme — of:

- **only having one central e-Biz Integrator, through which everything goes**
- **placing e-Biz Integrators as concentrators/hubs in each geographical location (say Hong Kong, Singapore, Sydney, London, etc.) — linked to multiple customers in these geographies**
- **locating an e-Biz Integrator on each customer's site**
- **any combination, or variation, of these.**

In practice the first — the centralized solution — ran the risk of exposing us (and our customers) if there was a serious hub failure. That was not acceptable.

The third, locating an e-Biz Integrator hub on each customer's premises, is really only an option for those customers which have high volumes or already have an e-Biz Integrator (or MQSeries Integrator) installation. Typically, these are most likely to be found at MEBs in the major New York, London and Tokyo markets — rather than in B or C tier brokers. Of course, the latter could generate sufficient volumes to warrant their own e-Biz Integrator platform. We would be delighted and this is helped by the fact that

e-Biz Integrator runs on a variety of operating systems, from low end Windows NT/2000 through to AIX, Solaris and even mainframes.

What we decided to do was introduce the second approach. We place an e-Biz Integrator hub in each geographical location where there is sufficient justification. These hubs are then interconnected by our own private network, over which the messages flow.

This has several advantages:

- **customers only have to make a local call, to a dedicated number**
- **response times are short and fast (no long line delays)**
- **multiple hubs provide some measure of redundancy (if one hub is down it does not bring all connections down)**
- **we can work with each customer on an individual basis; this matters where (for example) they might not natively support FIX and we have to include specific reformatting and rules to put messages into FIX for transmission out (and then from FIX back into the customer formats on the way back to the customer's system)**
- **it can work with customers having their own e-Biz Integrator hub.**

We can also use different transport protocols from the customer to the local hub, although all hub-to-hub communication — and most hub to MEB communication — is in FIX format. For example, some customers wish to generate messages in FIX format which can be carried by MQSeries to the hub. Others, in contrast, already use FIX engines or wish us to receive a file or message and then convert to/from FIX for them. All are possible and we manage to exploit the fact that FIX possesses two layers, a session layer (for handshaking and log-on/off) and an application layer (for the orders, etc.).

We go one stage further. We support a browser connection to our hubs. The customer, over a secure connection, uses a Web-based user interface to submit trades to Tigerex. We accept the trade and format it into FIX before transmission to the destination. This 'Tigerex Direct' (as we call it) interface enables real time trading, albeit with limited reporting and administrative functions. It has the additional disadvantage that it does not provide STP integration to the customer's front or back office systems. On the other hand it attracts; there is no need to undertake all the internal systems integration effort that STP demands.

Downsides, and achievements

The only downside which we worried about was that locating hubs on a dispersed basis meant that we would need to have people in many places. Here Sybase/New Era has assisted us. It provides professional services where we need additional resources on the messaging foundation, and its related formats and routing. We can contract it to provide integration to customers' existing systems where they need it. Although we know our part of the solution well (from our development HQ in Sydney), Sybase/New Era knows its own software better than we do (at least for the moment).

Having the option to obtain additional and technologically competent resources at short notice is valuable. It applies particularly where we need to map new inputs and outputs into the e-Biz Integrator Formatter. Furthermore, such mapping tends to be a one-off exercise: once captured it can be re-used (by us and by Sybase/New Era).

On the other hand, the one area we reserve to our own people — for obvious reasons — is that which is specific to our solutions and clients, whether B or C tier or MEBs. And the measure of our success, if I may put it that way, is that we have gotten up and running in less than 8 months.

Size of operation

In Sydney we are about 21 people, mostly development but with a local business operation of about 6 (of that 21). We have already a presence in Hong Kong and London — each with about 4-5 staff. In addition to these we have regional directors who represent the company in our target markets, and make the introductions between the B and C brokers and the MEBs. They smooth the way forwards.

We do not anticipate needing to add hugely to these numbers. One of the attractions of our business model is that, once an MEB or broker has become a customer and his or her systems have been linked to our e-Biz Integrator hubs — then the traffic flows. Yes, we see ourselves expanding into New York, Tokyo, San Francisco and other places but only a few staff are needed in each, once the connections are made.

Indeed, if you think about it, we offer a simple service. We do not put any software on anybody's site (although we can). Our customers format FIX messages which they communicate to our hubs. We look after the connections from there on. That is our business and we can usually have a new customer up and running, from start to finish, in 4-6 weeks, or less (see Figure 1.3); and this includes connecting direct from/to their systems to our hubs plus the contractual arrangements between all parties.

Tigerex futures and settlement

We have not stopped at order execution. We do offer foreign exchange capabilities. But even more importantly we have introduced post trade clearing and settlement — normally the killer for securities. What used to happen — primarily because the transaction relationship exists between the B or C broker and the MEB — was that the MEB's existing settlement process was used, even if this meant faxes or email or letters.

Lessons learned

The first lesson is that we were right to undertake that review when we arrived. The original plan was not likely to deliver a result as resilient or flexible as that we have now implemented.

We have proved that you can create an order routing network with off-the-shelf software and that applying a FIX format works. We were on time and budget with delivery.

Figure 1.3: Adding a new customer

STAGES:	1. Discovery (2-3 weeks)	2. Development (6-8 weeks)	3. Testing (2-3 weeks)	4. Support (ongoing)
Planning	Review business requirements Determine general implementation timeline	Present and review Tigerex client kit [2] Provide oversight and support for integration issues	Identify testing requirements Provide oversight and support for testing issues	Notify of all system or integration failures Communicate functional goals and features for upgrade
Operations	Dedicate project staff Review technical requirements Outline integration issues Approve feasibility of general milestones	Adapt systems for international trade capability Install communications link to local Tigerex POP Integrate to X - system	Develop testing model Set up availability for testing Conduct test	Provide 24/7 customer support Maintain system and issue upgrades
Contractual	Sign LOI / MOU Discuss Tigerex contract details Discuss third party [1] contract details	Complete contracts	Provide approved Tigerex certification	Contract with additional execution partners

Notes:

1 Third party contracts include those for market execution brokers, custodians and FOREX partners. These contracts are standardized by Tigerex and are tailored to client requirements. A price ceiling can also be negotiated.

2 The Tigerex client kit includes white papers, APIs, architectural information, use cases, integration requirements and standardized third party contracts.

This opened up a whole different ball game for us. Today the messages that the MEB sends back to the client for post trade settlement can actually go through our links. What we have designed is a system which is S.W.I.F.T. enabled. We have filed for a S.W.I.F.T. Bureau Licence and hope to become a S.W.I.F.T. Service Bureau Member soon.

This has many implications. Not only will we be able to handle S.W.I.F.T. formats (using the Sybase/New Era S.W.I.F.T. Adapter running with e-Biz Integrator) but we will be bringing the strengths and virtues of S.W.I.F.T. to those of our clients who may not wish to incur the heavy integration costs associated with S.W.I.F.T. Tigerex will provide the integration to the S.W.I.F.T. network and our clients will receive the S.W.I.F.T. messages from us.

The result is that our local brokers can receive MEB-issued end-of-day trade confirmations for each order executed. The configuration is issued in the currency in which the security was traded — with the delivery options as varied as S.W.I.F.T., OASYS, fax, email or even 'plain old' telex. In the future we are even thinking of how to accommodate any third party custodian relationships that the local broker has in place.

From a software tools perspective we have had the usual unexpected issues you encounter with all software — but nothing that halted us. We are pretty satisfied.

We also proved one dimension to ourselves. In all our working with third parties we had two estimates — the one that everyone knew and the one we budgeted for and scheduled. Consistently, our internal scheduling (at about twice what others estimated) was correct. Fortunately we had anticipated this and were not caught out.

Management conclusion

Tigerex demonstrates what can be achieved, and in a short time, with a message broker. In the Tigerex instance this was not broking between applications within one organization but between applications running in multiple organizations (the MEBs and local dealers/brokerages).

In addition, Tigerex supports financial instruments. It is about 'money'. If an organization can deliver such capabilities securely and reliably using a middleware platform like e-Biz Integrator (and its associated FIX and S.W.I.F.T. adapters), other organizations can reasonably have the confidence that they could achieve the same.

Automating error correction and repair at HypoVereinsbank

Michael Wiedemann
Head of Operations and Technology
Regional Service Centre Europe
HypoVereinsbank

Management introduction

HypoVereinsbank (HVB) is the second largest bank in Germany and is headquartered in Munich. Globally, it ranks about number 15 — based on assets. It has about 72000 employees worldwide, mainly located in Europe. The retail side is concentrated primarily in Germany, Austria (where it recently purchased Bank Austria) and Poland. It has wholesale financial operations in the major global centers — like London, New York, Tokyo, Hong Kong and Singapore.

Michael Wiedemann is the regional director responsible for the London Regional Service Centre, which also supports operations in six other European countries — Greece, Italy, France, Ireland, Lithuania and Latvia. The Regional Service Centre provides the operational and IT support for the Bank's activities in this geography.

In this interview, Mr. Wiedemann discusses the Bank's use of the PaceMaker work-flow monitor as a means to monitor transactions as they move between front office and back office systems. As he describes, PaceMaker now plays a significant role in reducing errors and failed transactions as well as improving their repair and successful resubmission.

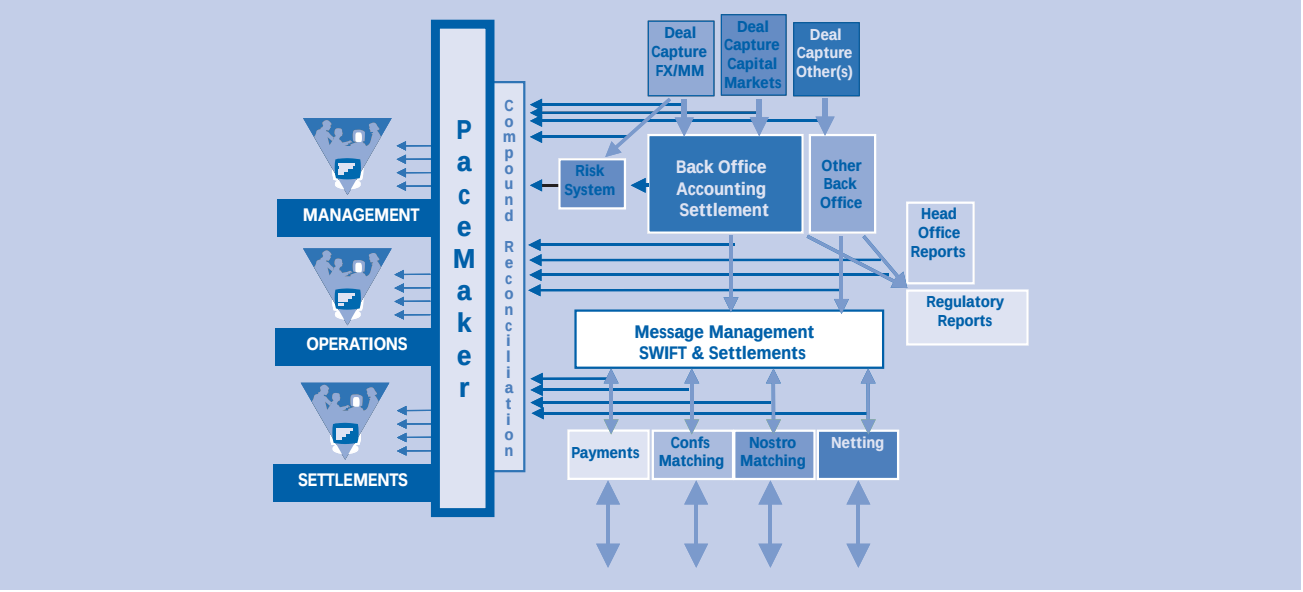
Our business challenge

The basic business issue that all operations centers face is that, as you introduce more and more data processing across ever increasing numbers of different business systems, it becomes ever more difficult to keep databases synchronized. If the data is processed straight-through or manually — the challenge is the same. Why does this matter? First of all, data inconsistency among the various front and back-office systems increases operational risk.

What we need is for our various systems to work together, and to be synchronized. We want to have STP, from when the front office enters the deal or trade through to the back office including settlement (Figure 2.1).

Historically, however, much of the processing has had a manual element. This can result in the same data needing to be input into multiple systems. There is an acute need, therefore, to check for internal processing errors. This is most important before deal or trade details are sent out as confirmation of those deals and trades. The (fairly obvious) reason is that the quality of what you send out has a major impact on the speed and accuracy of subsequent activities.

Figure 2.1: Front to back office at HVB



The second reason is that the German regulatory authorities now require its banks to ensure that information, and thereby the databases, are synchronized on a regular basis. Indeed, that is the minimum requirement. Ideally, we want to ensure that databases are synchronized on a real time basis, in order to minimize risk exposure across all of our activities.

HVB systems

Like most other financial institutions we have a wide range of different systems. The major ones here in London include:

- **Kondor+, for foreign exchange, money market and liquidity management**
- **INTAS, for the remaining interest and derivative products**
- **MIDAS, for back office, accounting and regulatory reporting.**

We want to send out accurate confirmations to our counter parties so that the matching is faster and more accurate. The automatic matching of outgoing and incoming confirmations completes the cycle.

Front and back office differences

So, returning to the business problem, our real issue lay between our front office and our back office systems (between our back office and counter party systems there are other confirmation checking products and processes in place). Although it may seem improbable that your front and back offices might not be aligned, the reasons are relatively simple to understand. If inputs are made manually, human errors do happen. Even if inputs are delivered via

straight through processing, system errors or data mismatches can occur.

The reason for still performing manual input is that many of the financial instruments in which we deal are complex, specialized and do not necessarily happen continuously. It is often not worthwhile, for the small numbers of these deals, to build an STP sequence — the volumes are not sufficient to warrant the expense of automation (even though some inaccuracies may occur when manual input is undertaken).

In such instances it is our experience that mistakes of understanding happen. It is highly desirable that these errors be caught early, and be corrected.

On the higher volume instruments, a different problem arises. Put simply it is that 'obvious' mistakes are difficult to pick out when you have a mass of similar deals. Yet such errors cause subsequent problems of settlement, including delays and their associated costs.

Finally, even though we have already automated many of the links, the reality is that these do not necessarily work all the time in all the ways expected. Attention is required if we are to be both consistent and in line with market requirements.

Introducing PaceMetrics as a middleware monitor

I have described the basic problem — consistency. But a different way of characterizing the problem is that we want to have exceptions identified and resolved as quickly as possible.

To achieve this we started to write our own solutions in-house, particularly for our high volume products. Our efforts here were successful, but it taught us that the effort to deliver and maintain such solutions is great.

Then we came across a company called PaceMetrics, at a trade show. PaceMetrics offered a 'middleware' product (PaceMaker) that had the same objectives as those we had built internally — except that it was designed with a friendlier and easier to use interface. This attracted us because we felt it could automate the examination of deal information and visually show on screen:

- **the deals that matched**
- **the ones that did not match**
- **the ones where there might be possible mismatches.**

It also possessed drill down capabilities. These enable us to see errors from a very high level down to (say) the individual fields in a trade or deal.

So we started to investigate the PaceMetrics offering. Our justification was that we already understood how expensive it was to build and maintain an in-house solution. In addition, PaceMetrics looked as if it might provide a tool which would enable us to keep a combined database to hold information across all the systems in one place.

What PaceMaker does and how

The way PaceMaker works, at least for us, is as follows. Its database runs on a separate RS/6000 system. It is fed by a mix of communications. These range:

- **from database extraction programs that run at regular intervals, for example every 5, 10, 15 minutes (using traditional client to server communications)**
- **to real-time links, updating PaceMaker as journals are updated.**

Now think of a simple scenario consisting of:

- **a front office system (System A)**
- **a back office system (System B).**

The way that you typically connect two systems is to use some form of middleware layer — like a publish and subscribe messaging mechanism. The objective is that what originates on the front office system should be transmitted correctly to the back office system.

What PaceMaker offers is intermediate work flow processing (Figure 2.2). You establish a data feed to/from each of the systems you want to have checked. This data feed is mapped into the PaceMaker database, including which fields you wish to have checked, plus any calculations or translations that you might require.

In addition you state the time interval in which you expect activities to occur. For example you can set the interval time between when a deal is input on System A and by when you would expect that particular deal to appear on System B. You can also set sequencing — the order in which you expect particular actions to occur.

With transactions defined, you can enter a deal in the front office. If you go immediately to PaceMaker, you will see that there is no match as yet in the back office system (the deal may not have made it across the communications link

between System A and System B). But, as long as you are within the expected time interval, that transaction will not show up as a problem — it will just show as a 'yellow', meaning that something is waiting to happen.

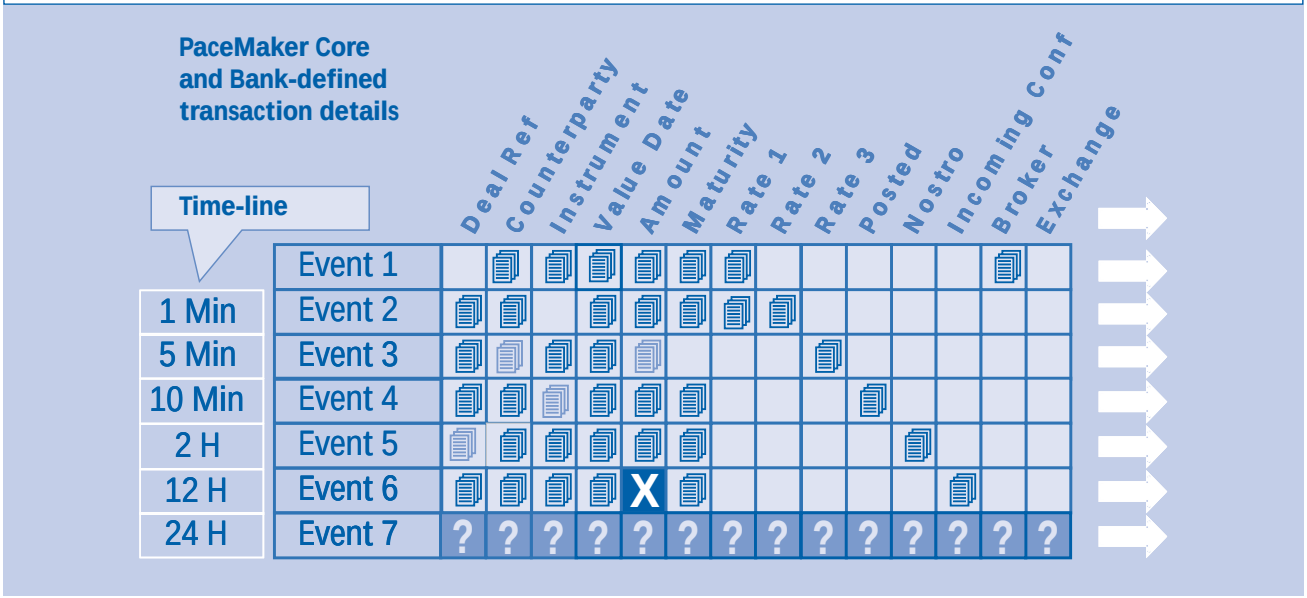
If, however, the deal does not reach the back office system (for whatever reason) within the expected time — or there is a mismatch between the data sent by System A and what System B is seeing — a red alert indicator would light up (if all had been processed as expected OK, a green would show). You can then click on this (or any) transaction to discover more detail, right down to the specific field level.

Back to HVB

What I described above encompasses what PaceMaker can offer in a generic instance. Now let us return to the issues at HVB.

As I said earlier we have two front office systems (Kondor+ and INTAS) and the MIDAS back office. Certain transactions are required to be recorded on both front office systems as well as the back office one. For example, we have interest-related instruments which include a cash component. Such an instrument bridges the two front offices. We need to hold all the parts if we are to maintain accurate

Figure 2.2: PaceMaker and real time compound reconciliation



In addition, PaceMaker holds a complete transaction history — including previous changes, time logs, past values and who did what and when. You can even see 'incorrect corrections' — remedial actions which were themselves incorrect.

What is so clever about this is that, with a PaceMaker system, everything published by the front office to the back office is 'inspected' and matched. Furthermore, the descriptions of the transaction types, and supporting details, are captured in the PaceMaker database.

By describing what the deal types are, and what you expect to happen, you are monitoring the state of the databases and catching those items where something is not happening or there is an error. In doing so you are reconciling your databases and ensuring the consistency of data.

positions.

Clearly, we need both front office systems to be aligned with each other, as well as with the back office. Equally, we want to discover mismatches, failures of processing or outstanding items as soon as possible. The attraction of a solution like that offered by PaceMetrics is that it can achieve this as well. In the generic example I described above, there was only one front office and one back office system. There is no reason why there should not be several front or back offices (or,any type of system) where monitoring is desirable in order to ensure that transactions are correctly sent and received.

Indeed, one could go even further (although we have not). It is technically conceivable that one could put in place a PaceMaker implementation between us and counter par-

ties — to ensure that both we and they received correct data. The difficulty is more technological than practical:

- **who would own the information and the responsibilities for chasing up failures?**
- **would we be able to see confidential details at a counter party, or vice versa?**

Such issues may be insurmountable in the near term. More likely is that, further down the road, we will use PaceMaker to monitor responses as these came back from counter parties — for example over S.W.I.F.T. or other networks. But this would be in-house rather than across enterprise boundaries.

How long and with what results?

We started some months back with a small implementation, applying it to a particular part of our derivatives operation. We chose to start here because:

- **the transaction volumes were not large, but they were sufficient so that we could monitor what was happening on a daily basis and there was sufficient potential for a payback**
- **the number of matching fields required would be limited and could be accomplished at a relatively low cost**
- **we could draw conclusions about how long it would take to add additional matching fields (front office and back office)**
- **we could check the results and ensure that what we expected was actually occurring**
- **we could still return to manual checking if PaceMaker did not work out for us.**

That initial trial was a success. We have now moved on, to incorporate all our fixed income and derivative products in PaceMaker. For Foreign Exchange and Money Markets we have not introduced it because this is where we had built our own internal system (thus this area has a lower priority for the moment, although we intend to include it eventually). Instead we are extending the derivatives and fixed income implementations to include various additional systems that we use. In so doing we are expanding the coverage so that we can see the status of our operations any time during the day.

In terms of headcount we have indirectly saved on staffing costs. This is because our volumes have been increasing. In some areas the volumes has more than doubled. Yet we still do our work with the same number of staff.

Lessons learned

Our experience has been positive. We have obtained what we wanted. But one reason that this was possible was that we could start small and grow: PaceMaker did not require us to convert everything all at once. Rather we could try out PaceMaker with a suitably sized implementation, observe the results and draw conclusions. In addition, we have been able to add to our implementation on an incremental basis — when and where it suits us and our business.

The second lesson is that staff satisfaction has increased. Looking at hundreds of transactions each day to try to find errors is not the most interesting of tasks. Today PaceMaker identifies the problems for us. Our back office people are happier now — because the focus has changed:

- **from searching for errors (not unlike searching for needles in a haystack)**
- **to actual exception processing.**

This is much more rewarding. You can see that you are making a difference.

I would like to make one additional point in this context of my lessons learned, although strictly it is more of an observation based on common sense. Earlier I talked about PaceMaker as work flow monitor middleware. We also use a different product (eFlow) for STP. This is deliberate. The reason why we prefer to use two different products is that, if we used only one product for processing and checking, the likelihood that any logical mismatches would be discovered would be remote. By using two systems — with separate data feeds and business logic — we enhance our control of operational risk. We think that this emphasizes an aspect which is often overlooked — the importance of having double checks or mechanisms to prove accuracy.

Management conclusion

Accuracy, consistency and completeness are almost always the objective in IT. In the financial world this is particularly important (even before you introduce regulatory requirements).

When you have multiple different systems which need to be kept in sync., middleware alone (as in messages or whatever) is insufficient. Furthermore, as Mr. Wiedemann points out, you also need a 'mechanism to check the mechanism' if you are to be certain that everything is correctly aligned. This is what HVB has achieved. Not only does it have eFlow for STP but it uses PaceMaker to ensure that deals are correctly processed on front and back office systems. These act as mutual double checks for accuracy.

Integration banking — using middleware to integrate a network-centric banking strategy

Mark S. Allcock
Senior Partner and CEO
TMC Alliance

Management introduction

In this third, of four, part series on the strategic use of middleware and integration technology within the finance and banking market place, Mark Allcock, CEO of TMC Alliance (which specializes in strategic integration consultancy for financial markets) discusses key service integration issues for the finance sector.

As will be seen, Mr. Allcock is passionate about the importance of addressing management and organizational issues as part of all integration projects. He believes these issues are as least as important as the technology and solution choices. At TMC ALLIANCE he has enabled a number of major financial firms successfully to complete business driven integration initiatives. He has direct experience with most of the leading technology solutions in the marketplace today.

Each analysis in this series builds on the previous one. It includes hints and tips plus comments on pitfalls that can be avoided.

Figure 3.1a: Financial life stages

Life stage	Consumers' financial needs
Pre-Birth....	Parents plan larger home Parents build nursery Parental school fee planning Parents enhance medical care Parental bank accounts
Birth....	Parent and child medical care Parents increase credit needs Parental bank accounts
Newborn...	Parents increase credit needs Children's nest egg savings Parental bank accounts
Infant/ Kindergarten....	Parents increase credit needs Children's nest egg savings Increased mortgage needs Parental bank accounts
Elementary School....	Parental bank accounts Parents increase school investment Parental home equity increasing Increase mortgage needs to accommodate larger family
Teenage years....	Parental bank accounts Teenagers' bank accounts Teenagers' savings Parental home equity increasing School fee investment draw down
High School....	Parents increase retirement asset provisions School fee investment draw down
College pre-graduation....	Student loans Bank overdraft Credit card
Graduation....	Car loans Car insurance Credit card
First job....	Car loans Car insurance Home financing

Setting the scene

In the two previous analyses, I described the planning and management guidance that we believe all companies should follow prior to commencing any integration project. For example, identifying key managers as well as undertaking project planning and assessing risks (preferably at the earliest possible stage in any major project) massively improves the prospects of on-time, on-scope and on-budget delivery.

Those two earlier analyses identified the key factors — that can turn out to be the difference between failure and success as being:

- **scope**
- **skills**
- **budget**
- **time**
- **plans**
- **risks**
- **rewards**
- **capabilities**
- **supply**
- **demand**
- **knowledge**
- **dependencies.**

Building on these, I will here present an alternative viewpoint on integration, one which highlights the role of technologies and vendor solutions but asserts that these must be applied in an appropriate setting for the consumer (rather than the vendor) of financial services. In essence I am arguing that a lifestyle oriented approach to integration is the key to creating a successful end to end integrated service for financial markets. Such an approach highlights the importance of the network, and makes the need for new XML message standards abundantly clear.

Integration banking design: wind in the sails

Today's financial market providers present a broad and deep range of product, service and investment opportunities to clients who are:

- **increasingly mobile**
- **decreasingly loyal to brands (in isolation)**
- **more loyal to excellence in service and performance.**

Clients are, therefore, demanding more, rather than less. Equally, consumers are more sophisticated with un-

paralleled access to information and knowledge about products, opportunities and risks.

Much of this is being facilitated by their access to the Web. The dynamics of our enterprise culture and global technological and communication advances — which is accelerating as we move into the 21st century — has created the opportunity for distributing wealth amongst an increasingly broad base of people spread across the globe.

Wealth management is a financial markets product principle that reflects the economic life stages of our families and their financial needs. Banks, investment management firms, supermarkets, retirement and life insurance companies all have a part to play in providing customer life cycle products and services to today's customers and clients (Figures 3.1a-b). This creates an incredible potential for those financial service firms able to identify, capture and retain the customer relationship over the customer's lifetime. Yet integration is often seen as a way of achieving this — by piecing systems together to create the notion of an integrated service or solution. The core pieces are certainly the same but this view is too common and constraining a view of integration. Integration is not a technology issue alone.

B2C, B2B, E2E really means C2G

True integration is about re-engineering business and product services around the life stage needs of customers, consumers and clients, irrespective of whether these are in a classic B2C or B2B market place. Many technology companies now talk about E2E (end to end or enterprise to enterprise) initiatives. While such discussion is a start, a quantum leap is still required to re-orient solutions:

- **around the financial expectations and needs of customers, the consumer, throughout their life time**
- **rather than around 'what the business would like the customer to want'.**

No one organization has the capability to provide such a range of services and products. Yet, some are close. To us the Swiss banks have come closest. They have evolved — in our analysis — into one possessing the broadest product and service capabilities. This is, in effect, the kernel of a true integrated C2G (cradle to grave) solution.

Our analysis is that those vendors who can best be expected to provide C2G solutions are unlikely to be software technology companies. No matter how capable their products, they simply do not possess, and cannot provide, the combination of industry-specific experience,

Figure 3.1b: Financial life stages

Life stage	Consumers' financial needs (cont.)
First job (cont.)....	Credit cards Store cards
Career development....	Investment portfolio based on liquid vanilla products (for example mutual funds) Venture capital/business finance and extensive credit card finance for entrepreneurial types Significant mortgage Tax efficient retirement portfolios and pension wrappers Major wealth creation
Career established....	Mid-life crisis Wealth enhancement Portfolio of diverse and complex assets Car loans Divorce settlement
Career maturity....	Significant home equity value Inheritance income Wrapper investment assets to risk and tax balanced portfolio Establish family and retirement trusts
Pre-retirement....	Mortgage debt discharged Maximize investment opportunity Downsize home Career emphasis reduces
Retirement....	Release lump from retirement portfolio Health care provision Partial income draw down Transfer assets to family/trust Prepare for succession
Old age....	Release home equity Income draw down to support healthcare and attended lifestyle
Next generation inheritance....	Assets transfer to family and friends Family and friends commence inheritance investment

products, services and delivery platforms needed to provide true C2G solutions. [This explains their urgency in trying to set up relationships with systems integrators and other professional services organizations.]

The truth is that delivering C2G is about issues that are very different from producing great software products. In our experience, C2G involves an expertise and awareness which only organizations routinely serving consumers every day can achieve.

That is not to say that great technology companies do not have a major role to play in providing C2G solutions: they do. But success in C2G delivery requires much more than the rush we saw in recent years when major consulting and business application companies (such as Accenture, PwC, KPMG and others) took equity stakes in technology firms and surrendered their independence. C2G, in our opinion, is being driven by the acquisition of integration technology companies by those actually providing the products and services — banks, insurers, brokers and others.

In the same way that the major name business application vendors saw an opportunity to broaden their product portfolio by acquiring application components (and then integrating them with middleware and EAI solutions), so the C2G leaders are understanding the need to integrate products and business divisions to support all the life stage needs of a customer.

Financial life stage needs

Before looking at the design context for a C2G integrated banking system, let us first look at the financial life stage needs of a typical financial products consumer. This has been simplified for the purposes of this analysis but, as can be seen from Figures 3.1a-b, there are many steps with quite different implications.

By analyzing the life stage needs of consumers, the banks, insurers, brokers and financial service companies have discovered that they can understand the breadth of the true integration needs of their business when they look at the problem from a customer's perspective. It is this which should drive the information, IT infrastructure and application component requirements needed within and across organizations — and their integration.

Set against an increasingly mobile set of consumers, one can see (in Figure 3.2) how various product providers need to acquire or form collaborative integrated relationships if they are to provide global financial services set in a 'whole life' context.

Client-driven C2G integration through Web Services, XML and P2P

In our experience the companies that will succeed in the C2G market will be those that really appreciate how to realize the capabilities of their products and services when these are harnessed around all the future life stage needs of their clients. This requires integration and collaboration between businesses on a larger scale than we have today.

By looking outside the financial sector, companies can see how such conceptual initiatives have been successful elsewhere. The manufacturing and automotive industries are showcase examples of where supply chain management concepts have been extended to create cross company collaborations between both major and minor companies. All are harnessed through a business driven need to optimize the flow of components and information to manufacture the products as customers need them.

This, we believe, is the true meaning of integration. And, in that context, we can set out what the financial sector must do to satisfy life time integration:

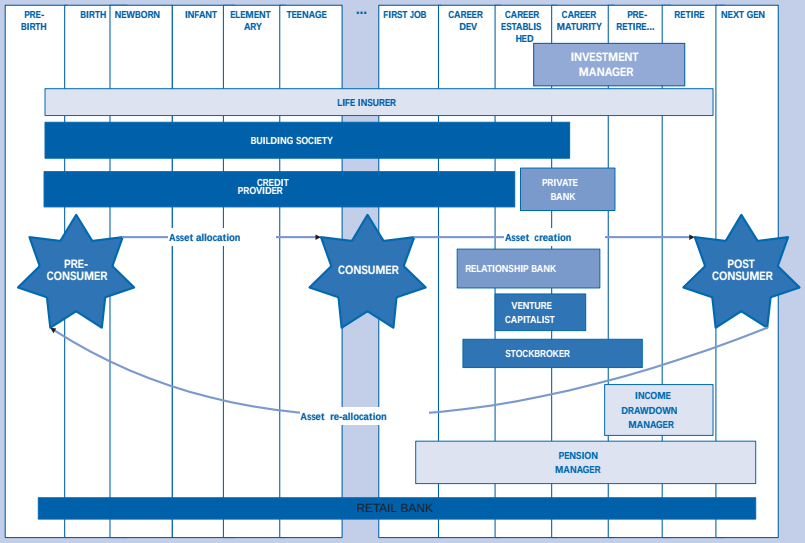
- **provide business components which offer the means of capturing information from consumers in order to supply them with a breadth of financial products**
- **understand a consumer's life stages, as well as profiling of current and likely future financial product needs**
- **offer a broad range of service and delivery channels to enable each consumer to interact with one or more providers without regard for location and the communication medium**
- **choose technology that is able to integrate provider companies on a global basis**
- **adopt global standard message formats that enable a wide range of providers to bid, collaborate and provide information, products and services to consumers**
- **establish an industry body that is sufficiently funded by commercial banks, assurors, life companies and brokers to create a cross sector L.I.F.E.T.I.M.E. message standard (probably based on XML) and which promotes member participation (akin to a personal S.W.I.F.T.)**
- **take small but progressive steps to success rather than one huge effort to try and map all life stage message exchanges**

- **introduce cross industry initiatives led by commercial companies to bridge the divide between existing standard message formats used by sector companies; this should be an interim step while a L.I.F.E.T.I.M.E. format is created (for example L.I.F.E.T.I.M.E. to S.W.I.F.T., to FIX, to ORIGA, to OFX)**

required in all cases is the drive to change your organization and evolve it to the new cradle to grave (C2G) picture.

In this context, middleware and integration solutions are critically important to the financial sector. But, in our experience, this is only true when they are applied 100% to meet the needs of the business strategy — in this analysis

Figure 3.2: Integrating financial relationships across 'life'



- **entice technology integration providers — with broad enough offerings to justify acquisition by provider companies (for example, SeeBeyond, Sungard, IBM, webMethods, etc.)**
- **identify a leader who can bring together all the constituent standard bodies and industry initiatives and create a groundswell of activity that bridges both business providers and consumers**
- **implement an application of peer to peer (P2P) technology that focuses on what (we believe) should really be termed Person-to-Provider.**

The relevance of middleware for integrating a network centric banking strategy

What I have sought to present in this analysis is a view of a globally integrated financial solution that will meet the needs of consumers (focusing on the individual as an example). Equally, application of the same concepts can be applied to commercial and corporate finance. What is

the I have used the 'lifetime financial business strategy' as the example. In creating an overall integrated solution, middleware and associated products are more than technology delivery tools. They enable vision and free thinking. In a sense it is middleware which puts wind into the sails of belief, so that a strategic vision can be created and delivered.

Management conclusion

What Mr. Allcock presents here is a view of a globally integrated financial solution that meets the needs of consumers — by concentrating on the individual, as his example. The same principles can be applied just as well to commercial and corporate finance.

In his view, what is required in all cases is a desire to change, to drive financial organizations forward. In his vision a new order will evolve — one that will be described in the final part of this series as he brings together all the pieces in order to describe how you should go about obtaining success.

Addressing pitfalls with EJB development and deployment across different J2EE application servers

Dan Rolnick
Senior Consultant
Cacheon

Management introduction

J2EE is the great technological hope for those who do not wish to follow the Microsoft .NET path. Application servers are all the rage. Over 50 different products are available. Competition is intense. The common assumption is that, if you deploy Enterprise Java Beans (EJBs) built using two or more J2EE application servers and development tools, they will work happily together.

Sadly, this is not necessarily the case. Indeed, J2EE-based application servers have — as most pessimists anticipated — specific incompatibilities. What you build for deployment on (say) iPlanet will not necessarily run on WebSphere or WebLogic or Oracle 9iAS ... and vice versa.

In this analysis, Dan Rolnick explores the dependencies that arise when developing an EJB for deployment on a J2EE compliant application server. He examines how vendor-specific implementations of low level services and entity bean persistence, as well as proprietary deployment descriptors, can irrevocably tie an EJB to a particular vendor's application server. He then looks at possible solutions.

The practical, business context

To remain competitive, enterprises must balance business and technology challenges. A primary challenge facing many enterprises today is the ability to develop rapidly and then deploy new business applications and services in response to changing customer needs.

The need for this — and the fact that not all Enterprise Java Beans are created equal — was brought home to us on a implementation project at a leading international financial institution. This institution wanted to create a new business service in response to its customer demands.

Previously, its customers had to log into:

- **one account application to access a current account balance and bill payment information**
- **a separate brokerage application, in order to access portfolio balance and loan availability information.**

But these same customers wanted to log into one application that would provide them with a consolidated view of both their bank and brokerage account information. To satisfy this, the financial institution needed to do something if it was to avoid alienating, and losing, customers.

In response, the financial institution accepted that it needed to create a new business service. For operational and financial, as well as common sense, reasons the preferred approach was to combine functionality that was already offered in the existing bank and brokerage applications in order to deliver a consolidated view of account information.

What the financial institution knew was that, by leveraging existing application components (to deliver the new service to its customers), it would obtain a competitive advantage (however briefly) because it would:

- **decrease the time to market for the new business service**
- **increase revenue generation as a result of the new business service**
- **improve customer satisfaction and retention.**

However desirable these were, as a set of justifications, the financial institution also knew that it faced a daunting prospect — the integration of components that had been built on two (or more) application servers and/or legacy production environments. Furthermore, these applications had been developed by different vendors and different teams at different times using different technologies. Integration, especially of the application server elements (on which I will focus) was not going to be easy.

Setting the EJB/J2EE stage

In the financial institution, the 'opportunity' involved:

- **a banking application deployed on an Oracle 9iAS application server**
- **a brokerage application deployed on a BEA WebLogic application server.**

As will be discussed, the problems arose because the applications and services ran on heterogeneous platforms. Despite the fact that both application servers were hosting J2EE-compliant EJBs, they were not easily integrated.

The reality is that EJBs are developed for deployment on a specific application server. These require modifications before they can be deployed on a different vendor's application server. This is because some of the code developed — the tools used for creating EJBs — establish a dependency on a specific vendor's application server.

Because of these dependencies, moving an EJB between different J2EE compliant application servers produces deployment problems. For example, deployment issues arise when trying to move an EJB developed for (say) a BEA WebLogic application server to an IBM WebSphere application server.

Additionally, EJB to EJB communication across J2EE compliant application servers can produce run time problems. For example, run time issues arise when an EJB deployed on (say) a WebLogic application server attempts to communicate with an EJB deployed on (say) a WebSphere application server.

Dependency details

Most of the functionality available through the Enterprise JavaBeans Application Programming Interface (API) is included in the javax.ejb package. This package primarily consists of interfaces. These interfaces are implemented by each application server vendor, each one of which is free to interpret and implement them as they think best.

Inevitably, differences occur. The EJB interfaces define the services provided by a particular container. They also define the client interface to an EJB.

For example, the container is responsible for managing the interactions between an EJB and the application server. Yet

each application server vendor implements the services provided by their own application server container. These include services such as:

- **security**
- **transactions**
- **multi-threading**
- **distributed programming**
- **connection resource pooling.**

The worthy objective of J2EE and EJBs is to free developers from having to be concerned about such low level services. If they do not have to be, they can focus instead on developing the EJBs which represent specific business logic. These EJBs can then make use of the functionality provided (to the EJBs) by the low level services.

The problem is not with the principles. An application developer can utilize the services provided by an application server container. The challenge lies in how these services are implemented by each vendor.

Difficulties arise because EJBs developed for deployment on one application server have dependencies on that application server's implementation of these low level services. Yet, in order to take advantage of the benefits provided by such services (not least, protection from the low level considerations), developers must implement EJBs in ways that make use of the services as implemented by that vendor.

The result is that EJBs are created which depend on the underlying vendor's implementation of the key low level services. Consequently, if you attempt to move EJBs between different J2EE compliant application servers — or even try to have them work together while running on different applications servers — incompatibility problems occur.

Other problems: client interfaces

Similarly, other problems occur with client interfaces to an EJB. The EJB architecture allows clients to invoke an EJB over a network via the home interface and the remote interface for the EJB. The home and remote objects handle the communication between a remote client and an EJB deployed in an application server container.

Although many container implementations handle communication using Java RMI, the EJB specification does not mandate which distributed object protocol should be used for communication between a client and the container managing the remote EJB. As a result, vendors have latitude in how they choose to implement the communication between a client and an application server container.

Furthermore, clients use the Java Naming and Directory Interface (JNDI) to locate and access a remote reference of an EJB on an application server. This requires the instantiation of an initial context object, which provides the starting point for the JNDI lookup. The initial context factory is part of the JNDI API and requires a developer to set properties that are specific to that vendor's implementation of JNDI. The type of initial context object which is instantiated varies, based on each application server vendor.

It turns out that each vendor has unique JNDI properties that must be populated. As a result, the initial context object used for locating an EJB deployed on one application server cannot necessarily be used for locating that EJB when deployed on an application server developed by a different vendor. Therefore, an EJB cannot call another EJB deployed on an application server developed by a different vendor without encountering run time errors.

Persistence issues and entity beans

Additionally, the tools available for configuring the persistence properties for entity beans which use container-managed persistence are almost invariably proprietary. They are included as part of the application server software.

These tools matter. They are used to:

- **generate tables from entity bean objects**
- **map container-managed entity bean fields defined in deployment descriptors to database fields**
- **provide support for complex object to relational mapping, such as a one-to-many relationship.**

These tools, however, can be used to edit entity bean 'finder' method expressions. Yet 'finder' methods are defined in the home interface for an EJB. Again, each application server vendor has a proprietary strategy for defining how the 'finder' methods are implemented.

With container-managed persistence, each finder method defined in the home interface must be explained to the container managing the entity bean. This explanation is provided to the container during deployment of the entity bean using vendor-specific deployment tools and syntax.

Being proprietary by nature, these tools bind the definition of an entity bean's persistence to the application server

software provided by that particular vendor — resulting in the development of entity beans that have dependencies on the tools that have been provided by the application server vendor. As a result, entity beans written to run on a specific application server cannot be moved to other application servers developed by different vendors without encountering deployment errors.

Packaging into JAR files

In addition, each EJB is packaged into a JAR file for deployment onto an application server. This JAR file includes the class files and the deployment descriptor for the bean. The deployment descriptor:

- **defines the deployment properties for an EJB**
- **directs the deployment tool (on the chosen application server) as to how to deploy the EJB within the container.**

The Enterprise Java Beans specification version 1.1 requires that the deployment descriptor be saved as an XML file named ejb-jar.xml in the META-INF directory. Each application server includes a container that manages the interactions between the bean and the server. Deployment tools use the ejb-jar.xml file to add a bean to the container.

Most application server vendors require one or more proprietary deployment descriptor files for deployment of beans. While ejb-jar.xml defines common EJB properties, the vendor specific deployment descriptors define custom properties for deploying a bean within a specific application server container. The container then uses both the required and the proprietary deployment descriptors to apply primary services to the EJB at runtime.

Similarly, a deployment JAR file for an EJB includes both the vendor's proprietary deployment descriptors and the required deployment descriptor. The proprietary deployment descriptors are specific to the application server vendor and are therefore incompatible with an application server developed by another vendor. Once again, deployment errors will be encountered when attempting to move an EJB across J2EE compliant application servers.

Possible resolutions

From all the above it should be more than obvious that 'all EJBs do not necessarily talk to all other EJBs' — unless they were built on and run on the same application server, which was not the case at the financial institution. The question is: "Can such difficulties be solved?"

An EJB can require manual intervention — the rewriting of code in order to accommodate the service implementation of another application server vendor prior to moving the EJB. To promote the development of EJBs that can be more easily modified to accommodate another vendor's implementation of low-level services, a developer should identify service implementations that are specific to a particular vendor. These services might then be isolated into service factories in order to create a layer of abstraction between:

- **the application framework**
- **the services provided by the application server vendor.**

Similarly, an EJB that is dependent on a specific vendor's implementation of persistence for container-managed entity beans may require modification prior to being deployed on an application server developed by a different vendor. Although entity beans can be modified using the persistence tools provided by the new application server vendor, this often results in EJBs that are still dependent on a specific vendor's implementation of persistence. To overcome this, a developer may consider using bean-managed persistence rather than container-managed persistence to handle entity bean persistence.

Finally, an EJB that includes deployment descriptors that are proprietary to a specific vendor's application server cannot be deployed on another vendor's application server without modifications. One solution here is to develop deployment descriptors that are required for the new application server. But this solution also requires that a new deployment JAR file be created for the EJB that includes the new deployment descriptors.

Another approach

However, these are not the only approaches. For example, there are products which complement application server tools and processes in order to automate the conversion of deployment descriptors. [One such tool is the Cacheon one (Figure 4.1), which was used at the financial institution.]

This tool is able to read proprietary deployment descriptor files for several vendors and convert the contents of the files to the deployment descriptor representation appropriate to the new application server. Figure 4.2 shows the deployment of a JAR file — BankAccounts.jar — for a session bean.

In this, the BankAccounts.jar file includes:

- **the META-INF directory**

Figure 4.1:
The Cacheon Business Services Center (BSC)

The Cacheon BSC is built upon a dynamic component architecture which acts as a collaborative environment for discovering, assembling, deploying and maintaining business services at runtime. The main tools that comprise the Cacheon BSC include the following:

- **the Business Services Engine (the 'Engine')**
- **the Enterprise Warehouse (the 'Warehouse')**
- **the Enterprise Warehouse Console (the 'Warehouse Console')**
- **the Business Services Console (the 'Console').**

The BSC Engine, an extension of the Java Virtual Machine, supports a local warehouse and the loading of business services into that local warehouse. Additionally, it supports the loading of modules that facilitate communication across Engines using 'Voyager'. Voyager provides Object Request Broker functionality and facilitates communication between warehouses and multiple application servers (including JBOSS, IBM's WebSphere, BEA's WebLogic, Oracle's 9iAS, with others being added).

A module is a dynamically loaded class that augments the capabilities of an Engine on a particular machine. This enables an Engine either to acquire essential capabilities or to discard unneeded ones, as requirements evolve.

The result is a modular environment that provides a secure way of augmenting the run time functionality of the Engine. The modular nature of the Engine enables functionality to be added, including support for a range of networking technologies, protocols and standards.

Users of the Cacheon BSC can publish business services to the Warehouse. Business services that have been updated in the Warehouse can be propagated to users, using 'pull' methods. With the pull method, a user can refresh a business service, or a set of business services, to the latest version. When retrieving a business service, a user selects the version of the business service he or she wants to retrieve.

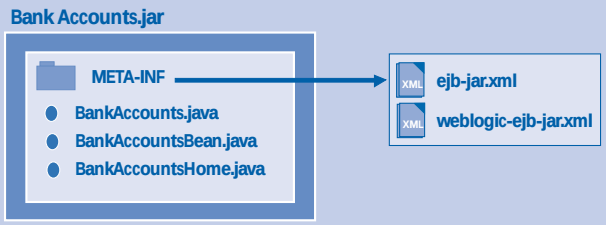
The real attraction of the Cacheon BSC, in a J2EE/EJB environment, is that EJBs that were built to run on different applications servers can be linked together — which is not always possible. As the attraction of EJBs is supposed to be that they are building blocks, the BSC adds an otherwise missing piece to the J2EE equation.

- **the Java files that comprise the EJB.**

The META-INF directory includes:

- **ejb-jar.xml, which is required for EJB deployment**
- **the proprietary deployment descriptor required by — in this case — the WebLogic application server (weblogic-ejb-jar.xml).**

Figure 4.2: Deployment of a JAR file

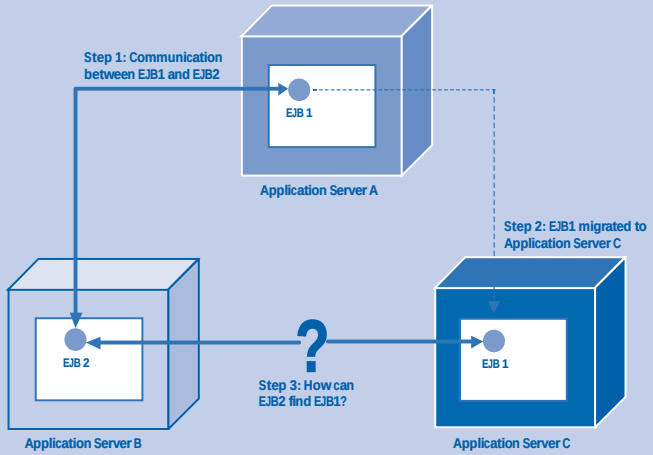


Assume that the session bean is currently deployed on a JBoss application server. This session bean could be moved from JBoss to WebLogic, using software to automate the conversion of deployment descriptor information from jboss.xml to the format required by weblogic-ejb-jar.xml. Furthermore, this session bean could later be moved from WebLogic to Oracle 9iAS, using the conversion software to automate the conversion of deployment descriptor information to the representation required by orion-ejb-jar.xml.

For all this to work, a developer must know about the issues which must be addressed if vendor specific implementations of context factories are used to facilitate communication between EJBs across application servers. Ideally, any such solution must include context factory implementations for multiple application servers.

In order to facilitate communication between EJBs on disparate application servers, context factories for one application server can be deployed onto another application server. This enables EJBs to communicate across application servers while eliminating problems that arise from vendor specific implementations of context factories.

Figure 4.3: Communicating EJBs



More complications, and solutions

However, a more complicated issue will arise when an EJB has already communicated with another EJB that has then been moved to an application server developed by a different vendor. This scenario is depicted in Figure 4.3.

Step 1 in Figure 4.3 illustrates the cross application server communication occurring between:

- **EJB1, an EJB deployed on Application Server A**
- **EJB2, an EJB deployed on Application Server B.**

If EJB1 is moved from Application Server A to Application Server C, as indicated by Step 2, the communication between EJB1 and EJB2 will no longer work. This is because the context factory for locating EJB1 is implemented for Application Server A and not Application Server C, where EJB1 has been moved.

In this scenario, a developer would need to utilize a solution which deploys the context factory implementation for Application Server C on Application Server B. This would enable Application Server B to locate EJB1.

Additionally, a developer could deploy the context factory implementation for Application Server B on Application Server C, thus enabling Application Server C to locate EJB2. Furthermore, using some form of automation software, both EJB1 and EJB2 can be recalled as services and modified in order to facilitate two way communication across application servers.

Management conclusion

To compete effectively, enterprises must respond quickly to changing customer, competitive and market demands. Further, they must be adept at functioning within a diverse and complex computing environment. This requires that existing enterprise resources be leveraged regardless of how these resources are developed and where these resources are deployed.

Unfortunately, despite the hopes for application servers, EJBs and J2EE, incompatibilities do exist between different application server and J2EE implementations. While not major, these issues are not simple to resolve — being located as they are in the implementation of low level services. Nevertheless, the result is a headache for enterprises trying to make maximum use of J2EE and EJBs.

Although there is a push to develop standards between the different application server vendors, the desire for competitive advantage inevitably takes precedence over the acceptance and introduction of standards. This, coupled with the fact that the Java standard is itself still evolving, makes the development of standards between application server vendors even more unlikely.

That is, however, not the whole story. As Dan Rolnick has described, alternate solutions (like the one from Cacheon) do exist and work. They do this by providing 'reference points' for EJBs. With these, different implementations of EJBs are able to work together — which was the original objective after all.

Application brokers or application servers: a 21st Century dilemma

Charles Brett
President, C3B Consulting Ltd. and President, International Advisory Board, MIDDLEWARESPECTRA

Management introduction

In today's software market place there are two broad categories of product which go under similar names but are, in practical implementation terms, very different. Both can deliver quite similar results, albeit via different paths. These categories are:

- ***application servers (iPlanet, WebSphere, WebLogic, 9iAS, etc.)***
- ***application brokers (MQSeries Integrator, e-Biz Integrator, e*Gate, etc.)***

Customers are constantly being invited to choose between these. There is, however, a lack of comprehension about how these are, at the same time, so different and so similar.

Application servers

In general terms an application server is a run time engine on which applications can be run. In this context, application servers can vary from the simple to the sophisticated. Arguably, an operating system, or even a database on an operating system, is an application server — you can run applications on such a server ... Yet application servers, in their more complex guises, are designed to be scalable, secure and transaction-capable (Figure 5.1 shows a list of 'typical' functions).

In today's market terms, an application server tends to be a specific run time engine which has a variety of technologies associated with it. For example, IBM, BEA and Sun have all built their application servers around Java. Java considerations (constraints) permeate their whole approach to application servers, although not necessarily to the exclusion of previous technologies (as both WebSphere and WebLogic demonstrate).

Unfortunately, the name 'application server' is not just associated with running applications. It is also heavily associated with application development in general (and Java in particular). One has only to look at the Oracle 9iAS Web site or IBM's WebSphere or Sybase's Enterprise Application Server product families to see that each of these comprises and gathers together multiple different varieties of run time engines plus multiple varieties of development tool (in IBM's case as varied as Visual Age for Java, C++, C, ... through a host of other tools).

There is, therefore, a common confusion about what an application server is. This applies as much to the vendor community as to customers. Neither seem quite sure whether an application server is about:

- **application development (where most people-time is spent)**
- **or application serving (when the ultimate value is delivered).**

Application brokers

Now think about application brokers (which are sometimes known as integration brokers or even message brokers). These are also software products and are available from a myriad of vendors, including:

- **IBM, with MQSeries Integrator (and even MQSeries Workflow)**
- **Sybase, with its e-Biz Integrator/Process Server family**
- **SeeBeyond, with its e*Gate collection**
- **webMethods, with its Integration Platform**
- **TIBCO, with its Integration Server**
- **Sun, with the iPlanet Integration Server**
- **some 30+ other organizations.**

Application brokers sit between existing applications and enable these applications to work together (Figure 5.2). An application broker 'brokers' the linkages between the

Figure 5.1: Application Server — typical components

J2EE support
Java Servlets
Enterprise JavaBeans (EJB)
Java Server Pages (JSP)
Java Database Connectivity (JDBC)
Java Naming and Directory (JNDI)
Java Transactions (JTA)
Java Messaging (JMS)
Java Security
RMI
HTTP
SQLJ support

Web Server to JSP/Servlet to EJB connectivity
HTTP and HTTPS tunneling
Load balancing
Availability
Connection re-routing
Application failover
Clustering
Session state replication (and failover)
IP multicast
Logging
+ copious and varied development tools

source and destination applications, thereby reducing the potential number of interfaces from ***n(n-1)*** — where ***n*** is the number of interfaces — to ***n*2***, a significantly more manageable number.

As such, application brokers are a mechanism for connecting applications — which explains their utility in EAI (Enterprise Application Integration). They reduce 'application connection spaghetti'. Furthermore, they can be extended to include process flow automation and work flow management, which introduces the capability to manage state and long running transactions. They are, inherently, asynchronous — in the sense that different application elements can be loosely coupled together.

Application brokers are, essentially, run time products. The focus is on the message and process flow, with little human interaction (until you reach full blooded work flow and its inclusion of people interactions).

But that is not to suggest that development is unnecessary: it is, except that development usually takes place externally to the application broker itself. In MQSeries Integrator, for example, nodes are built in C++ or Java. Working nodes are then assembled into flows on a palette which is then deployed onto the application broker run time engine.

A similar approach is delivered by e*Gate and e-Biz Integrator. In the latter, the rules and formats are created externally, and loaded into a database for execution.

Application brokers and servers are the same?

Ah ha, you may say. Does that not mean that application brokers and application servers accomplish the same only differently? The answer is 'Yes'. Both have run times engines. Both have development 'environments'.

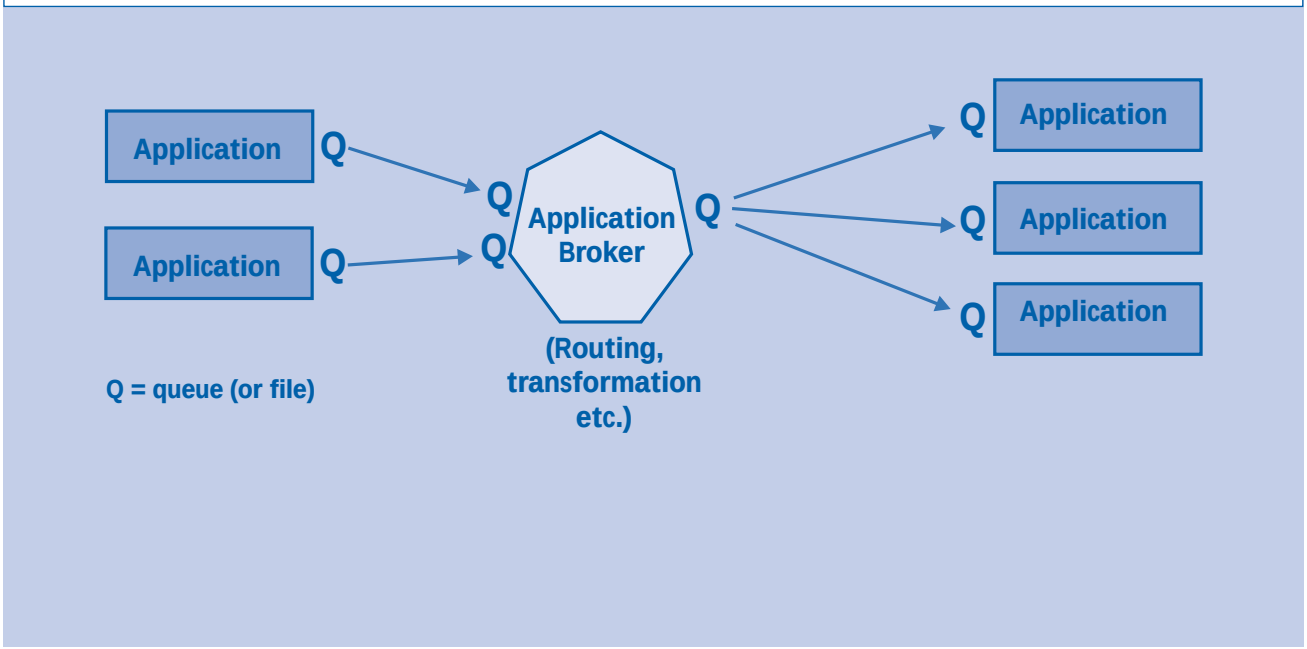
Indeed, the similarities go even further. You can use an application server to build what looks like an application broker. You can use an application broker, especially if process handling is added, to deliver what is to all intents and purposes an application server (a system running applications). Both can be used to provide application integration — which is what both are being sold to do.

Do the differences matter?

This raises the question of whether the differences — which are considerable in terms of approach, technical architecture and style — matter. We argue that the differences do matter. They probably matter most if you come from:

- **an application development background (when an application server will almost always look more understandable)**
- **a business process background, when the concept of an application broker will generally appeal more (working as it does between existing entities, applications).**

Figure 5.2: Application Broker — a typical configuration



If you come from neither background, the only prediction we will confidently make is that you will be confused — as you attempt to acquire both development and business process knowledge before sorting through the many possibilities.

Too many, too much

In one sense, therefore, IBM's initiative to place both its application broker and application server families under the one WebSphere name makes sense — if you know from where you are starting. If, on the other hand, you are new to the potential being offered, trying to sort out what you want and why will be a true challenge.

But perhaps the more meaningful, in architectural terms, is the approach taken by Sun. It has placed its application server (the run time) beneath its application broker. The application server is the engine on which the application broker function works.

While the Sun implementation is relatively unproven, it points to a simplification — of description, implementation and deployment — that we think is both desirable and clear:

- **the application server should be exactly that, a run time engine for applications (which could include application broking, process flow, work flow, etc.)**
- **the application broking (or other functions like process or work flow or whatever) is a specific implementation of application function which uses the underlying application server**
- **development tools, applicable to each functions, are kept separate**
- **transports, the mechanisms which bring payloads for processing, are also separate.**

This would be a much more attractive story for customers. It would be a more logical story for vendors. It would certainly simplify decision making. But it is not all. Now add PFA.

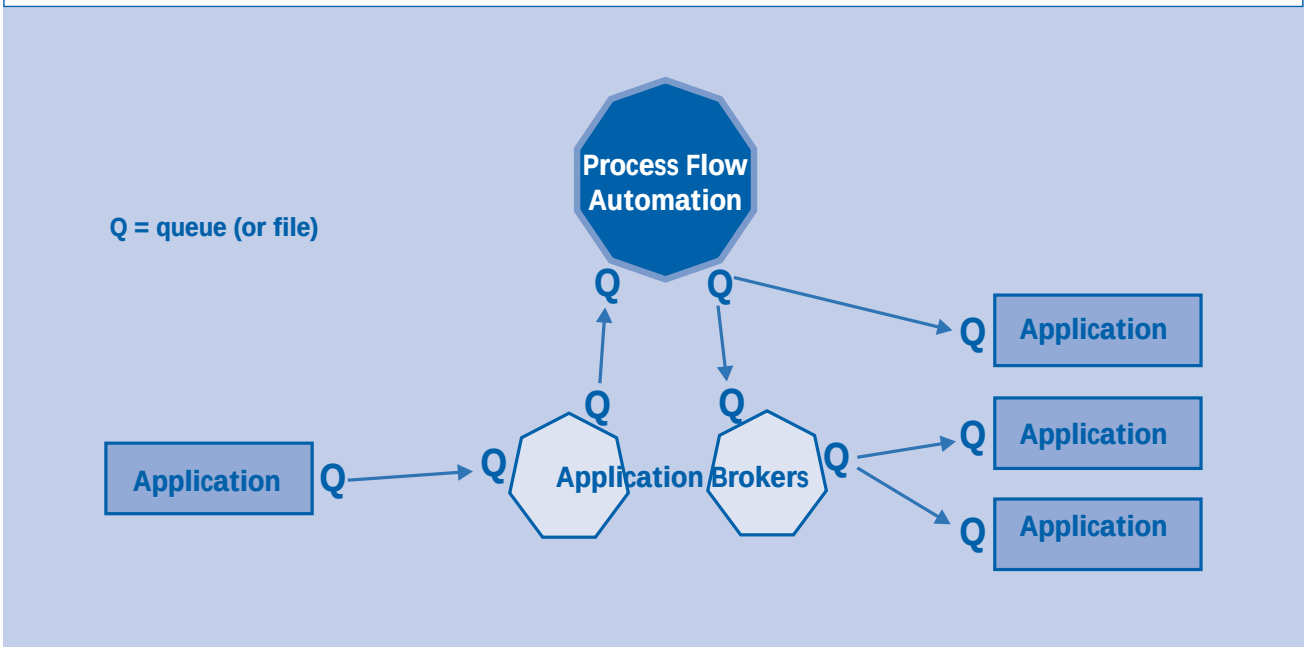
Process Flow Automation

As described above, both application servers and application brokers are relevant for integration, if in different ways. Traditionally, application brokers were associated with linking business applications inside an enterprise (intra-business application integration), which was often true for application servers as well.

With the arrival of the Internet, the need for linking selected business applications across the Internet (inter-business application integration or B2B) accelerated.

However, neither application servers nor application brokers are a total answer. They automate specific processes in

Figure 5.3: Process Flow Automation with an Application Broker



a relatively fixed manner. Once working, they run as needed.

In contrast, process flow automation (PFA) enables businesses to add an additional level of sophistication by which to direct whole business flows. To comprehend what PFA can do, think about two typical business processes:

- **a mortgage company making a house loan**
- **a business making a sale to another business over the Internet.**

In the first, the mortgage house will possess a formally defined process about how mortgage applications should be handled, from the customer completing the form through authorization through actual delivery of money to either the customer's account or the seller of the house. In practice, because of the way that systems have evolved, as islands of automation, it is likely that several different, and specialized, applications running on different existing physical systems will be used:

- **from the mortgage origination entry application**
- **through credit checking (which might involve external references to third party credit databases)**
- **through a payments system**
- **through to the mortgage house's own ledger system.**

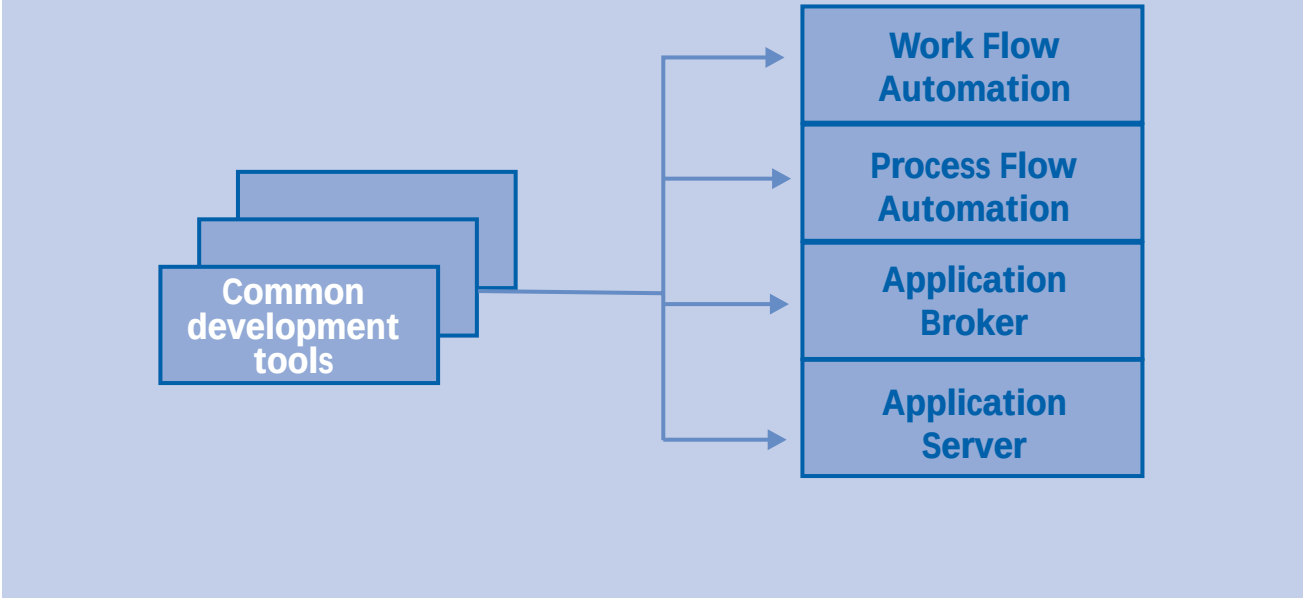
Using the traditional approach, even for the simplest mortgage, there is substantial human interaction — primarily supervising the application through the various sub-processes. Automating as much of this as possible has the potential to be faster and cheaper for the mortgage house (and customer), as well as require fewer people.

In the e-commerce instance, a business might have a Web site with an order form. A business customer decides what to buy (say) 1000 widgets. That order has now to be processed, and the widgets delivered. But to do this a sequence of processes — the order and delivery business process — has to be completed, from:

- **establishing whether the widgets are in stock (checking against inventory)**
- **making a credit check (possibly against external credit agencies)**
- **arranging for delivery (and notifying the customer)**
- **calculating discounts**
- **invoicing, once delivery is made**
- **posting the sales and general ledgers (which may be held on different systems)**
- **calculating the sales commissions to be paid.**

With the exception of the physical delivery, most of these processes could be automated. People involvement is not needed — if the various applications can work together.

Figure 5.4: PFA+Application Broker+Application Server



What PFA enables is the automation of a predictable sequence of events, but where different choices of actions are enabled — which can link two or more applications. This is instead of those events being triggered by one or more people.

In business terms, PFA is not the same as work flow automation (WFA). WFA originated with large paper-based organizations — like insurance companies and government — which looked for software to 'manage' the paper flows within their organizations. This frequently included long, complex processes — for example, the completion of an insurance claim, from submission right through to final payment or rejection.

In the terms of the mortgage application or the sales of 1000 widgets examples above, PFA can manage the process to process steps. Even more, it can manage (if sufficient attention is paid when it is set up) the automated catching and correction of errors or discrepancies. By reducing the involvement of people (where they are not really wanted or needed) processing can be further automated, and more accurately.

While PFA offers more than an application broker, it does not aspire to deliver the complexity (the human involvement) of WFA. The attraction of PFA is that it is:

- **simpler**
- **restricted to processes that can be automated without human involvement.**

PFA is, therefore, WFA without people and applied to processes or sub-processes which are already described in software. This is significantly easier to deliver than WFA. PFA bridges between processes that already exist in applications that are already running in organizations (or between organizations) — by applying a layer of logic which can direct and feed the applications with which it (PFA) has connections (Figure 5.3).

Layering a common sense solution

In some ways it (PFA) sits atop application servers and brokers and below the people-rich involvement of WFA. It does this by automating the co-ordination of activities between processes that are already 'captured' in existing or new applications and brokers without surfacing interactions to people (and all their associated complications).

In practical terms, think about the newly expanded WebSphere family or Sybase's portfolio — of Enterprise Application Server, e-Biz Integrator, Process Server and Portal Server. Envision a world where application brokers and PFA function are built using application development tools which produce function that can run on application servers. Go one stage further and appreciate the increased attraction of using those same development tools, which can be used for defining process and work flows. This would be coherent.

Most vendors' product stories would be more attractive if their application servers borrowed from the production, deployment and resiliency capabilities that application brokers possess. Similarly, most application brokers would benefit from being associated with modern run times, like those found in application servers and with commonly used development tools. And PFA would make even greater sense sitting above them both (Figure 5.4).

Management conclusion

This may be to ask too much. But, if vendors are bold enough, they might remove the fruitless 'angels on the head of a pin' debates about whether to buy an application server or application broker solution — when both accomplish the same broad objectives, only differently.

This would be a step forwards. Middleware is already complicated enough without having to wade through largely meaningless differences — that are mostly in the eye of the developer (vendor) rather than the end user.

The state of business rules

Martin West
Vice President of Research and Development
SpiritSoft

Management introduction

In this analysis, Martin West — the Vice President of Research and Development at SpiritSoft — reviews the evolution of business rules in the context of event driven systems. He introduces and positions reactive rules in a number of business areas, as expressed through the 'Event-Condition-Action' (ECA) paradigm.

In addition, Mr. West describes:

- ***the need for rules, by segmenting the problem domain into deductive rule processing and reactive rule processing***
- ***the concepts that underpin 'ECA' rules — and how these are instrumental in solving many of the business critical problems that arise in today's highly connected environment.***

Background

Up until the late 1980s, applications were generally procedural in nature. Business processes had to conform to the way the application that supported these processes was written.

A number of developments occurred towards the end of the 1980s that began to change the way applications could be written and operated:

- **Apple, IBM and Microsoft led the development of 'windows-based' systems; applications could now be written to include event driven user interfaces**
- **with the introduction of IBM's MQSeries, the development of message orientated middleware (MOM) came of age (it had existed before, but not really in self-standing product form)**
- **the introduction of triggering functions within database systems enabled applications to be called when data changed or was added to the database**
- **object orientated programming began to emerge from its evangelistic stage.**

However, none of these fundamentally altered the way that applications were written. The old, familiar procedural techniques were, and are, still fully in operation. Business rules were, and are, still deeply encoded in traditional applications.

As such, business rules were not visible to business people. To all intents and purposes, this meant that only programmers could change applications. But such changes were often costly as well as being prone to error.

Realization

By the 1980s there were those in the IT industry who recognized that these recurring problems were unacceptable — and that alternative solutions were required. One example of an effort to break free from traditional bounds was IBM's Insurance Application Architecture (IAA). This defined the concepts of business rules and business rule processing.

Since then, along with other developments, business rule technology has been following a typical 'technology adoption curve'. Today, IT professionals:

- **are seeing the emergence of business rules technology from the evangelical state**
- **can reasonably expect it to become 'matter of fact' in the next five years, in much the same way that object technology has been accepted over the past five years.**

Standards are, as so often, playing a critical role in its wider acceptance. JMS (Java Messaging System) and JAXM, in particular, offer definitions for the delivery of synchronous and asynchronous events. Equally, business related XML standards — such as ebXML and finXML — provide the semantics and choreography of industry specific business events and scenarios. Finally, Business Rules Standards are being defined by:

- **RuleML, which is part of the W3C Semantic Web Standards activities**
- **the Java Rule Engine activity, sponsored by Sun.**

In other words, rules have moved beyond the conceptualization stage. They are moving towards production.

Business rules

A business rule is a statement that defines or constrains some aspect of a business. It asserts structure or control which influences the behavior of the business. Traditionally, the dominant force in rules technology has involved 'deductive solutions'. These are rules that are inferred without the need to be told how to do this.

But this is not the only type of rule. As this analysis will describe, there is an alternative type of rule solution — known as a 'reactive' rule.

Today, many products support business rules. More often than not, however, products are specific to an industry.

The reason for this is that it has proven difficult to build a 'one size fits all' business rules system, not least because the terminology (and audiences) vary so differently between industries. For instance, an application that enables actuaries to define business rules for life and pension products has surprisingly little in common with that appropriate for the administration of a property and casualty system — and that is within the insurance industry. Between industry types, say, insurance and manufacturing or banking and pharmaceuticals, the differences are even greater.

Why do you need rules?

Business rules are, already, pervasive. They cannot be

avoided. Whether it is calculating the value of your life insurance premium or the cost of a mobile telephone call, it is a business rule which does the work. In this sense, business rules are the synthesis of one or more processes ('find out the number of minutes used, find out the cost per minute, multiply the two together and obtain an amount to be invoiced ...').

In the past, applications too often became unwieldy and inflexible as they grew or expanded. In so doing they began to fail to fulfill business needs. In addition, such concentration of logic and process created maintenance nightmares. These became ever worse as applications were modified in an attempt to address new business requirements.

The consequence was that organizations realized that they could not avoid the fundamental problem, which was that business requirements change faster than applications can either be created and/or modified. The attraction of rules is that they offer a way to:

- **encapsulate business semantics**
- **promote them to the surface in the same way that databases enable us to separate data from applications.**

We need rules so that we can withdraw the business semantics that currently are embedded in applications. In so doing we can provide improved support for the decision making processes that underpin the ways in which organizations operate. This is an essential requirement in today's operating environment where businesses need to remain flexible, whilst at the same time driving their IT infrastructure forward towards supporting complex real time decision making.

Having said all of that, business rules have not yet taken off — principally because of the lack of agreement on the standards and the technology that exists to implement the systems. However, this is changing as messaging, business process and business rule standards win ever greater acceptance and approval.

Different rules for different problems

Because rules enable us to capture business semantics as well as to support on-the-fly decision making, they clearly possess the potential to become part of IT's total offering. But not all rules, however, are created equal.

It is here I depart slightly from Barbara von Halle's view that all rules are about data, which she expresses in an excellent series of articles on Business Rule Systems. This was published recently in the DM Review (www.dmreview.com) . In my view, you should separate:

- **rules about data**
- **from rules about business events.**

Data-centric rules can be applied to large data sources to infer patterns and to answer questions. We might apply rules to an historical database to look for the reasons why something happened.

For instance, these might infer a relationship between two or more claims, and associate their risk, which then has an effect on the premium calculation for an insurance policy. These sorts of analytical problems are best solved using Rete-based deductive systems and data mining applications.

In contrast, business events form an essential part — the life blood — of any enterprise. Events codify the way in which a system responds, whether it is in procurement, insurance, finance, supply chain management or any one of the many forms of electronic data interchange that exist today.

For example, when an insurance product is sold, an insurance product administration system might generate a 'product sold event' which would be sent to the sales compensation system. The compensation system would process this event — according to its set of business rules; these rules might even generate additional events — that would eventually reach the payroll system (Figure 6.1).

Why business rules are different from rules about data

It is not sufficient to be able to expose the business rules and enable their rapid modification. A business rule system needs to support change management and version control. In this sense it differs from rules about data. Let me illustrate why.

Over time different versions of a rule will exist. A rule may have to be changed because the business itself has changed, or because the rule does not correctly model the business (a bug fix).

A further complication is that a rule may be part of a contractual agreement, say an insurance policy for instance. This may have an operational life of tens of years. Take the example of a sale person paid by the amount and longevity of a policy. He or she sells a policy, and the reward is a one-

off payment based on the assumption that the policy-holder continues to pay for ten years or more.

But what happens if the policy is canceled after six years? The sales person will have been overpaid. His or her compensation needs to be adjusted in terms of the compensation scheme in operation six years before. [Such long intervals are not that uncommon: just think of products with three or five year guarantees.]

So, in addition to the development version control and change management of business rules, there is an operational dimension as well. A business event will have a business date associated with it. When a rule is executed the business rule system must locate the appropriate version of the rule for that business date. Business rule systems today are immature in this respect.

ECA and rules

An event-centric rule is best expressed as a forward chaining set of Events, Conditions and Actions:

- **an event can be a message from a messaging service (say, JMS) (for example about a product sold or a 'request for quotation' message), a property change (a button press to enter a communication received) or a state transition (an insurance policy is about expire)**
- **conditions are predicates that are used to decide what actions should be executed; they are applied to the event (or events) that precede it and return true or false answers ('the beneficiary's age is within acceptable limits')**
- **actions represent what should be done; typically actions are built-in functions (sending a sales compensation update message or making a request for external communication to use a print service) or any method accessible to the rule.**

The benefit of a rule-based approach is that it develops and codifies the semantics of all business transactions that describe 'how things are done'. This reduces the cost of integration, training and operating a business over time.

At their simplest, ECA rules decouple application interfaces from business logic. They enable each to be optimized and, most critically, changed independently.

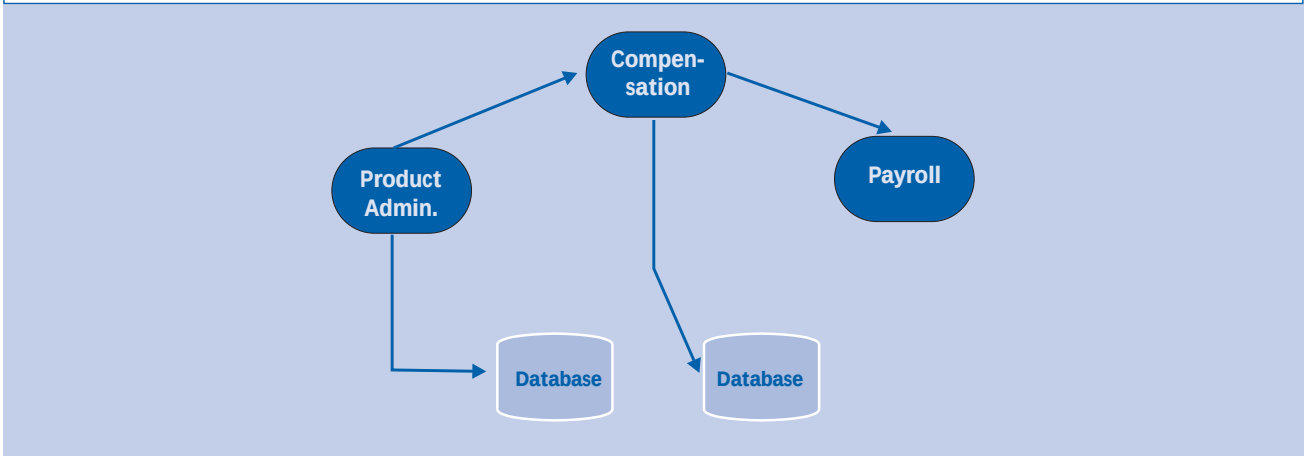
The ECA framework (multiple ECAs)

By looking at a simple conceptual model for describing re-active rules — in the form of Event, Condition and Action — it is possible to introduce ECA rules and an open ECA Framework. The advantage of this is that it forms the basis for collaboration and co-operation between:

- **different ECA rule implementations**
- **re-active rule and deductive rule implementations.**

In essence the framework enables the definition of business events — and the conditions and related actions that should be performed — to be expressed in an interchangeable format based on an XML schema. In the case of a sales compensation system:

Figure 6.1: Compensation and Payroll rules



- **the sale of a product would be a business event**
- **it would produce a result which sent another (new) business event to the payroll system.**

A Java-centric ECA Framework implementation is a set of interacting components that conform to the ECA Framework interface definitions, allowing multiple user interfaces to be constructed that fit the needs of the user community. The framework supports forward chaining ECA rules and, by so doing, produces implementations which are able to take a generative approach that, in a Java implementation, enables the rules to be generated (on-the-fly) into Java code. A C# based solution could conceivably do the same while still conforming to the ECA Framework interface definitions.

Components, embedded or stand-alone, are able to subscribe to rule changes delivered from a JMS message bus, and load them dynamically. For example, in the SpiritIntellect implementation with which I have been involved, a unique version-based class loader is used — without the need to stop the running rule.

In this way the framework is unique. It enables Java Virtual Machines to be used as rule engines, by exploiting a JMS compliant messaging bus for rule distribution as well as event delivery.

In this implementation, the ECA rules are geared to the problems inherent in event-based systems, the footprint is small and, because the rules are implemented as Java code, the rules can be embedded seamlessly into Java applications. This means that rules can be run on devices as small as PDAs — at least those that support Java Platform 2 Micro Edition (J2ME) with a Connected Device Configuration (CDC) — as well as on high end servers.

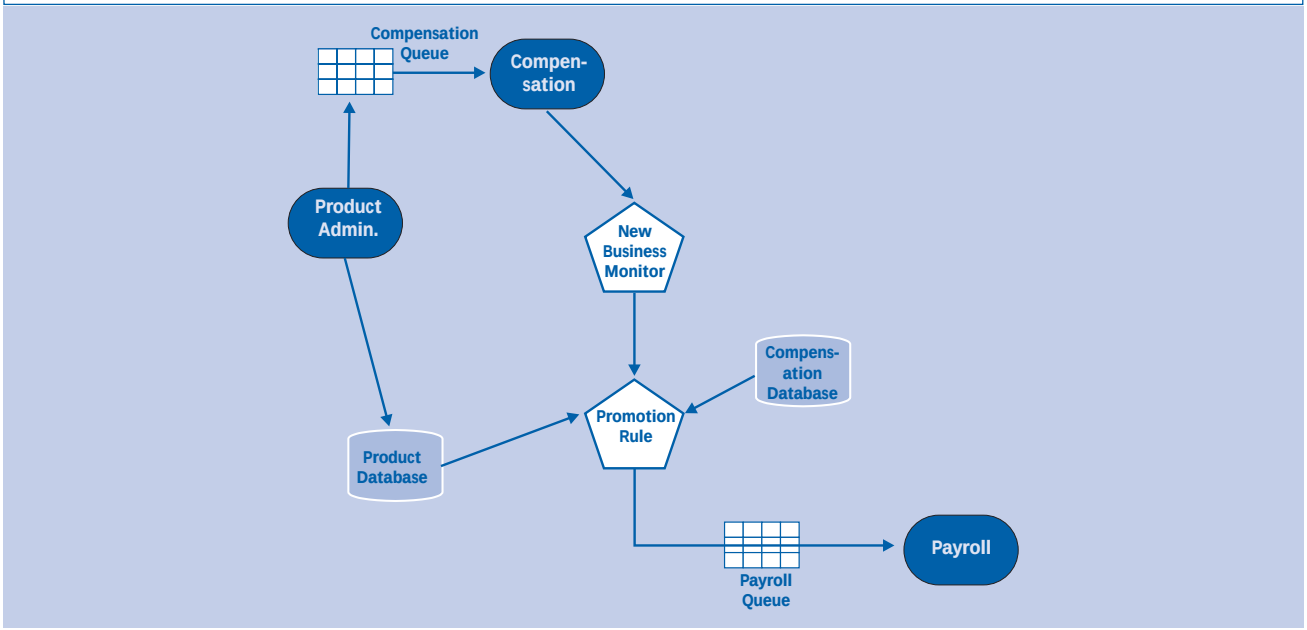
Rules can, therefore, be embedded into anything from a simple Java bean to an EJB session or entity bean. This makes ECA rules deployable in a wide range of environments and applications. It provides event-based flexibility wherever it is needed.

In practice

The sales person compensation system I referred to earlier could easily be implemented using an ECA framework. There would be a monitor to process incoming events that execute an appropriate business rule depending on the kind of the incoming event (Figure 6.2).

For example, the business could react to a new competitive product by introducing a sales promotion which offers the sales force an extra incentive to sell a particular product. To deliver this (without traditional coding), you would create a 'new business rule' which would in turn invoke a promo-

Figure 6.2: Compensation with ECA rules



tion rule whenever the product sold was the one being promoted.

This is an example of a short lifecycle rule. It could be implemented and deployed in a matter of days. It might exist only for a few weeks or months. Equally important, it could be discontinued — by removing the rule, with equivalent ease and speed.

At the opposite end of the spectrum, a life insurance premium payment event might trigger one or more rules that allocate portions of the premium to investment funds. The deployment and validation of this rule would be longer and more formal — because of the contractual nature of the business event.

In this instance, the rules might need to be operational for several years. Indeed, the rules might be upgraded over the lifetime of the investment agreement because of corrections, new requirements or regulatory changes. Whether subsequent payment events operate with the new version depends on the nature of the changes to the rule. One could even envision circumstances where meta-rules dictate which rules are applied to a particular business event and determine which rules should be used to police other rules.

Management conclusion

The technologies and standards for the successful development of business rules are just beginning to mature. The infrastructure to provide asynchronous messaging is well established and the JMS standard will increase the general acceptance of message oriented middleware.

XML standards — like ebXML and finXML — will enable message payloads to carry information against which ECA rules can be easily applied. Business rule standards will enable the development of componentized business rules, ones which can be assembled, shared and adapted by the wider community. Event driven execution, management and monitoring of business rules can be achieved in a truly flexible way to ensure that systems remain adaptive to business needs at all times.

As Mr. West has illustrated, ECA rules are a technology which enables the management and execution of business events. By providing an environment for the specification of complex rules which allow multiple events to participate in a business transaction, ECA rules can be constructed and deployed anywhere in a network.

Mr. West has also shown that:

- ***flexibility of deployment is an important requirement with the ability to run rules as stand-alone rule agents or as embedded rules within applications***
- ***an open ECA Framework exists — as a conceptual model that has the potential to allow interoperability of rule based solutions that includes both event and data driven rule technology***
- ***this obviates the need for a separate rule engine — thereby reducing the footprint of rules and ensuring an easier integration between the rules and their surrounding environment; it enables the choreography of existing components to be easily achieved.***

A reality check for .NET and J2EE Web Services

Mark Creamer
Consultant

Management introduction

More than a year ago, Microsoft announced its new vision for the future, the so-called .NET initiative. Broadly speaking, it was about three principle technology areas:

- ***first, there is common language runtime and C# ('C sharp')***
- ***second, there are a set of enterprise servers which are XML enabled***
- ***third, there are 'Web Services'.***

Each of the three principle areas has value independently of the others. Taken together, the technologies comprise a platform for distributed computing across enterprise boundaries.

Yet, of these, it is the Web Services area which has become the lead story — at least from an IT point of view. But it is not only Microsoft that has a story for Web Services; so do many other software vendors, from as diverse a collection as Sun, BEA, HP, Oracle, Sybase and many others — as Mark Creamer explores.

Web Services

In essence, Web Services are an old story with new provisions. Take established distributed component models (DCOM and CORBA), add support for Web protocols on an interface to a function and the browser is no longer the only potential client for HTTP-based services.

Web Services are significant because of the integration of the range of application components. These are intended to be implemented on different platforms using different programming models. Historically, however, such grand intentions required binding with proprietary middleware APIs, for example for messaging. Unfortunately, there has not been an industry accepted standard for wire-level messaging.

Now there is. Furthermore, that standard is within both .NET and the Java platform. It is the Simple Object Access Protocol (SOAP). It is SOAP that enables Web Services.

SOAP, and more

SOAP runs on top of HTTP, among other transports. This is crucial because HTTP is ubiquitous and avoids the need to punch undesirable holes in firewalls. A group which included IBM and Microsoft, two of the most significant forces in the software market (with products shipping that support Web Services today) authored SOAP.

In addition to SOAP, two other standards are fundamental to Web Services:

- **the Web Services Definition Language (WSDL)**
- **Universal Description, Discovery and Integration (UDDI).**

As with SOAP, these are XML-based. Unlike SOAP, they are not required to have a Web Service (although this is highly desirable).

WSDL is used to describe service capabilities and invocation procedures. It is roughly analogous to some early, and proprietary, EAI middleware connector functions.

UDDI is a registry specification. It is capable of storing WSDL, among other things.

Development

These are not the only standards relevant to the Web Services technology area. The others, however, are barely out of their discussion groups (so far). More important than these, by far, is the development tool set.

Development tools deserve attention immediately. Over past years, middleware — and platform suites in general — have focused mainly on the run time environment. Vendor answers to development questions were along the lines of:

- **'we support your favorite IDE'**
- **or 'we have an SDK you can use to create your own development tool set'.**

This was manageable, if barely. But it was certainly not desirable, or helpful, if you were planning a distributed systems development project or infrastructure.

In a Web Services scenario, where developers are in different organizations and potentially using hundreds of components, there is a real risk that scale itself becomes a problem. There must be support in development workbenches for:

- **data model aggregation**
- **automatic generation of interfaces**
- **discovery of services.**

In addition, there needs to be coherent (and consistent) support for team-based application component assembly. And there must be:

- **project management support**
- **independent budget monitoring**
- **accounting**
- **work flow management (for the whole engineering process).**

The back plane of the Web Services vision is the same as for distributed components in general — mass customization and rapid assembly of applications, albeit with a cross-firewall spin added. Given that, today, B2B trading, e-business and customer-facing applications represent the majority of new application initiatives, the battle among the major software vendors to win your business — as the Web Services platform of choice — has already begun. These three areas, by definition, are cross enterprise applications.

Needless to say, we are some years away from seeing all these concepts made available in development environments specifically for the purpose of supporting the peculiarities of Web Services projects. In this context, therefore, the Web Services vendors have yet to make a serious improvement on what their predecessors had failed to do.

.NET

The core product for .NET platform is Visual Studio.NET.

The focus, for Microsoft, is to support development of new, Internet-based, subscription services as a part of creative projects. One need only look as far as plans for Windows XP to observe that Microsoft sees applications becoming much more 'lively'.

While the Microsoft solution is intended for Windows only, at least as far as serving up the resulting services goes, many programming languages are supported for Web Services — and for consumption of them. In a time of scarce technical talent, this is a plus. Microsoft intends to be the first to offer a complete package that supports the Web Services concept.

Available now, or shipping this year, are:

- **development tools**
- **enterprise server software products**
- **a host of ready-to-use services (like an email Inbox, Wallet, Contacts, Filing, Device Properties and so forth)**
- **Microsoft applications, including Office, which are being Web Services-enabled.**

Sun

The Sun platform, ONE, is — as you might expect — based on J2EE. ONE is commercially oriented and offered primarily, although not exclusively, on the iPlanet Integration Servers.

J2EE — from multiple vendors, not just Sun — is the only alternative to the .NET/WinTel combination. That said, Sun's focus continues to be to try to prevent Microsoft from making inroads into the higher levels of computing.

Sun was later than most with its Web Services announcements. But it has picked up momentum recently, adding a Web Services Pack that includes XML-based Java APIs (JAX — Java APIs for XML) which is intended to foster integration between Web Services and J2EE such as:

- **JAXP (Processing), for DOM support, SAX, XSLT and schema processing support**
- **JAXM (Messaging)**
- **JAXB (Data Binding)**
- **JAXR (Registry)**
- **JAXRPC (Remote Procedure Calls), the mechanism for XML-based remote procedure calls in the Java programming language.**

Sun is taking the approach that developers need only write ordinary EJB components and the application server will do the rest. This is in stark contrast to the .NET approach, where Web Services support is found in the components themselves.

Sun's position on JAX and ebXML appear intended to slow adoption of Microsoft's C# and common language run time. However, the Sun story is not the only Web Services alternative in the Java market.

IBM

IBM has a strong Web Services platform for Java, aptly named the Web Services Architecture. Built around its WebSphere application server and WebSphere Studio tools (both IDE and downloadable SDKs), IBM offers a typically IBM-like infrastructure that supports all Web Services related standards — and not only the core ones (SOAP, WSDL and UDDI).

Unfortunately, 'typically IBM-like' means that the tools and installation are complex. This should not be surprising. In contrast to Microsoft, IBM's focus is on legacy integration with the Internet. This is readily explained: IBM has so many customers — with applications locked up behind firewalls in large data centers which are trying to escape — that any other approach would have alienated those customers. Indeed, many of these customers are also regular clients of the IBM Global Services division, for which Web Services offers a considerable upside in the way of future business.

The result is that IBM remains a clear choice whenever there are:

- **significant legacy integration requirements**
- **or complexity has to be addressed.**

On the down side, IBM:

- **is unlikely to appeal to those who want to start from scratch or to be 'lively'**
- **has committed to include Web Services support only in its Java/EJB tools and technologies.**

BEA

Another leading Web Services provider with good legacy capabilities is BEA. Its e-business platform is based on its WebLogic application server. BEA has recently announced Java Connector Architecture (JCA) support. JCA is significant because it permits EJBs to be populated from legacy artifacts, and EJBs are the likely components which will expose CORBA objects (and mainframe procedures) as Web Services.

BEA was later than one might expect with its Web Services announcements, having virtually nothing of significance to say until the second quarter of 2001. The JCA announcement is significant, given that WebLogic is the market leading J2EE certified application server.

Major vendors of packaged applications will be attracted to BEA because of this. It offers them a chance to portray a Web Services solution without cannibalizing their own monolithic suites and revenue models. JCA is the secret ingredient that makes the application server an integration solution. An integration platform that incorporates Web Services is a powerful construct.

If BEA has an Achilles heel in the Web Services arena, it lies in its lack of the equivalent, commercially accepted and comprehensive, development tools that Microsoft or IBM possess. Balancing this is that you can use most IDEs in association with BEA products. But, as discussed earlier, it appears that Web Services adds a sufficient number of distinctive requirements that most customers prefer tightly integrated Web Services development support to be available from their Web Services supplier.

HP

HP originally articulated the vision of Web Services — in 1999, as part of its then proprietary e-Speak architecture. Today, its *netaction* suite — based on the acquired Bluestone application server — is shipping. This is still based on the original e-Speak technology to some degree, and consequently is not as standards-based as many of the other competing vendors who started down the Web Services route that much later.

The HP architecture calls for SOAP and UDDI support, but the product today is simply XML-based. One benefit is that *netaction* supports C and C++ — in addition to Java. It can be used on Windows NT in addition to HP-UX and Linux. Furthermore, HP has added renewed life to HP Process Manager (the former Changengine product) — by casting this into the realm of Web Services.

Having made all these points in favor, HP does not have a software orientation or sympathy. One consequence is that it lacks the tradition of developer loyalty to its tools which Microsoft, IBM and BEA possess — which must limit HP's market appeal in the short and medium term.

Other players

Other well known players have announced their own, and different, Web Services initiatives. These include:

- **Borland**
- **Iona**
- **Oracle**
- **SilverStream**
- **Software AG**
- **Sybase.**

In addition a number of startups — like Asera, Bowstreet, Eltegra and WebCollage — continue to enter the market to exploit gaps and grab mind share. They do not, however, have hardware, operating systems or legacy software technology businesses to prop up. They are, thus, free to push the envelope with new concepts. On the other hand, few of these recent entrants have much in the way of revenues, which is not the case for Microsoft, IBM, BEA, Oracle, Sun, Sybase, etc.

Catches

Given all the hype and commitments from vendors of every ilk, there must be some catches. Do not expect any vendor at such an early date to have a complete architecture filled with all the constructs you need, or even a sufficient number to provide interoperability (see also page 18). A substantial investment in pre-purchase due diligence is prudent, if not mandatory.

We have had standards-based products for years, or so it has seemed. But remember how long it took for CORBA implementations of 'one specification' to become interoperable? This was not on account of pre-IIOP CORBA; it was due to the fact that vendors are competitive and the only air-tight specification is running code.

Three potential pitfalls or issues, which Web Services face, illustrate the difficulties:

- **UDDI pervasiveness: UDDI is a community process with over 200 members which is not likely to settle down and offer a tight specification anytime soon; people will have to start UDDI activity inside a company as a complement to DNS and LDAP, then move towards use within small, trusted, B2B communities before UDDI 'brokers' emerge — and you will have to decide if you want their content; balancing this, there is considerable potential for commercial exploitation (as per Yahoo)**
- **the volume of security/authentication requests skyrocket as finer grained functionality makes its way onto the Internet (PKI takes on a new level of importance, SAML (security profile exchange), XKMS (key management) and**

XTASS (authorization) are all under development); while security requirements are more complex (and certain to be based on policy management eventually), the ramifications of spurious purchase orders and wire transfers are worrying — EDI VANs or networks like S.W.I.F.T will be worth understanding for their lessons learned

- **service levels across a complex ecosystem of nodes on an inherently flaky Internet remain problematic; systems management vendors have been noticeably silent when it comes to Web Services — yet they are needed, not simply for use of Web Services in their platforms but to support failover, load balancing, alerting and instrumentation in Web Services enabled applications.**

Can .NET and/or J2EE stand up to Web Services requirements and expectations?

.NET can stand the load for smaller, non-critical projects today even though Microsoft has had to tune its architecture, and tools, more than J2EE needs to do. The tight integration of ASP/IIS/MTS/MSMQ and VB/VC/VJ — all basically COM+ centered — is producing a tightly coupled paradigm on top of (some argue within) the operating system itself; this is proving a wrenching challenge.

Yet history suggests that Microsoft is up to it. Microsoft's capability to deliver tools and infrastructure middleware is proven (if not often on time). This suggests that .NET will be a viable platform for serious projects in 2002 (it claims 2001, but that looks dubious).

The use of .NET will likely remain where Microsoft has always been strongest — in small to medium scale deployments or until proven otherwise. This is also where ease of use and fast development cycles are important. Host Integration Server will also play a pivotal role. This is Microsoft's entry point for its Web Services to link to legacy applications on legacy platforms.

J2EE would be a disappointment if not for IBM. A strong developer channel (a clear Microsoft strength) is needed for Web Services to fly. Where Sun has yet to tread, IBM has already trodden.

IBM already has a strong developer constituency, although this — with its legacy heritage — is very different to that owned by Microsoft. Despite this, it was IBM and Microsoft working together that produced several key standards for Web Services. Both IBM and Microsoft are committed to SOAP (along with many other vendors — with Sun only recently joining up).

IBM also supports all the key standards required for Web Services, though not always via J2EE directly. IBM has EAI middleware (MQSI/WebSphere Business Integrator) as well as wrappers for enterprise-level applications on CICS, IMS and Tuxedo.

BEA, with the marketing leading application server (in WebLogic), has stated its intent to implement JCA. This, potentially, will offer an alternative gateway for Web Services to link to legacy environments and commercial application packages.

Most other ISVs supporting J2EE have announced Web Services strategies for their product lines. Some platform vendors, like IONA with its iPortal XMLBus, have been shipping for some months. On this basis J2EE products for Web Services are ready for consideration.

Yet careful evaluation of product choices is still necessary since — with the clear exception of IBM — there really is no one-stop, multi-platform alternative yet. This is particularly true when comparatively large implementations, with many concurrent users, are the objective.

Management conclusion

A few months ago it appeared that Sun might fragment the Web Services initiative by supporting an alternative to SOAP (ebXML transport). However, Sun's J2EE application server partners were already shipping code that delivers Web Services on their application servers — using SOAP. Inevitably Sun had to support SOAP or see ONE shrivel — and arch rivals Microsoft and IBM walk away with the riches. BEA, IBM, Sybase, Oracle and IONA are among the other J2EE front-runners which, coming from a proven middleware heritage, provide decent offerings.

Microsoft is still ahead in terms of comprehensiveness — offering more tools, more discreet server products and more architectural skin in the Web Services game. But it is still a Windows-only scenario for developing and deploying your Web Services. Recent Linux porting announcements to the contrary, Microsoft has in the past made agreements (for example with Software AG) for large parts of Windows and DCOM/COM+ to be made available as third party platforms. None of these has had significant commercial impact on market acceptance or breadth of support.

While there are a number of specialists who have entered

the software market specifically to address Web Services, making a commercial success of distributing Internet-based components remains an unproven market — even though, logically, there is a demand for business processes, portals, and complementary tools, etc. Tracking partnership agreements with other vendors (especially the larger ones) will likely indicate which — if any — of these will survive.

That said, Web Services is no longer likely to be a passing fad — any more than distributed components are such a fad. Web Services are a natural evolutionary step for taking applications from behind the firewall to going through the firewall for business to business connections. On this basis, organizations should be preparing to select and experiment with Web Services development tools now — in preparation for tactical deployments of such projects in 2002.

In this context, first efforts should be directed inside the organization — where matters can be controlled. Subsequent steps should be through the firewall, in non-critical application scenarios.

The absence of standards for transaction management, security and service level management (for example, 'eventing' and management consoles) should not stand in the way. Opportunistic software vendors will plug gaps with solutions in due course.

On the other hand:

- ***strive to avoid lock-in to peripheral services***
- ***understand that the core to be protected is the fundamental Web Service itself***
- ***beware of Web Services components available on the market that are not based on SOAP, WSDL and UDDI.***

Even though useful and almost always attractive, proprietary Web Service components will defeat the intent of the 'open' vision. You will only be able to use them inside organizations deploying the same proprietary technology. But is that not where we started? Is that not what Web Services are trying to avoid?

Model Driven Architecture

Tom Welsh
Consultant

Management introduction

In March 2001 the Object Management Group (OMG) made what could be the most important announcement in its 12-year history. Unfortunately, the revolutionary nature of the new Model Driven Architecture (MDA) was thoroughly obscured by the statement's abstractions:

"In response to the growing and ever present challenge of enterprise interoperability," read the statement, "the MDA offers a full-lifecycle approach to solving the problems of developing, deploying and integrating existing distributed systems with emerging technology, assembling virtual enterprises that span multiple companies, implementing business intelligence solutions and enterprise information portals in a multi-vendor environment".

The automatic generation of application code from detailed models is an old dream. It goes back to the early days of Information Engineering,. It received a crushing setback with the collapse of IBM's AD/Cycle strategy in the early 1990s.

The question is whether software technology has now evolved to a point where MDA is a practical possibility. Surprisingly, as Tom Welsh explores, the answer appears to be a qualified 'Yes'.

Interoperability

Ever since its inception in 1989, the OMG has been dedicated to interoperability. For the past decade its slogans have been:

- **there will not be consensus on hardware platforms**
- **there will not be consensus on operating systems**
- **there will not be consensus on network protocols**
- **there will not be consensus on programming languages**
- **there must, therefore, be consensus on interfaces and interoperability.**

To this it has added one more affirmation. There will not be consensus on middleware platforms and component models.

Rather than trying to make disparate applications interoperable by imposing a standard format for the interfaces between them, however, the OMG has now become still more ambitious. It plans to make applications interoperable by deriving them from common design models — in its abstract technical terms, "distilling out middleware-independent semantics".

MDA specifications are based on a platform independent model (PIM) of business functions. The idea is that a single platform independent model can be translated to yield platform-specific models (PSMs) for a variety of middleware platforms — which in turn hide most of the underlying operating system and network details.

According to OMG, the benefits of MDA will include:

- **reduced costs throughout the application lifecycle**
- **reduced development time for new applications**
- **improved application quality**
- **increased return on technology investments**
- **rapid inclusion of emerging technology into existing systems.**

The chequered history of modeling

Ever since programs were first written, the programmer had to have both:

- **a clear overview of the problem to be solved**
- **the logical steps leading to a solution.**

These correspond, roughly speaking, to analysis and design. Analysis normally denotes the process of formulating the problem to be solved, without any indication of the methods to be used. Design follows on from analysis, and entails working out (in as much detail as necessary) exactly how the given problem can best be solved.

When problems were relatively simple, analysis and design could mostly be done with pencil and paper — or even in the programmer's head. Once the era of 'throwaway' programs had passed, however, it came to be recognized that software is a valuable asset.

But it is also perishable. Without someone who can understand its design, software cannot safely be modified. It then loses its most important characteristic. Complete, easily understood and up to date analysis and design documents are a vital safeguard against this eventuality.

In the late 1970s and 1980s, the principles of structured analysis and design were laid down by a number of 'methodologists'. By the mid-1980s, a few pioneers were starting to talk about object oriented analysis and design (OOAD).

Although each method differed from the others in details of notation, there was much common ground. Essentially, the idea was to draw standardized pictures of:

- **data structures**
- **data flows**
- **procedures**
- **and the like.**

If analysis and design could be carried out at this level, it made it much easier for everyone involved to participate. This included end users and project sponsors as well as systems analysts and programmers.

One serious problem that arose, however, was the diversity (and multiplicity) of methods. No consensus emerged, partly because new ideas were succeeding each other so rapidly.

Workstations and the CASE connection

The consequent fragmentation helped to constrain the use of analysis and design methods to a small minority. Instead, most programmers went on writing code straight from the requirements, sometimes with the help of flowcharts at least until Apple Macintoshes and IBM-compatible PCs started to appear on desks (soon to be followed by power-

ful UNIX workstations). All three delivered graphical software analysis and design as attractive applications, and so-called CASE tools soon proliferated.

The challenge was, nevertheless, to generate source code automatically from the design models. This idea became associated with Information Engineering and — according to James Martin, its best-known advocate — Information Engineering is "[t]he application of an interlocking set of formal techniques for the planning, analysis, design and construction of information systems on an enterprise-wide basis or across a major sector of the enterprise".

What could be called the 'first wave' of modeling technology culminated in IBM's AD/Cycle ambitions, which envisaged automated generation of applications. CASE tools from 'best of breed' vendors were to interface with IBM's Repository — which would be the central co-ordination point for all analysis and design metadata.

It was a great idea. Unfortunately it collapsed on account of several glaring weaknesses:

- **the process of agreeing standards and rolling out compliant products took too long**
- **AD/Cycle did not cater for emerging or non-IBM platforms (such as UNIX and Windows)**
- **development Repositories turned out not to be the 'modest extensions' of known database technologies that had been expected**
- **the whole apparatus, on top of everything else, was too complicated and too expensive.**

In retrospect, AD/Cycle — and, to be fair — Information Engineering shared one questionable assumption in particular. This was that modeling should be done on an enterprise-wide basis. The preparation of enterprise information models was a virtually impossible task for an organization of any reasonable size, unless it froze itself in time (not an action that most businesses were or are prepared to contemplate).

OOAD arises

In the early 1990s OOAD methods began to predominate. Again there were many of them, arguably too many.

The critical breakthrough came in 1996-1997, when three of the leading methodologists (Grady Booch, Ivar Jacobson and James Rumbaugh) agreed to merge their ideas. Subsequently their employer, Rational, submitted the resulting method to the OMG, calling it the Unified Modeling Language (UML).

While imperfect, like all compromises, UML was accepted by large numbers in the IT community — vendors and developers alike. For the first time the IT industry had a single, standard analysis and design method. By 1999-2000 all important modeling tools supported UML, allowing prospective purchasers to compare other aspects — such as:

- **reliability**
- **completeness**
- **performance**
- **price.**

OMG, MOF, UML and XML

At the same time, OMG members invested much time trying to tease out the implications of UML. For a start, they considered whether other meta models — in other words, modeling methods — might be needed in future.

To ensure their compatibility with UML, OMG adopted the Meta Object Facility (MOF). This provides a standard 'meta meta model' — a framework within which any number of consistent meta models like UML can be created and worked out.

Next on the agenda was a standard format for communication between modeling tools, repositories and other products that make use of meta data. Given the popularity (and power) of the eXtensible Markup Language (XML), this was a natural choice. The outcome was XMI Metadata Interchange (XMI), which encapsulates metadata within XML documents.

The Common Warehouse Meta model (CWM) was created — using MOF and XMI — to provide a standard means of exchanging metadata which described data warehouses. This was extended, eventually, to embrace relational, multi-dimensional and record-based data definitions, followed by more and more types of meta data.

In September 2000 the Meta Data Coalition (MDC), to which Microsoft had entrusted the development of its Open Information Model (OIM), voted to merge with OMG. This decision left CWM, like UML, as the only broad-based industry accepted initiative in its space. [MOF, XMI and CWM are essential for MDA but they are not described further in this analysis — primarily for reasons of space.]

The architecture

The OMG has published a diagram representing the MDA (Figure 8.1). This picture accomplishes several things:

- **it replaces the older Object Management Architecture (OMA) diagram (Figure 8.2); this does not mean that the OMA is no longer valid, or that CORBA is being relegated to the background — rather that the OMG is building a new layer of abstraction which goes beyond CORBA**
- **the MDA diagram becomes an official symbol of the OMG's work**
- **the diagram can be used to explain some of the basic properties of the MDA.**

At the center of the diagram is the MDA itself, in close association with UML, MOF and CWM. This is the world of platform independent models. In the next ring are all the middleware environments for which platform-specific models can be supported. Currently, the list includes CORBA, XMI/XML, .NET, Java and Web Services but any other environment could be added, in principle.

The third ring features a partial list of pervasive services that can be standardized through the MDA — directory, transactions, events and security, to name but a few. Lastly, the 'spears' projecting from the sides of the diagram denote the numerous vertical industry domains within which the MDA can be of value.

On the face of it, creating a platform independent model is fairly easy. All that is required is to construct a UML model representing the problem to be solved, in such a way that no assumptions are made about what hardware or software will be used to implement the model.

The OMG has placed no limit on how many levels of platform independent models can exist (it is simplest to assume only one). However some profiles — such as the EDOC Profile for UML described in the following section on specifications — may require more. There may, for instance, be a business model and a platform independent component view. Having completed the business model, that would be mapped to a component view by selecting appropriate business components.

The next step (Figure 8.3) illustrates the heart of MDA. By applying specific mappings to the platform independent model, it is translated into one or more platform specific models for particular middleware platforms — for example:

- **CORBA**
- **Enterprise Java Beans (EJB)**
- **the Simple Object Access Protocol (SOAP).**

From independent to specific

At this point some interesting points of detail emerge. The mappings used to convert a platform independent model into a given platform specific model will, in the first

Figure 8.1: Model Driven Architecture (Source: OMG)

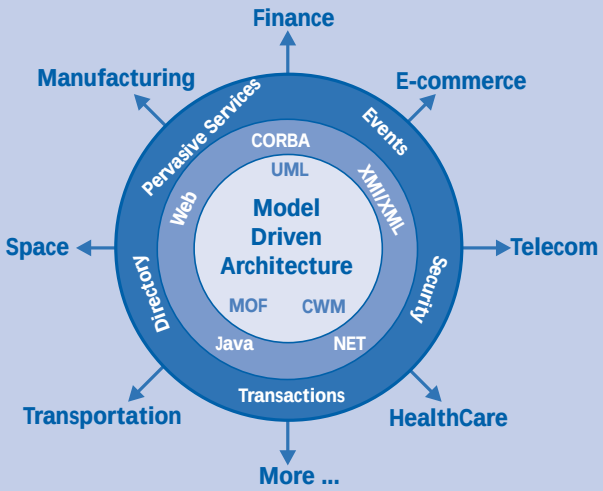


Figure 8.2: Object Management Architecture, a recap

The Object Management Architecture (from the OMG — the Object Management Group) envisioned a complete distributed environment. The foundation is built upon the concept of an Object Request Broker which:

- **manages communications between the OMA's components**
- **provides the basis by which objects interact in a heterogeneous and distributed environment.**

The desire is that objects should be independent of the platforms on which objects sit. In performing its task an Object Request Broker (ORB) relies on the OMG's definition of Object Services which, in turn, are responsible for:

- **creating objects**
- **providing access control**
- **keeping track of relocated objects**
- **etc.**

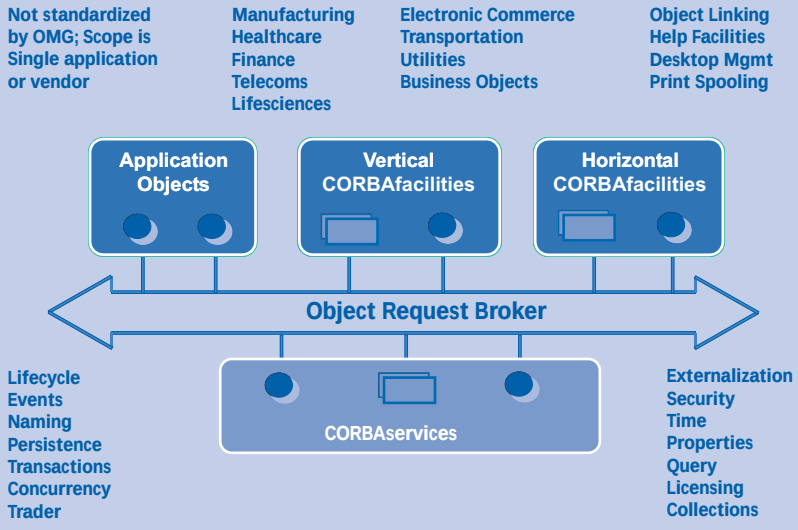
Common Facilities and Application Objects are the components closest to the end user. Their functions invoke services of the system components.

The Object Model defines common object semantics for specifying the externally visible characteristics of objects in an implementation-independent way. In this model, clients request services from objects through a well-defined interface. This interface is specified in the OMG's Interface Definition Language. Clients access an object by issuing a request to the object. The request is an event, and it carries information including:

- **an operation**
- **the object reference of the service provider**
- **parameters (if any).**

The Figure below shows the main components of the ORB architecture and its interconnections. The central component of CORBA is the Object Request Broker (ORB). It encompasses all of the communication infrastructure necessary to identify and locate objects, handle connection management and deliver data. In general, the ORB is not required to be a single component; it is defined by its interfaces. The ORB Core is the most crucial part of the Object Request Broker; it is responsible for communication of requests.

The basic functionality provided by an ORB consists of passing the requests from clients to the object implementations on which they are invoked. In order to make a request the client can communicate with the ORB Core through the IDL stub or through the Dynamic Invocation Interface (DII). The stub represents the mapping between the language of implementation of the client and the ORB core. Thus the client can be written in any language as long as the implementation of the ORB supports this mapping. The ORB Core then transfers the request to the object implementation which receives the request as an up-call through either an IDL skeleton, or a dynamic skeleton.



instance, be written by experts in the target platform. Normally, quite large pieces of work will turn out to lend themselves to automation — for instance, setting up EJBs or when to use entity beans rather than session beans. Eventually, some mappings may be completely automated (or nearly so).

Also, provided that the platform independent model is detailed enough, there will be no need to make changes to the platform specific model, or the code that is generated from the platform specific model. Instead, all maintenance and application changes are performed at the platform independent model level.

The prospective benefits are enormous. Testing can be carried out at the earliest possible stage, saving huge amounts of rework. In addition, all the platform specific models derived from a single platform independent model — and all the applications generated from the platform specific models — should:

- **be functionally identical**
- **eliminate a whole class of software defect, especially those arising from the subtle inconsistencies that arise between different versions of a program.**

The next step

Th next step is generating source code from the platform specific models (Figure 8.4). This may not be as hard as would be expected — for a number of present-day modeling tools are already capable of generating code from UML models for CORBA, EJB and other types of distributed application.

The advantages of the MDA approach are now obvious:

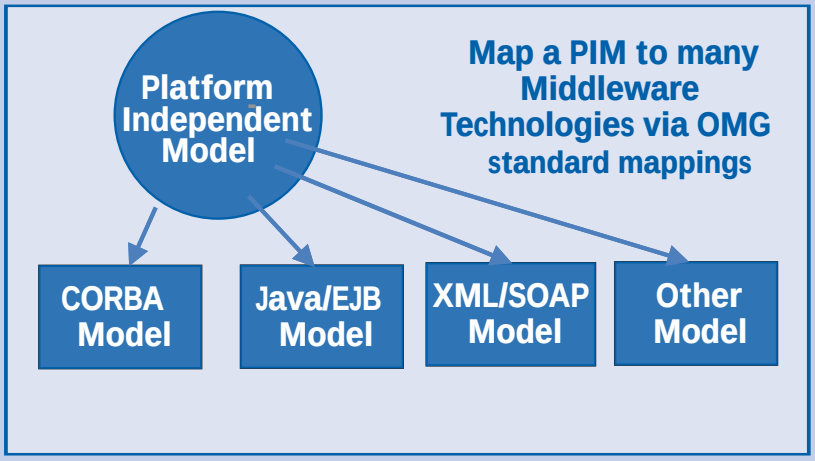
- **applications and data models need to be created once only, and can then be centrally maintained or enhanced with a minimum of additional effort**
- **changes to the core business logic (and data) are made in one place only, and are then automatically propagated to all deployed applications that rely on that logic**
- **once written, a single application can be semi-automatically deployed to CORBA, .NET, Java 2 Enterprise Edition (J2EE) and other environments.**

All deployed applications based on a single platform independent model are guaranteed now to share a single, consistent version of the business logic and data. Among other advantages, this means that they should interoperate.

Specifications and current UML evolutions

When trying to come to grips with the MDA, it is easy to

Figure 8.3: Mapping a PIM to multiple PSMs



feel that (as Gertrude Stein said about Oakland) "there's no 'there' there". At present about a dozen OMG specifications underpin the MDA — there will be many more in future — but not a single one of these was written with the MDA specifically in mind. That is because they are all responses to Requests for Proposal (RFPs) that were issued before the MDA announcement in March 2001.

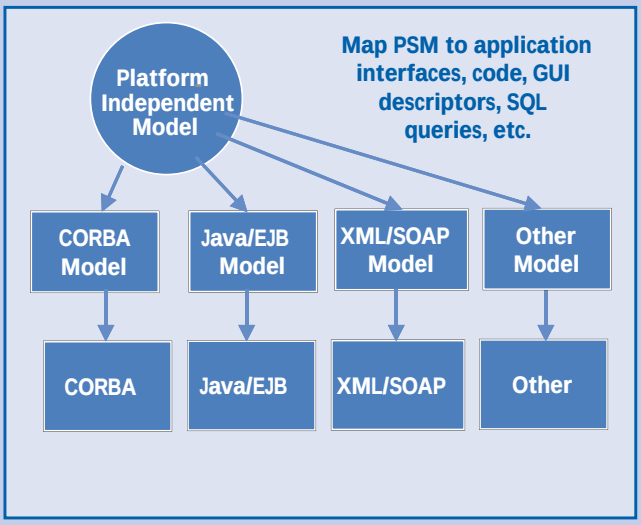
It cannot be over-emphasized that the MDA is a logical extension of the OMG's previous work. Indeed, it can be seen as a 're-factoring' of much the OMG has attempted, since the adoption of the UML specification in 1997.

- **diagram interchange.**

Work has been going on for some time in these areas — and is expected to continue throughout 2002. Support for the Model Driven Architecture is now one of the project's highest priorities.

UML evolved from a variety of methods, many of which had distinctly data-oriented pedigrees. So it should not be a surprise that it does an excellent job of describing data structures but is not 'computationally complete' — in other words, there are (many) programs that cannot be expressed in UML.

Figure 8.4: Generating source code from multiple PSMs



Other than press releases, presentations and an introductory white paper, the only document that concerns itself exclusively with MDA is the 31-page MDA Definition (OMG reference ormsc/2001-07-01). Written by the Architecture Board's Object Reference Model Sub-Committee (ORMSC), this is the authoritative account of what MDA is and is not.

UML has gone through several minor revisions since being adopted as a specification in 1997. Now it is going through a much larger set of changes which will eventually result in UML 2.0. In all, 37 letters of intent were received from organizations wishing to contribute their ideas, and UML 2.0 has been split into four pieces:

- **infrastructure**
- **superstructure**
- **Object Constraint Language (OCL)**

This is where Action Semantics are introduced. The submission on this, currently being evaluated, replaces all the UML constructs that relate to actions and computational behavior. It will allow the construction of both platform independent models as well as platform specific models — with platform specific models that can be executed (by special interpreters) to test their validity and completeness.

In addition, another submission, the Human Usable Textual Notation (HUTN), describes how a text language can be automatically generated from a MOF model. The objective is a mechanism by which developers and domain experts may read the contents of MOF models with relative ease.

UML Profiles

Throughout, the OMG has tried to keep UML itself as sim-

ple and uncluttered as possible. It has accomplished this by defining overlapping extensions which can be used when necessary. Such an extension is called a UML Profile, and consists of a subset of UML (which may be the whole) together with additional constraints, standard elements and semantics.

The first profile to be adopted (unsurprisingly) — and the only one so far — is the UML Profile for CORBA, which provides a way of expressing the semantics of CORBA IDL in UML notation. The specification is based on Rational's "Rose CORBA" (part of the Rose98i Enterprise Suite).

There are, however, other profiles currently working their way through the system, including:

- **a UML Profile for Enterprise Distributed Object Computing (EDOC)**
- **a UML Profile for Enterprise Application Integration (EAI)**
- **a UML Profile for Schedulability, Performance and Time.**

The EDOC profile is a massive specification, 447 pages long, which builds on the International Standards Organization (ISO) Reference Model of Open Distributed Computing (RM-ODP). Among other things, it defines mappings to EJB, Java and the Flow Composition Model (FCM). Profiles and mappings for the CORBA Component Model (CCM) and Web Services are planned.

The EAI profile addresses loosely-coupled distribution through products like:

- **IBM's WebSphere MQ Integrator**
- **Java Message Service (JMS)**
- **the C, C++ COBOL and PL/I languages.**

The Schedulability profile defines UML extensions for real time and embedded systems. This will include Quality of Service (QoS) concepts.

In July 2001 the Java Community Process (JCP) announced the completion of its UML Profile for EJB. What is notable about this is that it is the first such specification to be created outside the auspices of the OMG. It is not however, likely to be a diversion: most of the 11 companies primarily responsible for the work are also members of OMG.

Software Process Engineering Meta model (SPEM)

Unlike most OMG specifications, SPEM relates to the software development process. It deals with collaboration between roles that perform activities to create work products. Like UML and CWM, SPEM is defined as a metamodel; it is also a UML profile.

SPEM supports:

- **the Rational Unified Process (RUP), arguably the closest thing to a standard development process that exists today**
- **the DMR Macroscope**
- **IBM's Global Services Method.**

Future specifications

Two important kinds of MDA specifications are anticipated: Pervasive Services and Domain Facilities. These correspond quite closely to the present-day CORBAservices and CORBAfacilities.

Pervasive Services will include important optional pieces of functionality such as Directory, Security, Transactions and Persistence. If these can be defined once and for all at the platform independent model level, a great deal of time and effort can be saved (and scope for error avoided) at the specific implementation level.

Domain Facilities refer to vertical industry models, previously applicable only to CORBA, which under MDA will become available for EJB, Web Services, .NET and other important forms of middleware. The Healthcare Resource Access Decision Facility has already been implemented in Java and EJB as well as CORBA; other domain task forces are expected to follow suit.

Market implications

Over a dozen vendors (and several other organizations) have already committed to support the MDA. Several of these vendors are already marketing products that at least resemble the MDA in various ways.

Obviously, no product can yet comply fully with the MDA, as the specifications are not finished. It is, however, worth mentioning:

- **Adaptive, whose repository is MOF-compatible**
- **Interactive Objects Software; its ArcStyler tool closely mirrors MDA's lifecycle**
- **Kabira Technologies**
- **Rosch Consulting**
- **Secant**
- **Softeam.**

Project Technology and Kennedy Carter already offer modeling and code generation tools based on the methods worked out by Sally Shlaer and Stephen Mellor of Project Technology. These products perhaps come closest to the spirit of the MDA, although they were designed several years ago and omit the important platform specific model stage. They generate code directly from the platform independent model.

Although no announcements have yet been made, it is believed that some of the biggest players participating in the OMG — such as HP, IBM, Oracle and Sun — are working quietly to support the MDA in their product lines. The adoption of the MDA industry-wide seems the more likely because the new architecture is so firmly based on UML and MOF. As such, it represents an evolutionary rather than a step change.

Is the MDA good?

Clearly the OMG thinks so, as it did of CORBA. According to the OMG's technical director Andrew Watson, there are two fundamental business arguments in favor of the MDA — future proofing and the ability to roll out new forms of middleware without effort and disruption.

Future proofing matters. Today we can ask: 'five years ago, did Java or XML figure in your plans?' Of course not. What will occur in the next five years that you cannot possibly predict now? Lots. Being capable of accepting change is ever more important. The opportunity is to avoid, or reduce, 'technology dissonance'.

The ability to roll out new forms of middleware without effort and disruption will appeal to CTOs and IT directors. It has more to do with delivery than development. It does not take great understanding to accept that Java or XML or .NET are not the last waves of innovation that we will see.

It is high time that there was a standard notation for writing down business logic and data structures in a platform-independent way. After all, business logic is coming to be recognized as an important form of intellectual property. Yet, in most organizations, it is embedded in reams of incomprehensible low-level code.

Without hours of effort, even qualified programmers cannot figure it out. And every time a new system has to be deployed, or a merger or take-over or other business event imposes a new platform, everything has to be done over again. These are sufficient reasons to justify the MDA, before even considering the development impact.

Management conclusion

Ten years ago 'platform' meant a combination of hardware and operating system. Today there are fewer operating systems, but the term 'platform' has been extended to include middleware as well as operating system-neutral environments (like Java). One effect has been to increase the need for something like the MDA, which promises to generate platform specific models from platform independent ones that contain little more than business logic.

If OMG can deliver on the promise of the MDA, we could be on the verge of a new era in computer programming, as different from today's practices as Java is from assembly language. Better still, there will be no need to cut over sharply from the old way of writing software. With the help of 'reverse-engineering' tools, existing source code can be translated to UML models which can then be used as the basis for new designs.

Gradually, at their own pace, organizations should be able to enter the world of the MDA and focus their efforts on writing and maintaining business logic instead of coping with the myriad distractions of rival operating systems, databases and middleware that are so beloved of programmers (although not businesses). And herein lies the rub. The MDA is clearly a 'good idea'. So was AD/Cycle, at least conceptually — and the MDA has already gone far past that.

But the development community is conservative. It knows what it knows and does not like change. You only have to examine the extreme reluctance of synchronous programmers to think asynchronously to understand resistance. However virtuous the MDA is, its success will depend on it working ***and*** *being inexpensive (q.v. AD/Cycle)* ***and*** *being acceptable to those who will have to use it. Fail on any one of these and it will likely fail on all — irrespective of its other virtues.*

Yet, let us offer the last word to Microsoft. Ever since it attained any prominence in 1994-1995, CORBA has been the target of constant sniping from Microsoft and its allies — presumably because it was seen as competing with Microsoft's own Distributed Component Object Model (DCOM). In view of Microsoft's enormous influence with analysts and the media, this was a substantial handicap (for CORBA). The MDA is unlikely to meet with such overt resistance:

- ***partly because it is such a logical step forward from UML***
- ***partly because Microsoft has always been a firm supporter of UML.***

Financial MIDDLEWARESPECTRA
Enhanced Intranet Subscription

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:

- ✓ 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 versions of the 14 volumes of the *Issues in Middleware Collections* (collectively priced at over US$7,500/Euros 8,000 if all bought individually)
  - ➢ 5 printed copies of each quarterly Report (these copies can be mailed to separate addresses)
- ✓ only US$4500 or Euros 5000 (saving over US$3,000/Euros 3500 if purchased as an Enhanced Internet Subscription)

Updated ■ Middleware In Action Collections (5 Volumes, 449 pages in all: US$95 each Volume; US$245 for all five)
Updated ■ Middleware Architecture Collection (637 pages, $795)
Updated ■ Strategic Issues in Middleware Collection (748 pages, $895)
Updated ■ Distributed Systems and Middleware Collection (791 pages, $795)
Updated ■ Middleware in Distributed On-Line Transaction Processing Collection (504 pages, $995)
Updated ■ Middleware in Data Warehouse, Database and Database Access Collection (225 pages, $375)
Updated ■ Messaging and RPC Collection (514 pages, $475)
Updated ■ Queuing Middleware Collection (436 pages, $495)
Updated ■ OO and Middleware Issues Collection (369 pages, $395)
Updated ■ Middleware and Application Development Collection (537 pages, $475)
Updated ■ Enterprise Application Integration and Middleware Collection (308 pages, $395)
Updated ■ Internet and Middleware Issues Collection (426 pages, $495)
Updated ■ Middleware and Message Broker Collection (240 pages, $395)
Updated ■ Middleware perspectives (46 pages, $150)

To order your MIDDLEWARESPECTRA Enhanced Intranet Subscription, call:

(in the USA/Canada) 1-763 502 8819
(Europe/Rest of the World) +44 1962 878333

or email:
spectrum@middlewarespectra.com

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

USA and Canada:
Spectrum Reports, Inc.

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

UK and Rest of the World:
Spectrum Reports Limited

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

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

Email and Internet

Email:
spectrum@middlewarespectra.com

World Wide Web:
www.middlewarespectra.com

ISSN 1460-7220