Introduction

In 2002 I was employed to advise Chase retail bank on its Web services strategy (my eventual recommendation was that it didn't make sense for it to have one). As part of the work, we interviewed people in different parts of the Chase Empire: deposits and loans; credit cards; debit cards; and mortgages. They all had big offices in impressive buildings in downtown Manhattan. Then we needed to interview retail financing. This is the department that provides loans for cars and small home improvements. They weren't in Manhattan, or even New York, but on Long Island. They weren't in an impressive building; in fact they were in what we in the UK would call a Nissan hut. We started the interview by trying to explain what Web services were and why they might be important to Chase. The line we took for this was that they could be used for integrating the many different information systems (a legacy of mergers between Chase, Chemical Bank and Manufacturers Hanover over many years) in service at the bank. "Aha", said the patient financing manager, "in that case we don't need to say any more because we integrated all that last year." We were dumbfounded, as to our knowledge integration was a continuing problem at Chase, as at every other large bank in the Western world. "How was that done?" we asked. "We did that with the portal" he told us. He had been sold, by the IT department, that participating in the portal project would solve the Chase integration problem.

There seems to be a lot of confusion about integration, the cynical among us would say that this was deliberately fomented by the marketing departments of some vendors (and sometimes those advising IT departments). However, in fairness to the vendors and consultants, the confusion is probably deeper than that. There is an almost irresistible temptation, when faced with the problem of integrating two existing information systems, to solve it by building a third application. But the sad truth is that such a 'composite' application actually requires an integration capability to be in place before it can be built, because it itself is an application that is to integrate with two others. It helps to be clear what the aim of integration is. Another story will help here. We were living in New York and my wife's nephew Diarmuid came to stay, from Ireland, for the summer and needed a job. The Fitzpatrick hotel chain in New York is Irish and my wife was able to use a friend of a friend to get Diarmuid work there, from Memorial day to Labor day. He duly arrived, watched the Big Lebowski on the DVD player (which he watched for every night throughout the summer) and went the next morning to his job. That night we asked him what his job was. "My job is to take the reservations entered on the Web site during the day and enter them into the actual reservation system" he said.

Figure 1 Diarmuid's summer job

There, in a nutshell, is the objective of most integration, to eliminate Diarmuid. Some people call this the elimination of swivel chair integration, so called because Diarmuid had to interact with one system to read a reservation and then swivel to another to enter it. Once all the swivel chair integration has been eliminated, it would then make sense to go on to eliminate batch processing (except for data integration). And once all the batch has

been eliminated, it would then make sense to enable composite applications and orchestration. But for now, in all the enterprises I deal with, eliminating Diarmuid is the real need.

The integration needed to replace swivel chair is really very simple. But you will need some convincing that this is the case. We have to convince you that the only scenario to be covered is that of a message leaving one information system as a result of a transaction completing, and then entering another information system to start a transaction. This is shown in the diagram below.

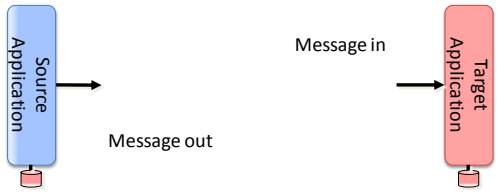

Figure 2 The only integration scenario

This is a 'fire and forget' scenario. If the message out from the source application fails to arrive at the target application, then it is not the source application's problem to re-deliver it. Similarly, if the target application does not successfully apply the message, that is not the source application's problem to fix (just as when Diarmuid entered a reservation, if there wasn't in fact a spare bed, that was Diarmuid's problem, not in the Web reservation system's problem).

If the source application does want the target application to do something, then the 'three transaction' pattern has to be used. This case is a minimal orchestration problem and needs at least two integration interfaces to be used. We call this the 'three transaction model' as shown below.

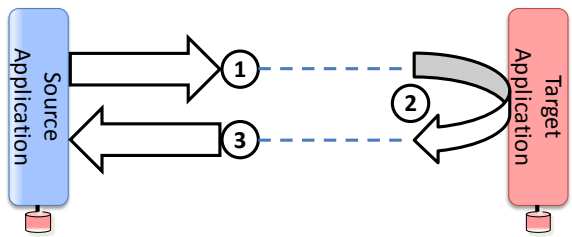

Figure 3 Three Transaction Model

This scenario is, for instance, the one used by the Base24 application that manages the majority of ATM transactions in the world. When a request to dispense, say, £100 arrives at the ATM device handler (the source application), it starts a timer, enters a record in the database and then emits a message towards the bank account (the target application). That is transaction one. The bank account listens for messages from tellers, automated ones in this case, and authorises the transaction and emits a message to that effect. That is transaction two. Finally, the device handler is listening for messages from the banks and when it receives one, it looks up the record in the database, cancels the timer, and uses the information from the database to connect to the ATM session and dispense the cash.

In order for this to work correctly, we have to be using transactional messaging. That is, the PUTs and GETs to the queues implied by the processing, have to be part of the two phase commit transactions at the applications. This is described in more detail in the basic engineering paper.

The term application used above is a little too imprecise. In the application paper we distinguish between the user interaction part, which we call the application, and the resource management part (combining business and data logic), which we call the resource manager. The combination of an application, resource manager and agents to use the application is an information system. It is very important that we convince you that the integration is between the two resource managers, not between a application and a resource manager or an application to an application. These scenarios are illustrated below.

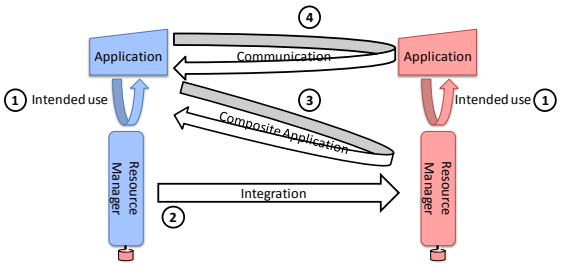

Figure 4 Interaction Scenarios

The diagram shows four different scenarios.

1. Intended Use. This is where the application part of the information system (that is, the agent using the application and the interaction logic) makes stateless requests of the resource manager for which it is written. If the request is for a change then that is business event handling, if the request is for information then that is business content handling.
2. Integration. Where a business event in one resource manager causes a business event to occur in the other.
3. Composite Application. Where an application intended originally for use with one resource manager also interacts with another. Note that it is not possible for a transaction (PUT or POST) originated in the application to span two resource managers (distributed two phase commit is not allowed in this architecture, see the paper on BASE versus ACID).
4. Communication. If the application part of one information system interacts with the application part of another, then that is communication, not business event or content handling.

What this shows is that, for the message passing to be integration, the message has to go from the resource manager of one information system to the resource manager of the other (business logic to business logic). This is a golden rule of integration.

You may be wondering, what about the scenario where the source application requests some information from the target? This is illustrated below.

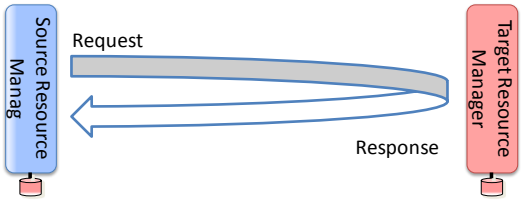

Figure 5 Forbidden Scenario

We have captioned this 'the forbidden scenario' because it violates a golden rule of information systems design which is that each resource manager is autonomous and asynchronous. This golden rule ensures that systems can scale and that systems are loosely coupled. The reason this scenario violates the rule is that the request and response is clearly synchronous, violating asynchrony, and the semantics of the request has to be understood by the responder and vice versa, violating autonomy. Nevertheless, I often see this scenario in use, for instance using RPC as an integration approach. What tends to happen is that, as the systems scale, the source RM finds that it cannot wait for the target RM so it caches responses. However, source RM now no longer knows when its cache is stale, so it asks for a message from the target RM when the target changes. And that gets us back to our original integration scenario.

We have now, I hope, established the following. Integration has only one scenario, which is when an asynchronous message, representing a business event that has completed, is emitted by one resource manager and causes a business event to be started at another resource manager. So, the essence of integration is causality. Indeed, in the front middle back view of the value chain, it is integration that allows an event in the front to cause an event in the middle, see the front middle back paper.

Now that we know what integration has to

do (get a message from a source to a target), we need to analyse how integration does it. In short, I believe that we need to achieve both interoperability and also loose coupling. Interoperability means that there is a physical path from the protocols and formats of the sender to the protocols and formats of the receiver. This is probably non-controversial. Loose coupling means that if we change the sender we don't have to change the receiver. For me this is a semantic problem. Converting protocols and formats (say XML over SOAP to JSON over HTTP) is a syntactic problem – I don't have to understand the contents of the messages to do this. On the other hand, to achieve loose coupling I have to be able to transform the contents of the messages. For instance, if the source uses the XML <name>John Schlesinger</name> and the target uses <given name>John</given name> and <family name>Schlesinger</family name> I have to know enough about the meaning both to be able to transform one to the other. In fact, I would go so far as to say that the real problem of integration is semantics, not interoperability. It is as though I was changing into my running gear to run a marathon and found that the changing room was locked. It looks like my problem is getting out of the changing room, but my real problem is to run the marathon. Interoperability is a relatively small problem we have to get past in order to tackle the real problem of managing semantics. Separating syntax (interoperability) from semantics (transforming messages) is at the heart of integration.

The golden rule that ensures we also make the separation is what we call the rule of three flows and two transforms. Let's start with the flows.

I have been careful to talk about messages representing business events. A business event is a coarser grained thing than, in general, an API to a resource manager. For instance, SAP can emit IDOCs (intermediate documents) to enable event based integration of SAP with either other SAP systems or with non-SAP systems. When Lufthansa integrated their SAP system with an aero-parts exchange for planned maintenance, they found an IDOC that was exactly for the event they wanted.

However, two important pieces of information were missing. So they wrote a little ABAP process that ran when the IDOC was triggered, transformed the IDOC to the SITA format the exchange wanted, got the two extra pieces of information and put them in the SITA message and then sent it. This turned the API provided by SAP into a Lufthansa business event. An IDOCs are the coarsest grain interface to SAP (there are over twenty kinds of interfaces to SAP with IDOCs, RFCs and BAPIs being the ones most commonly used for integration). In general then, to get a message representing a business event out of a resource manager we are going to need two things: an API that allows us to be triggered by the completion of a business event; and a stateless flow that fields that trigger and enriches it as required. This is illustrated below.

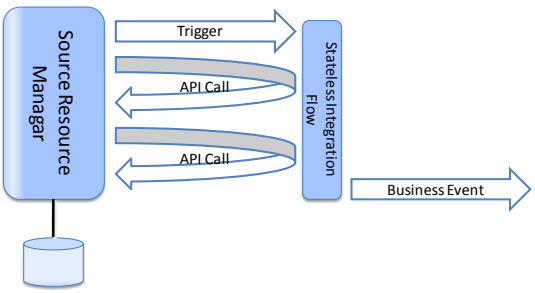

Figure 6 Generating a Business Event

The reason the flow is called stateless is because each time it runs it remembers nothing from its previous invocation. This

is true in almost all cases. The exception is when the trigger for an event is an adapter that polls a database table for an event. In this case, the flow that runs is stateless but the polling is not. The polling part has to remember the high water mark of the table from when it last polled (to ensure that it sees all events and doesn't process an event twice). The polling part is doing what the SAP IDOC manager does when it triggers a ABAP function module.

Getting an event into a resource manager is the same thing but the other way around, as shown below.

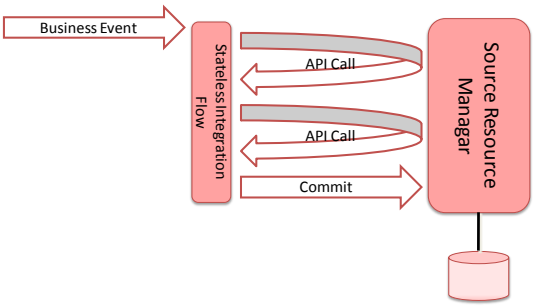

Figure 7 Handling a Business Event

As it may be necessary to make more than one update to a resource manager to complete the handling of the business event, in general the stateless flow has to run as a transaction. For example, when I was at iWay Software we developed SAP implementations of 15 OASIS business messages. The flows, on average, took five RFC calls per message to handle the message with a maximum of twelve and minimum of two. One message had to make two updates to SAP. At iWay we could run the database, MQ, JMS, MSMQ, SAP, IMS and CICS adapters transactionally. However, do not make the mistake of thinking that an adapter can run as a two phase commit between receiving the message and committing it. I used to think that was a good idea but having to design an adapter framework soon corrected that view, see the basic engineering paper for more information.

We now have two flows, the one to get a message out of a resource manager and, symmetrically, the one to get a message into a resource manager. This is illustrated below.

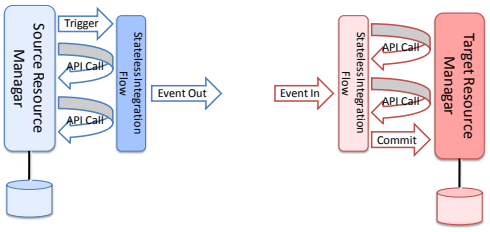

Figure 8 Business Event Flows

The golden rule associated with this diagram is that the flows that generate and handle business events are owned by the resource managers that they generate or handle events for. This golden rule was the one violated by the original enterprise application integration products. For example, it was typical when I was working for SeeBeyond for a collaboration to be written that did the following.

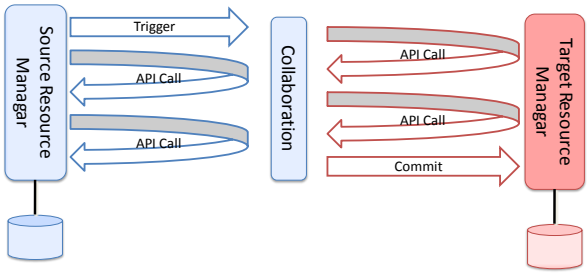

Figure 9 Enterprise Application Integration Approach

This was not just the way SeeBeyond did it, so did most of the EAI tools and certainly the most commonly used one, COBOL. This approach grossly violates the ownership rule and therefore frustrates loose coupling. However, we still need a flow in the middle because we need to be able to route messages from multiple sources to multiple targets. For example, the Merrill Lynch straight through processing hub had twelve sources and about six targets. We will discuss the ownership of the middle flow later, but here is a diagram of the middle flow routing a message to two destinations.

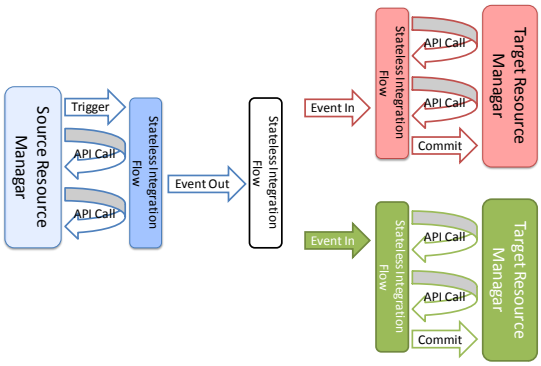

Figure 10 Middle Flow for Routing

An example of this kind of fan out is when an event is routed both to the next stage of the value chain and also to a data warehouse. Another of violating of the golden rule of adapter ownership occurred when I was consulting at Merrill Lynch I was asked by the CTO to look at a system they had built for integrating the bond trading platforms with a joint venture called BondHub. The idea was that bond traders would ask BondHub for a bond with certain characteristics (yield, maturity, and rating) and this would be relayed to the market makers who would respond with quotes. BondHub gave the best three to the requester who could then trade with one of the three responders. To make this work, the Merrill Lynch BondHub application had to interface to three bond trading systems. To do this, the developers created interface adapters to each system as shown below.

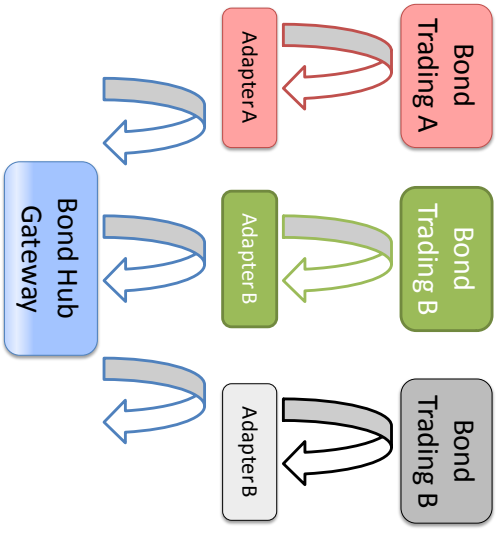

Figure 11 BondHub Architecture

The problem that the CTO wanted me to look at was that BondHub was very brittle, it kept failing. It turned out that any time the bond trading systems changed they broke the adapters. As these trading systems changed all the time, the adapters were constantly breaking and so causing BondHub to fail. The solution was to create one more adapter, a technical issue, and then changing the ownership of the existing three adapters, an organisational issue and, finally, putting in place agreements between BondHub and the three trading systems, a cultural issue. In general, I find that integration is much more a cultural and organisational problem than a technical problem. After these changes the architecture was as follows.

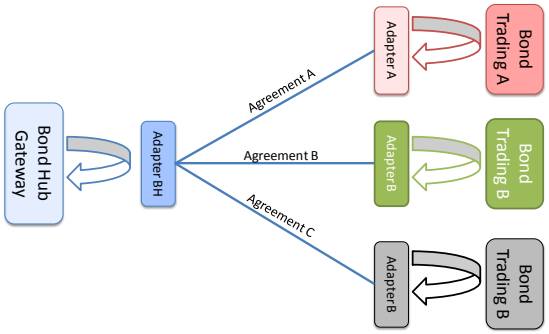

Figure 12 BondHub Fixed

The problem we still have is that, even if the source and target resource managers agree on syntax (they are both using HTTP and JSON say), they are unlikely to agree on the format of the message. Indeed, if this is a value chain causality interface, the event out could be a different event from the event in. For instance, when a foreign exchange trade is executed it causes a change to the real time risk of that trading desk. Similarly, the form of a trade execution is not the same as the form of a data warehouse load. We have our three flows but we don't have a transform yet from the source to the target format.

The naïve approach is to insert a single transformation from source to target. This achieves the aim of interoperability, it completes the path from source to target, but it does not achieve the aim of loose coupling. To illustrate this consider the

case above of one source and two targets. Assume we put the transform after the routing flow or we have to have two routing flows. When the source changes, we must change the transforms for both targets. We are also going to have to change the routing flow to accommodate the changes to the message. In order to achieve loose coupling we need two transforms, one before the routing flow to put the incoming event into a standard form, and one after the flow to put the event into the target form. Now we have loose coupling. This is illustrated below.

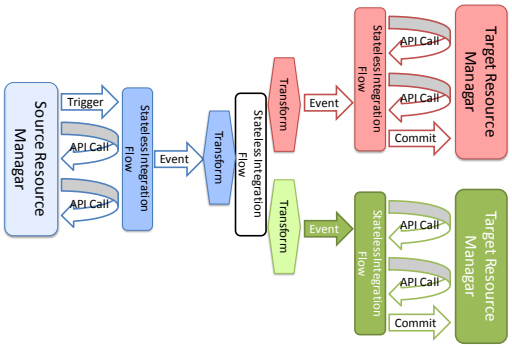

Figure 13 Inserting Transforms

For each interface, from source to target, there are now three flows and two transforms. This is more clearly illustrated below.

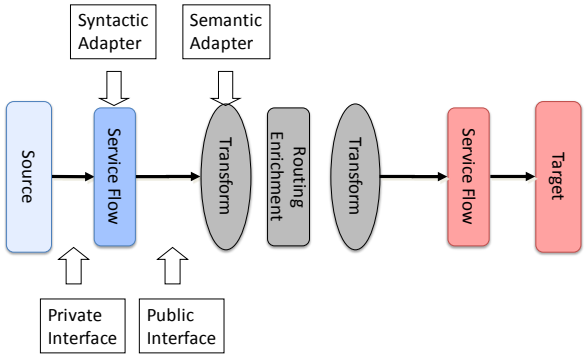

Figure 14 Three Flows Two Transforms

It is tempting at this point to declare the middle part of the diagram above to be an enterprise service bus and conclude that the problem of integration is now solved. This, indeed, is precisely the specious argument of those that push the term ESB. Unfortunately, there is a problem. Integration is an organisational and cultural problem. There is a limit of scope of such a broker. For it to be possible to broker between two applications, the owner of the broker must also be the owner of the source and the target. This is because the essence of such semantic broking is that the agreement is between the end points and the broker, not between the end points. But for that to work, both end points must completely trust the broker. Such complete trust is only conferred by ownership.

The term I like to use for a broker that manages transformation and routing is a semantic hub. The limit of scope of a semantic hub is a domain of ownership (see the one level enterprise paper).

There is also a strong tendency, once the concepts above have been grasped, to build the hub straight away. This, at its worst, is the 'build it and they will come' approach. Unfortunately, in general, they won't. In any case, you cannot safely build the hub if you don't have an organisation to run and manage it (integration is an organisational problem after all).

In the general case, then, the integration end points are in different domains, the domains have not been built and there are no hubs. In this case we have to integrate using the point to point approach shown below.

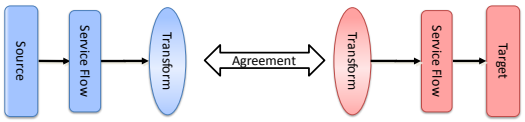

Figure 15 Point to Point Messaging

The diagram shows the approach that we took for BondHub. There is a missing middle flow, because the message is only going to one place, but there are still flows at each end and two transforms. This is the approach you have to use as you build up integration between end points in different domains. However, the end goal of integration, once the architecture is fully realised, is to be able to put any service (that is, resource manager) on any channel of access. To achieve that, the domains have to implement both hubs and

gateways. Point to point messaging cannot be eliminated between domains, but what it looks like when the enterprise is fully integrated is shown below.

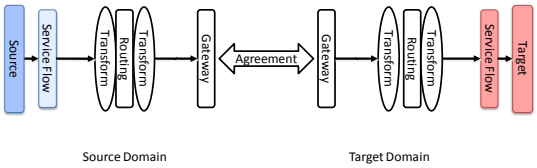

Figure 16 Fully Integrated Messaging Between Domains

About John Schlesinger

John Schlesinger is a Principal at Atos Consulting where he leads its Enterprise Architecture practice. John is an advisor to enterprises specialising in middleware and integration architecture. He has lead integration architecture development in retail banks, investment banks, retailers and manufacturing, both for integrating applications and for integrating information.

John has worked both as a consultant and also as a developer with software companies. He has taken over two dozen program products to market at IBM, Information Builders, One Meaning, SeeBeyond and iWay Software. These products included the world's most successful commercial software (CICS) and the world's most successful data middleware (EDA/SQL). John also led the Architecture department at Dun and Bradstreet when its IT department went global.

A member of the ACM and the IEEE, John has an MA in Physics and Philosophy from Oxford University and a Post Graduate Diploma in Software Engineering from Oxford University.

John has spoken at numerous conferences including the CIO Cruises run out of New York, during one of which he was the first speaker on after the collapse of the World Trade Towers in 2001.

John can be contacted at john.schlesinger@atosorigin.com