Enterprise Application Integration Platforms 101

Introduction

Application integration at its simplest is often just one application sending a message to another to notify it of some change. Perhaps when a patient arrives at a hospital, the registration system will send a message to clinical systems so they have all demographic data ready to use. Of perhaps it is just a nightly file transfer from the sales system to the accounting system.

But modern application integration platforms or suites can do a lot more than this to help applications work together and add real value to the enterprise.

Lots of terms have come into use over the years to describe different aspects and styles of integration, such as ESB, SOA, messaging, interface engine, Business Process Orchestration, composite applications, and many others. This paper briefly puts these terms in context and describes how they fit together to provide different levels of sophistication in your application integration solution.

Some products address all or most of these areas, others address one niche, while others come as suites of products that collectively address the majority of the concepts

Connectivity

There are many protocols used to communicate between systems, and integration platforms need to provide adapters to support these protocols. These adapters have to support not only the format of a message being sent, but also the behaviors implied by the protocol, which may include authentication dialogs, error handling, retry mechanisms, and others that are expected by applications. In practice, protocols come at different levels, ranging from low level technology (such as HTTP or ODBC) to higher, application-level technology provided by EDI protocols or application-level integration.

To be really successful, the adapter library coming with an integration platform has to be structured so that the higher-level protocols can be layered on top of a range of lower-level technology and that integration analysts or developers working with the higher-level messages don't have to worry at all about how the message was received.

A critical part of creating and managing any integration solution is to configure and monitor the connections to all the external systems, and the user interface for these functions is a crucial part of any product. It has to be easy to use and flexible, but it must also have a wide range of security features to control who sees what.

Data Transformation

Protocol mediation means the conversion from one technology to another. For example an invoice may arrive as a JMS (Java Message Service) message and have to be sent out over MQ Series, or records from a flat file may needs to be inserted into a database. This technology conversion has nothing to do with the structure of the invoice, which might be unchanged.

However, it is highly likely that the information in the message also needs to be changed to make it acceptable to the receiving application. This is often referred to as ***data transformation*** or ***semantic transformation*** and has to have knowledge of the two applications involved. Consider the invoice example

– if two applications have the concept of an invoice, then the structures will have a lot in common. There will probably be:

- Header information about the payer and the payee and the date
- Line items for the things that have been purchased
- Footer information about totals

But, of course, the format of each of these things will be different, so the data transformation has to know how to shuffle the information around to create the right format. As well as restructuring the information, the transformation needs to modify the actual content. The two applications may use different product codes for the same items and the transformation has to use a lookup table, or may need more sophisticated business logic to derive the values for the target system.

The ease and power of semantic transformations are key to the effectiveness of an integration platform and can range from graphical, drag-and-drop mapping tools that make things very easy to simple callout hooks that allow you to write your transformations in Java. The latter is powerful (with the right skills), but is certainly not easy and probably doesn't perform well at run time.

Routing and Orchestration

Once a request or a message has been received by the integration platform, it has to decide what to do with it. ***Routing*** normally means the integration platform can tell where to send it based on something about the message, perhaps its origin or type, and will optionally return the response from the destination back to the originator. More sophisticated routing engines will also route based on the content of the message and can send the message to multiple destinations, modifying the message as required for each destination. If the message is sent to many places, this begs the question of which response gets routed back to the originator.

Of course, the technology of the source and destination(s) may be completely different, so a message may come in as part of a flat file and be sent out as a SOAP request over HTTP. These technical details have to be transparent to the application-level person deciding where the message needs to go or the task becomes too difficult and too rigid.

Orchestration implies something more sophisticated than simple routing. Now, to process an incoming request or message, the integration platform will first invoke one or more external applications and, based on the response, decide what to do next. To service a sales request, for example, the integration platform might first call the inventory application to check availability, then the billing application to generate an invoice, then notify the shipping department to place the order, and finally circle back to the warehouse and inventory systems to automatically place a reorder request if the stocks are running low.

To really implement business processes of this type, the integration platform has to be able to be able to handle asynchronous response (perhaps a workflow approval or waiting for a physical delivery to the warehouse) and to store the state of the process securely in case the system goes down unexpectedly.

Business Process Management

Business Process Management (BPM) means the analysis of real (non-computer) processes within a business, including the flow of information between systems or departments. The goal is to measure, control, and thereby improve the way the business works. But in the context of integration platforms, it is more commonly used to mean the technical implementation and automation of a business process. Typically, it means starting with a graphical representation of the process flow and generating the code that executes calls to external systems. The level of monitoring against SLAs (service-level agreements), simulation, and other aspects of true BPM varies.

Business Rules

Typically, the overall business process is implemented by developers to call rules at certain points. The analysts are then given an easy-to-use editor that allows them to modify the rules without giving them free access to the application data. For example, a business process that implements an on-line sales process needs to calculate the shipping or delivery costs. By turning this calculation into a business rule, the process can be more responsive to changes in delivery costs and can be more agile in marketing campaigns for 'free shipping.'

Very often business rules are included as part of a BPM product or suite. The primary aim of any ***'business rules***' software is to allow non-programmers to make changes to a system and get these into use as quickly as possible. But change to running systems has to be tightly constrained or the results could be disastrous and modifications must also be strictly audited to prevent abuse.

Pure play business rules systems have sophisticated algorithms for supporting thousands of complex rules, but this isn't needed in most BPM solutions.

Enterprise Service Bus (ESB)

Some people define an ESB in terms of the technology used to implement it, with distributed Java components using JMS, but this is a limited view. The main point is that the ESB is the connection framework regardless of the technology. As time has gone by, expectations of the ESB have gone up, so now it is expected to include most of the functionality described above.

SOA

SOA (Service Oriented Architecture) is not a technology, and it isn't just an architecture. It is a methodology and a process for building applications. A large part is process and agreement between parties within the enterprise on how to build, reuse, and manage software.

The essential part is that functionality is not built for a single application; it is built as a service (not necessarily a web service) that can be reused by other applications. That doesn't mean a reusable library of modules you can include in your application but an actual running service somewhere on the network.

That simple description has many implications. Application writers must know where to find out what services exist, how to use them, and where they are running. In a small environment that might be easy, but in larger organizations a directory of some kind might be used.

Version upgrades, changing the location of a service, monitoring performance, security, and many other governance issues now become critical factors in the success of the SOA project.

In an SOA environment, the ESB is an important technology component. It used to be referred to as the "SOA backplane" to distinguish from an ordinary ESB, but now many of those extra features are being offered as part of the ESB.

COMPOSITE APPLICATIONS

Composite applications are one of the most sophisticated forms of application integration. A composite application can appear to be a new application but in fact is clever reuse of existing data and applications.

A new user interface is created but behind the scenes, instead of a distinct set of business logic, the integration platform orchestrates calls to existing applications to do most of the work with perhaps a small amount of new business logic being added to fill the gaps that aren't satisfied by existing applications.

Very often composite applications are single function apps aimed at specific groups of users. For example a self-registration kiosk in a hospital would have to check and update registration and other applications, verify insurance and might even accept credit cards for copayment with all the external processing involved in that. A simple user interface hides all this complexity from the patient and orchestrates the calls to the back-end system.

Major developments of composite applications can revolutionize a vendor's aging product line. By bringing together legacy systems behind a web 2.0 UI some vendors have leapfrogged past their competition without rewriting their long- established systems.

CONCLUSION

THERE ARE many DIFFERENT types of integration techniques, but this isn't just an accident of history. Different business situations need techniques and message exchange patterns. A strategic interoperability platform needs a full range of functionality and easy to use tools that are designed for each. The technology required is driven by the business needs first and foremost and successful integration platform focuses on business use cases and not just technology.