Choosing among JCA, JMS, and web services for EAI

Positioning alternative interface technologies for enterprise integration

Regis Coqueret
Marc Fiammante

March 01, 2003

This article discusses criteria for choosing among J2C Connector Architecture (JCA), Java Message Service (JMS), and web services implementations, depending on your existing environments, the patterns you want to implement, and the your preset requirements for loose or tight coupling.

Introduction

Organizations evolve rapidly, and they seek to meet changing business requirements while managing costs. This means that enterprises desire to structure their own applications in a way that allows easy reorganization of information systems. Major organizational changes such as mergers or the creation of subsidiaries might also introduce new variables into the information system.

Enterprises might also need to buy applications on the market or to subcontract part of their business needs, such as ledger or back-office management. There is no guarantee that such services are available on the existing technical framework.

As the complexity of information system increases, development must be simplified. This brought about interest in Enterprise Application Integration (EAI). Still, enterprises must complement EAI with business services and a flexible way to access the resulting, integrated applications.

Interface-based architectures currently address this increasing need for flexible access to business services and client independence. Interface-based architectures include such technologies as web services, J2C Connector Architecture (JCA; see Related topics for more information) and Java Message Service (JMS). They also include all of the variations of the command pattern, which isolates client code from the implementation of the business service. You can call such invocation frameworks from EAI middleware, and vice versa.
In this article we first discuss the main characteristics of each interface technology, and then the requirements that suggest one or another. After reading this article, you will understand how to position the various technologies and how to choose among them for a particular implementation.

**Web services, JCA, and JMS characteristics**

This section outlines the interface technologies of interest and details some of their characteristics.

**Web services**

Web services are an implementation of Services Oriented Architectures (SOA). A SOA has three parties: a provider, a broker, and a requester, which are loosely coupled. The provider offers a business service that represents a particular implementation, which is not directly visible to the requester. The requester learns from the broker the information structure that it has to send and receive from the provider and what protocol to use to access that service. The requester has no knowledge of the way the provider implements the business service.

Web services are defined as required business interfaces between a requester and a provider, and not as a common pipe for all business requests. There are several variables that can characterize web services, including that:

- They can be tightly coupled, and their deployment can be based on the use of invocation frameworks.
- They can perform either in a synchronous request/reply mode or in an asynchronous mode.
- They can be exposed by J2EE or non-J2EE providers.
- They might or might not offer support for transactions and security.

**JCA**

Java Connector Architecture mainly addresses the need to access the business logic of *Enterprise Information Systems* (EIS) in a tightly-coupled way. Connector architecture provides support for resource adaptation, which maps the J2EE security, transaction, and communication pooling to the corresponding EIS technology.

Initially, connectors were intended to access legacy transaction servers on mainframes in a synchronous request/reply mode, and this is how the majority of connectors work today. The standard is currently evolving toward more asynchronous and two-way connectivity.

Some user-defined variants of the connector are more sophisticated and work on a logical connection mode. As such, they can behave as invocation frameworks, selecting the appropriate physical EIS target and business operations in a manner similar to web services.

**JMS**

JMS is an asynchronous message-based interface. You can also use JMS to access business logic distributed among heterogeneous systems. Having a message-based interface enables the following functions:
Point to point and publish/subscribe mechanisms. Message-based frameworks can push information to other applications without their requesting it explicitly. The same information can be delivered to many subscribers in parallel.

Rhythm independence. JMS frameworks function in asynchronous mode but also offer the capability of simulating a synchronous request/response mode. This allows source and target systems to work simultaneously without having to wait for each other.

Guaranteed information delivery. JMS frameworks can manage the messages in transactional mode and ensure message delivery (but without any guarantee of timeliness of delivery).

Interoperability between heterogeneous frameworks. The source and target applications can operate in heterogeneous environments without having to handle problems of communication and execution related to their respective frameworks.

Making exchanges more fluid. The switch to message mode allows finer-grained information exchange.

Choosing interface technologies
The way you have already implemented business logic in your system will naturally lead you towards one of these technologies. As a first step towards making a choice, analyze your existing infrastructure. Is there an existing messaging system, or a legacy system such as CICS or IMS?

In many cases the most natural way to access mainframe EIS such as CICS or IMS is through the Java Connector Architecture. If on the other hand, you need to access .NET applications, you will probably tend towards a web services interface. In yet other cases you might use a JMS interface, which allows the exchange of messages with little constraint on the implementation language.

Use the following summary of decision points:

**You have existing Java applications or plan a new one**: use JMS or JCA.

**You need to interact with partners**: use web services for transport and connection.

**You need to cross the barrier between languages**: use JMS or web services.

Another factor in decision-making is the scope of the network: whether Internet, intranet, or extranet. This scope determines your flexibility in choosing the transport protocol. Deployment on the Internet probably demands loosely-coupled web services over HTTP. This would work with existing firewall and demilitarized zone (DMZ) infrastructure, and you can minimize the cost. JMS and JCA make more sense as Intranet or Extranet protocols. JMS is suitable for asynchronous mode or simulated synchronous mode, and JCA is suitable for tighter coupling.

**What the options have in common**
Web services are not synonymous with services offered over SOAP. You can consider any piece of code with a Web services Description Language (WSDL) description of its functionality and access...
protocols a web service. You can make any such service available over multiple transports and protocols.

You can therefore adopt a WSDL-centric approach, described and implemented by the Web services Invocation Framework (WSIF). This ties together your choices for integration regardless of the scope of the network: intranet, extranet, or Internet.

You probably seek to leave the broadest options for future expansion plans, including extension of existing enterprise systems or making connections to business partners. To facilitate this, you can describe the same business component in a single WSDL document, which you can:

- Make available on different protocols, such as SOAP and RMI-IIOP, through the notion of inbound bindings. RMI is Java remote Message Invocation. IIOP (Internet Inter-ORB protocol) is the CORBA wire protocol.
- Implement with different types of components, such as JCA or JMS-based components, through the notion of outbound bindings.

Following this approach, JMS and JCA are just a couple of the protocols that a server provider might use to implement a business component. Different network and interface technologies then affect only non-functional matters such as security, performance, response time, and availability. When the same component is available on an intranet and the Internet, the two networks will differ in the security required, the performance expected, and the availability demanded.

The WSDL-centric approach for web services gives you the ability to decouple the abstract interface from the exact protocol stack. You can implement this approach in two ways, both of which leverage WSIF:

1. With the IBM Web services Gateway (WSGW), existing WSDL descriptions are deployed and made available through SOAP channels. The inbound binding is SOAP, and the outbound binding is the existing WSDL implementation description.
2. With the IBM WebSphere Studio Application Developer Integration Edition (WSAD-IE), existing WSDL descriptions are consumed and made available through SOAP or RMI-IIOP proxies.

Interaction patterns

When architecting applications, you define patterns of interaction. These patterns generally reveal your preference for technical integration.

Event-driven, push model

Listeners that wait for events provide the standard event push model, which is particularly important to EAI. Business process automation often requires the capture of application events for propagation to other integrated applications.

Standard listeners for business services often use JMS or HTTP servers. Message Driven Beans in EJB use JMS, while web services use HTTP.
In this push model, the pushed events trigger processes. The web services community has done a lot of work to formalize the choreography of this process interaction. In particular, the web services Flow Language (WSFL) and Business Process Execution Language for Web services (BPEL4WS) standards address such event sequencing.

- In WSFL the flow engine can execute a new flow instance in response to an incoming message (marked by the spawn lifecycle operation). Its support for one-way interaction provides support for notification.
- In BPEL4WS, receive or pick activities can implicitly start business processes in response to incoming messages; pick activities can select the process to run based on one of a set of events.

**Synchronous request/reply mode**

In the enterprise, performance requirements may suggest a JCA implementation, especially when most of the business logic is currently in some existing EIS. However connectors are not available to access all systems, and in some cases adding message layers with JMS is the only solution.

Messaging is not the primary choice for request/response-type interactions. Implementing request/response with messaging middleware prevents the transactional coordination between caller and called because of the isolation implied by messaging. Also, response timeouts have to be managed by the caller's programming logic rather than a provided connector implementation.

**Asynchronous model**

All three interface technologies -- web services, JMS, and even JCA -- can work in asynchronous mode. The requests or events would be sent to the target without expecting an answer other than: "The message was delivered correctly." It is a fire-and-forget style of interaction.

Supporting this model is not a major issue in an architecture, and the techniques you use can be similar to those in other models. Usually you will pair an asynchronous model with event push support or a polling mechanism, and it is probably these details that will suggest a particular technology for implementation.

In the case of web services, asynchronous interactions are supported, although tools are not typically well-suited to this form. SOAP-based web services support not only a synchronous RPC interaction mode, but also an asynchronous message interaction mode. The basis of this is the document-oriented model, in which both requester and supplier must deal with SOAP envelope formatting and parsing.

The document-oriented model is a way of implementing true one-way communication.

**Requirements and implications of technology choices**

This next section addresses non-functional requirements that serve as decision factors in choosing an access technology for business logic.
Loose or tight coupling

Tight coupling means that there is a nailed-down, predetermined client-server or consumer-publisher relationship that is not so easy to change. In such a case you typically have for a given client only one specific server with technical interaction that is fault-sensitive, meaning that the client has to handle protocol-related errors. You can see this tight coupling at the interface definition level, or at the protocol stack level. You might be tied to a specific, abstract definition of the service or to a specific protocol stack to reach the service.

Loosely-coupled systems are often designed to work on a problem partitioned around data flow boundaries, in which workers solve only one part of a larger problem, the context of which is not necessarily known. Often such systems are inherently extensible by means of adding more workers.

You might be loosely coupled in the following ways:

- By providing a public contract that describes the relationship between the consumer/client and the producer/server, and you can safely build and integrate your application without having to let other people know about the technical details of your application (and how it changes over time). By the same token, you can use other developers' public contracts to consume their applications without needing to know the details.
- By providing a protocol stack that will deal with connectionless interactions against anonymous servers, and you can safely interact with components through fault-resilient mechanisms.
- By providing payload-agnostic communication channels, in particular through broker-based messaging systems.

Now we map the three technologies in question to tight or loose coupling.

JCA is a technique for tight coupling.

- JCA is a J2EE standard that ties the request and the connected Enterprise Information Systems using a container. The container is a coupler providing managed mode support for security, transaction scope, connection management propagation, and interaction with the target system.
- The JCA coupling interface is strictly defined through Common Client Interface.
- System contracts and container-component contracts are hidden for the application component, but they do strongly link caller and called.
- JCA does not yet address business issues such as billing and auditing. Implementing those requirements remains a problem of application business architecture.

JMS is a technique for loose coupling. As an example, it does not provide security or transactional binding with the target systems, but only with the message middleware. Messaging in general achieves loose coupling because:

- Message producers and consumers interact through the messaging transport (such as a message broker) in a point-to-point or publish/subscribe model.
• A technical header is usually provided independently of the business payload.

JMS is well-suited for:

• Integrating systems/components that participate in business events.
• Integrating slow responders.
• Integrating existing systems.

Web services target business services rather than technical connectivity. They are primarily implemented using loose technical coupling but tight coupling at the interface-definition level.

With web services the coupling is based on both an interface definition and a protocol binding.

• The contract is published in WSDL, which defines the interface, which defines the available operations.
• The protocol used to access a web service may vary. This protocol is known as the inbound binding and defines how to get to the implementation artifact.
• The way to access the implementation may vary as well. This protocol is known as the outbound binding, and defines the artifact implementing the public contract. Examples include JavaBeans, EJB, and JCA.

The style of coupling in web services offers flexibility, with two kinds of binding: static and dynamic.

• Static binding implies tight coupling: the requester uses the service implementation obtained at design time; no private or shared registry is used.
• Dynamic binding implies loose coupling: the requester discovers at runtime the implementation to be used for the service called; it might even determine at runtime the interface that should be called.

In reality, most of today’s web services use SOAP as the inbound protocol. Until class of service is supported, SOAP will rather be a loose coupling protocol. Class of service will address security, reliability, and availability.

SOAP is still the default protocol because of its ubiquity, firewall-friendliness, and other advantages. Also, SOAP is where web services security specifications are currently being developed; so in practice, defining a standard security approach implies using SOAP.

Some other auxiliary business functions, such as billing and auditing, are now available for web services.

**Portability and interoperability**

Web services impose no programming language or operating system limitations between caller and called. In the near future some SOAP interoperability issues are likely to be addressed, such as differences between the Apache implementation and others. After this is resolved, all platforms supporting web services will interoperate (including .NET and J2EE platforms). But even then, web
service client or server implementation code will not be portable between vendors. Implementation
code written for .NET will, of course, not run under J2EE.

JMS and JCA only address the Java world. With JCA the portability is achieved on the Java client
code side, and interoperability is limited to a specific target. JMS requires Java environments on
both sides to interoperate at a technical level, but the message payload is agnostic, and using
JAXM web services, SOAP payloads can be carried on JMS.

**Transactional support**

JCA does not always support end-to-end transactional support to an EIS. As mentioned earlier,
the connector provider implements a container that propagates the transactional context to the
target systems. This is done through the transaction management system contract, in which the
container acts as a transaction manager controlling the EIS as a resource manager. This is based
on the XA standard (see [Related topics](#) for more information).

Web services do not currently support transactions. Standards bodies are working on ways to
provide transactional support on a loosely-coupled model with the WS-Coordination and WS-
transaction standards. In the future both a loosely-coupled model using compensation and a tightly-
coupled model using an XA-like transactional model will be available.

In the messaging model, once the message has been delivered to the queue, the client has no
control on its propagation. So JMS can only support transactions to the queue entry point and not
to the target application.

**Reliability**

Currently the only assured delivery is through JMS (although there is no guarantee on message
delay). For the future, IBM, Microsoft, BEA, and TIBCO jointly propose standard mechanisms
for exchanging secure, reliable messages in a web services environment. You can access these
standards on the web Services organization site (see [Related topics](#)). The reliability of the JCA
connection will always be connector-specific, and JCA itself does not imply reliability.

**Security**

Security is the least standardized domain. In JMS the specification explicitly states that security
is left to the JMS provider implementation, which is compatible with the rest of J2EE security.
Fortunately, IBM also adheres to this standard. With JCA the support of the security will
depend on the capability of the connector container and targeted enterprise information system
implementation.

HTTPS support can provide some web services security at the transport layer. However, after the
web server deciphers and authenticates the request, it is vulnerable, and only some web services
implementations using the [SOAP Security Extensions](#) (SOAP-SEC) support the currently-
available security (see [Related topics](#) for more information on SOAP-SEC). Standardization work
is being undertaken in the WS-Security standard that should enable a fully interoperable end-to-
end security model in the future.
Conclusion

As you can see, the choice among web services, JMS, and JCA implementations for integration is based on multiple criteria. By mapping to the requirements for particular solutions, and by prioritizing these requirements, architects can choose the right implementation for their particular case.

The following table summarizes what we have said so far, and lays out decision criteria:

<table>
<thead>
<tr>
<th></th>
<th>Web services</th>
<th>JMS</th>
<th>JCA</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface coupling (abstract service definition)</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
<td>Dynamic interface discovery and request construction is possible</td>
</tr>
<tr>
<td>Technical coupling (protocol stack)</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>With WSIF, client is not bound to a client library for a particular protocol implementation</td>
</tr>
<tr>
<td>Portability</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>Multi-language</td>
</tr>
<tr>
<td>Reliability</td>
<td>HTTP-R binding for SOAP</td>
<td>YES</td>
<td>Specific</td>
<td></td>
</tr>
<tr>
<td>Transactional Support</td>
<td>Future</td>
<td>YES</td>
<td>XA model</td>
<td>WS-Coordination &amp; WS-Transaction Compensation and XA models</td>
</tr>
<tr>
<td>Security</td>
<td>Limited to SOAP</td>
<td>Not part of the standard, hence vendor-specific</td>
<td>Integration between EIS and J2EE</td>
<td>SOAP-SEC, superseded by WS-Security</td>
</tr>
<tr>
<td>Synchronous mode</td>
<td>YES</td>
<td>Do it yourself</td>
<td>YES</td>
<td>Major use</td>
</tr>
<tr>
<td>Asynchronous mode</td>
<td>YES</td>
<td>YES</td>
<td>Future</td>
<td>Document oriented interface</td>
</tr>
<tr>
<td>Event-driven, push mode</td>
<td>YES</td>
<td>YES</td>
<td>Future</td>
<td>Document oriented interface or flow support (WSFL, BPEL4WS)</td>
</tr>
</tbody>
</table>
Related topics

- The JMS specification is available at java.sun.com.
- Find a complete description of the JCA standard at Java 2 Connector Architecture (JCA).
- A good entry point to WSIF is "Applying the web services Invocation Framework," by Paul Fremantle (developerWorks, June 2002).
- Review the Business Process Execution Language for Web services, Version 1.0 (BPEL4WS) specification.
- Learn more about Apache Axis.
- Although superseded, current SOAP Security Extensions are best described at SOAP Security Extensions: Digital Signature.

© Copyright IBM Corporation 2003
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)