Technical White Paper

Web Services & JMS

Standards based message delivery across the web

spiritsoft
go beyond jms

CONTENTS

Abstract

Web Services. What are they? What can they do? We have all been bombarded by the buzzwords like Java™ , JMS, SOAP, J2EE, CORBA and RMI to name but a few. The idea was to make distributed systems easier to build, easier to deploy and easier to maintain. The advent of the World Wide Web, the Internet, and the drive towards Web Services have all contributed to the pressure to deliver enterprise systems in a widely distributed environment. In this whitepaper we explain what Web Services are, what they can do and, equally important, what they cannot do. We show how the SpiritArchitecture can be used to provide open, standards-based multi-plug communication through open JMS, generic connectivity through JCA, performance gains through JCACHE intelligent caching, and business transaction management through dynamic event management - all of which alleviate many of the problems in delivering complex Web Services.

Introduction

Building large-scale distributed systems is not an easy task. In a distributed system, building to scale while remaining flexible requires an understanding of the whole system and not just of individual components; this is in part why the promises of the Common Object Request Broker Architecture (CORBA) and the Internet Interoperable Object Protocol (IIOP) failed to deliver. Few CORBA/IIOP-based solutions dealt adequately with scalability in a loosely-coupled environment and as a result, many such systems ended up being crippled by large server-based bottlenecks. Even fewer delivered the flexibility required in distributed systems, in particular flexibility at the level of business transactions - the flexible, choreographed exchange of business events.

Earlier interchange standards such as EDI (electronic data interchange) have only achieved limited penetration in Fortune 500 companies because the formats are ambiguous or overloaded, and because - unlike ebXML for example - they don't address transaction 'choreography' - the way multiple interactions are sequenced to achieve a complete and successful business transaction.

The advent of the Java™ environment and the rise of XML (extensible markup language) present a fundamentally different landscape from the one that was prevalent at the time of CORBA/IIOP. A key difference is the ability to build portable applications using Java and the ability to seamlessly exchange information between applications using XML. More recently, the rise of Enterprise Java Beans (EJB) and Remote Method Invocation (RMI) fired everyone with enthusiasm but little was learned from the experiences of the past. Unfortunately, EJB and RMI are little more than CORBA/IIOP in Java clothing.

Dissatisfaction with both the CORBA/IIOP and EJB/RMI accounts for the popularity of Message Oriented Middleware (MoM) and the Java Message Service API (JMS) and the rise of "fire-and-forget" one way messaging (e.g. UDP) rather than request/response as a more scalable communications pattern for delivering distributed systems. Couple that with XML as a core technology for encoding information and we move rapidly towards Web Services, the current movement to be adopted in the world of distributed systems and Java™ 2 Platform, Enterprise Edition (J2EE or Enterprise Java).

What is new about Web Services and J2EE to warrant all the hype? This paper provides a common understanding as to what Web Services are, what they are for, and looks at some of the important challenges that they do not adequately address. In doing so we will look more closely at the benefits of standards based open multi-plug communication, multi-channel delivery and dynamic event flexibility and the business critical role of this standards based infrastructure in the construction, deployment and subsequent maintenance of Web Services.

What are Web Services?

Web Services are a new generation of web application components. They move away from the monolithic web applications of the past by being self-describing and modular. As a result they can be published, located and invoked across the Internet providing their particular function to anyone that wants to use it. A traditional web application might involve many Web Services in doing their jobs and these might be internal to the application owner's domain or might involve disparate web-based functions scattered about on the Internet.

A web service can perform a range of functions from simple requests to complicated business processes. A simple example might be to provide foreign exchange conversion and a more complex example might be to provide a credit card transaction or a complex set of financial trades. What differentiates Web Services from traditional applications we see on the web today is that, once deployed, other applications-not just browsers-can locate various relevant services, understand how to communicate with them and then ask them to do things. Web Services can be seen as a natural evolution of component-based architecture to the worldwide Web.

Web Services work by explicitly declaring what they do, how they wish to communicate and what vocabulary they understand. This is encoded and held in a global registry that other Web Services can interrogate. At first glance this isn't all that different from CORBA, but what makes it more compelling is the introduction of both Java and XML to the situation. XML is one of the fundamental building blocks for Web Services. It is used to encode how Web Services communicate, what services they offer and the methods that they execute. If XML is the key to descriptive semantics for Web Services then Java technology is the engine upon which many services will be built and deployed.

In essence, Web Services address the problems in bridging the semantic gap that occurs when two or more businesses wish to transact by providing a set of specifications for multi-protocol business management based on the Simple Object Access Protocol (SOAP), Web Service Description Language (WSDL) and Universal Description Discovery and Integration (UDDI) repositories that leverage complementary initiatives such as Electronic Business XML (ebXML).

W3C has a proposed service stack that documents the likely architecture of any web service and breaks it down into a wire stack, a description stack and a discovery stack in which SOAP, WSDL and UDDI each plays its part.

SOAP plays the role of wrapping data and functions as messages with a Quality of Service (QoS) indicator to fit on an appropriate communications protocol.

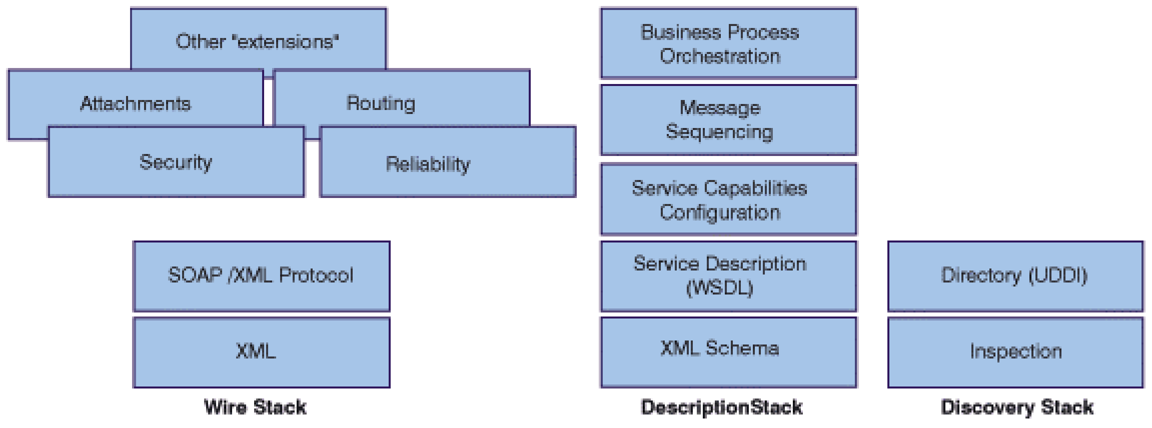

Figure 1: An overview of the web service architecture stack [from, "A summary of the W3C Web Services Workshop"]

WSDL allows us to ask questions about how we want to talk and what vocabulary we wish to use. UDDI allows us to tell the world what we have, and UDDI registries (or directories) from Microsoft, IBM, Ariba already exist for this purpose. ebXML - although it overlaps with UDDI in its provision of registries and discovery service - provides the higher-level message sequencing and generic business process coordination upon which business transactions are made.

Service discovery

The challenge of service discovery is at the very core of what the eCommerce revolution is all about. There has never been a common way to easily get information about what standards different companies support, and there is no single point of access to all markets of opportunity. As a result it has not been possible to easily connect with all the trading partners in any one domain. What has been missing is the ability to access information about trading partners and how to integrate with them. This fundamental challenge is what has limited the delivery of B2B collaboration on the web, making it harder for buyers to get a return on their eCommerce investment and for all B2B participants to add trading partners and services at a cost low enough to make it attractive.

The generic solution to this is the creation of a service registry architecture that presents a standard way to query and maintain information needed for the interoperation of businesses so that the information can be shared and globally distributed.

The UDDI project is an initiative to create a global, platform-independent, open framework to enable businesses to discover each other, define how they interact over the Internet, and share information in a global registry and is the favored Web Services solution to service discovery.

UDDI registries are one of the fundamental building blocks that will enable businesses to interconnect at low cost easily and quickly in order to transact using their preferred applications.

The core component of the UDDI specification is the UDDI business registration. This is an XML file used to describe a business entity and its Web Services. In this sense it is a more semantic Universal Resource Locator (url). The UDDI specification is based on the Simple Object Access Protocol (SOAP), which is used to define the protocol for service registration, enabling anyone who can be SOAP conformant to find or update an existing service or register a new one.

Service description

Service description requires the capturing of the interface, the protocol bindings and the deployment details of services. This is what the Web Services Description Language (WSDL) is all about. It is an XML schema that complements UDDI by providing a uniform way of describing the abstract interface of a service and the protocol binding to which they adhere. A typical service might offer access to "widget buying" but only offer it through a SOAP message embedded in an HTML request sent over HTTPS, or a SOAP message sent through a JMS connection on a particular port doing HTTP tunnelling. UDDI will incorporate WSDL as the means by which services are encoded into UDDI compliant repositories.

Service operation

Service operation is all about the ability to interact with a service. Interaction is the ability to execute functions against a service by providing data as parameters to functions and receiving corresponding data for return values. This is the basis of the Simple Object Access Protocol (SOAP). SOAP provides a more atomic mechanism for this, based on XML. While this is analogous to CORBA and IIOP, it also provides an easily understood (by humans and processes) on-the-wire format for such interaction. Being widely accepted by both the Windows and non-Windows communities it has the added advantage of bringing together the web community as a whole and so raises the debate above technologies such as Java.

SOAP however is not enough. It is used by WSDL (WSDL provides binding to SOAP 1.1 endpoints) and UDDI (SOAP is used to describe the interaction protocol with the UDDI repository) as a way of encoding functional access and data passing. It is not enough because it doesn't deal with the sequence of interactions that makes up a business transaction. This is the remit of protocols such as ebXML, Biztalk and Rosetta (each with their own conversation or "partner interchange process" definitions) for generic business exchange and fpML, fixML, etc., for financial services. Earlier EDI standards like X12 on the other hand contained a number of ambiguities and did not address the lifecycle of a long-running business interaction.

What are the challenges with Web Services and why?

While the fundamental principles that underpin Web Services are sound, and the ability to re-use components is certainly an attractive one, they are not new. CORBA and now J2EE in the guise of JSP and EJB technology are examples of similar service oriented architectures to those proposed by the Web Services initiative. What has changed that makes Web Services more interesting is that the "on-the-wire" protocol and the service description are XML derivatives. By adopting a document / message based interaction - rather than call and return - Web Services have begun to blur the line between synchronous and asynchronous interactions.

Let us separate Web Services into a number of categories. First, there are those that need to perform some simple function and return some simple values. The business transaction they enact is a simple request-response pairing. Secondly, there are those that need to perform some more complicated business transaction with the enacted process being a flow of requests from one web service to another, and with the response being more complicated than a single response. These sorts of Web Services may also require a degree of customization of the processes that they enact based on the contextual identity (e.g. who you are, what your interests are, what your buying habits are) of the requestor. Finally, the third example would build on the second, adding the ability of the web service itself to proactively distribute events which encapsulate change and new information to interested parties. Figure 2 (see sidebar 1 Strawberry Growers) provides an architectural overview of what such a service might look like.

An example of a simple web service is one that is used to calculate exchange rates for tourists. Such a service might be the backend to a web kiosk and at the same time be available to some smaller tier 3 banks. In this way the service is shared across a range of users.

An example of a web service that requires a more complex business process is to provide brokerage facilities between buyers and growers of strawberries. In this service the buyers talk to the broker to place an order. The broker sources the order from any stock available and places orders with growers. The complex business process requires handling of an event stream that results from the initial order placement from the buyer. From a business perspective the order is seen as outstanding until the order is fulfilled, the order is aborted or the order times out. An order might be partially or wholly fulfilled. The web service needs to ensure that the flow of events that make up the business transaction is managed and is flexible. There is a high likelihood that no two buyers and no two sellers are alike; each will want to talk to the broker in his own terms. This is not an arbitrary requirement. It is fundamental to the protection of the value chain as they see it.

The final example of a complex web service is one that has a high fan-out feedback loop. If we build on the strawberry grower example, adding a requirement to distribute price changes or availability changes to buyers, then we change the web service that the broker provides to one that is now a true marketplace with dynamic pricing. Each transaction can have an impact on the current price and availability. External events might change the price too. Imagine what happens if one area, packed with strawberry growers, suffers from a torrential downpour that results in half of their strawberries being wiped out. This also has an impact on price and availability.

The last two examples have very specific requirements. The complex web service requires flexibility in the way in which events are handled and mapped to business transactions, the aim being to enable the buyer, seller and broker to perform business transactions according to their preferences. In this way they can monitor and account for the business transaction as they require, and at the same time they should be able to communicate using their preferred mechanisms (e.g. http, https, leased line with JMS compliant messaging solution or even straight sockets).

Complex Web Services require a high fan-out feedback loop to publish information to interested parties in much the same way as financial services do today for pricing information. Interest is declarative and receipt is decoupled from the source. This provides near real-time feedback in support of decisions. Conceivably a decision can be initiated by a human or a reactive rule that decides what to do when a price rises above or falls below a threshold. Thus the web service becomes a reactive and not just a static service. Reactivity may apply to a wide range of potential usages and enable a more active approach to business decisions. To do this, a wide event plane is needed so that the events that underpin business transactions or informational notification traditionally associated with price and news feeds can be utilized for business transaction monitoring and management, intelligent routing of information and reactive rule support for real-time decision making.

The challenge with Web Services technology today is the lack of explicit support for real-time business management and business flexibility. The technologies that underpin Web Services - SOAP, UDDI, WDSL, and ebXML - do not by themselves address this. What is required is technology that augments and complements existing Web Services architectures that can provide flexibility, caching and loosely coupled high fan-out (1-to-many) notification all of which are potential solutions to these problems. The aim is to manage complex, multi-partner processes where some response times may be measured in hours or days rather than in sub-seconds. To handle long-running business transactions we have learned that we have to stretch or even break the synchronous call/return interface and offer more loosely coupled solutions.

How do we address these challenges?

What is needed is an intelligent infrastructure to support these requirements. Intelligence is needed to leverage existing network protocols whilst routing and caching information. Intelligence is also needed to provide business flexibility by managing events in a distributed environment. And an open communications mechanism is needed to harmonize connectivity and to take advantage of existing communications technology.

The SpiritArchitecture and Web Services

The SpiritArchitecture is a suite of tools that provides standards-based open multi-plug communication, multi-channel delivery and dynamic event flexibility. The SpiritArchitecture has grown up in the Internet age with the experience of people who have delivered mission critical systems in distributed environments akin to those found in financial services, investment banking and enterprise workflow solutions. All of this experience has been brought to bear to produce a collection of well-defined tools that provide business benefit in and of themselves as well as being more than just the sum of their parts.

The SpiritArchitecture comprises SpiritWave, for open JMS compliant messaging; SpiritLite, for web, wireless and real-time delivery; SpiritBroker, for connectivity; transformation, filtering and routing; and multi communication technology support; SpiritCache, for open standards-based caching of information where you need it; and SpiritIntellect, for dynamic business event management.

All of the tools are XML aware and are based on the Java platform. They complement Web Services by offering J2EE compliant solutions that work seamlessly to provide messaging, connectivity, routing and caching wherever you need it through SpiritWave, SpiritLite, SpiritBroker and SpiritCache, as well as dynamic business event flexibility, through SpiritIntellect, to manage business processes and business transactions that can grow and evolve as businesses change.

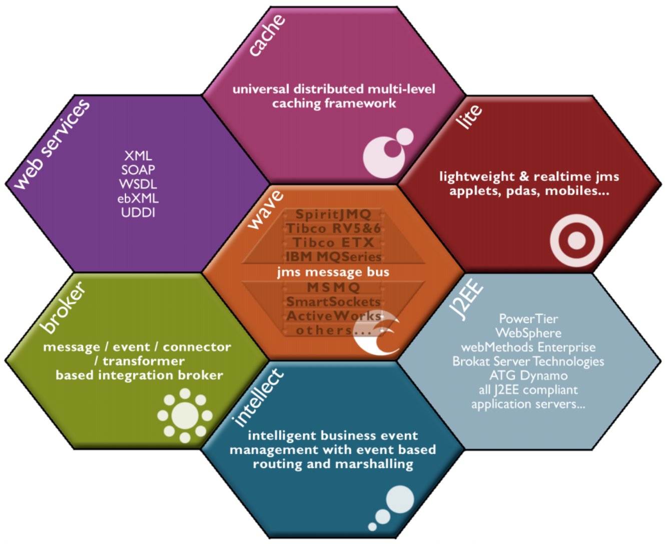
cache
universal distributed multi-level caching framework
web services
XML
SOAP
WSDL
ebXML
UDDI
lite
lightweight & realtime jms applets, pdas, mobiles...
wave
SpiritJMQ
Tibco RV5&6
Tibco ETX
IBM MQSeries
jms message bus
MSMQ
SmartSockets
ActiveWorks
others...
broker
message / event / connector / transformer based integration broker
J2EE
PowerTier
WebSphere
webMethods Enterprise
Brokat Server Technologies
ATG Dynamo
all J2EE compliant application servers...
intellect
intelligent business event management with event based routing and marshalling

Figure 3: The SpiritArchitecture

Let's take a closer look at out Strawberry grower and buyer example. Figure 4 (see sidebar 2 Strawberry Growers Solution)

To review, there are three stakeholders in this process. The Buyer who sends order requests to the Broker, the Seller who offers stock to the Broker and the Broker who mediates between the buyer and seller. In this example all of them are using SOAP as their on-the-wire protocol.

Half of the Sellers use a JSP front end and an http connection to talk to the Broker and the other half use a JMS connection with http tunnelling. The connections between the Broker and the Buyers also vary, with 25% using a JAXM interface, half using JMS and 25% using https.

Three quarters of the Sellers and the Buyers have adopted ebXML for managing their business transactions and do nothing more than follow the standard for interchange. However the bigger growers, about 25% of the total, have legacy systems that expect X12 or UN/EDIFACT messages, and so need a component that offers ebXML/X12 transformation. In particular the way that they journal and account for stock and shipments is quite different from ebXML's process and also different from each other's.

A similar situation arises for the buyers. The big corporate buyers have specialized features that they need to exploit in the way they conduct business. In the case of one of the leading buyers, a large supermarket, they place an order but the order must be sourced from at least 5 different growers.

The broker needs to manage such disparate needs to stay in business and yet provide a low cost mechanism to manage the simple cases as well as the complex cases.

The story is further complicated when the large supermarket and two large growers move to wireless and embedded notification. The supermarket automatically orders more strawberries as existing stock figures are updated. Growers are able to walk around the fields assessing the quality and quantity of their crop rather than waiting until they get back to the warehouse. This information feeds into the broker as availability, quality and price statistics in real-time.
As the Broker expands the client base, both growers and sellers, it becomes more important to cache

information for selected clients as well as at the Broker's IT center.

The architecture, shown in Figure 4, (see sidebar 2 Strawbery Growers Solution) represents a classic Web Services solution that uses the normal technology and leverages the SpiritArchitecture providing the ability to communicate with anything over anything and doing it from anywhere. It provides the caching where it is needed and the dynamic business event flexibility required to deliver the business transactions and reflect the customer's exact requirements in the way in which they want to do business.

In this way the SpiritArchitecture provides the strawberry buyers, growers and brokers with real-time business event management, intelligent caching to ensure information is accurate and timely, and a unified communications infrastructure enabling all parties to communicate and transact on their own terms as well as being able to monitor for anomalies in related information which might affect decision making.

The Infrastructure Answer

The mantra at the start of this whitepaper was to provide standards-based open multi-plug transportation, multi-channel delivery and dynamic event flexibility. Taking a holistic approach to these requirements enables us to look for a more unified and harmonious solution.

If tools can be built that unify the event plane, provide open caching, open loosely coupled communication and that can flexibly monitor and manage business transactions, we will have succeeded in providing the open standards based multi-plug communication, generic connectivity, intelligent caching and business transaction management that we alluded to at the start of this whitepaper. If we can provide multi-plug communications and multi-channel delivery, and utilize them as a transport for a decoupled caching solution and for dynamic event management then we have an architecture that is harmonious and is more than the sum of its parts. We should be able to receive information from any source and leverage any investments we have made with existing communications technology. We should be able to cache information wherever we need to and from whatever source. We should be able to monitor and manage business transactions using dynamic event management regardless of the origin of the events.

Such an architecture would need to be complementary to Web Services technology by being deployable within an EJB application server or within a JSP or web server. It should be able to play a dynamic and flexible role within the Web Services solution by providing intelligent routers for requests and responses (SOAP servers). It should be able to provide the business monitoring and management of SOAP data streams that underpin business work flow in technologies such as ebXML, fpML and fixML.

About SpiritSoft

SpiritSoft develops open-standard enterprise messaging-based technologies and tools that enable dynamic business interactions across diverse applications and devices. The company's SpiritArchitecture is the only complete integration platform that meets the challenges of building, deploying and managing distributed systems. SpiritSoft's open, standards-based approach leverages Java™ Message Service (JMS) and XML technologies to enable users to seamlessly integrate with legacy technologies and any proprietary middleware. SpiritSoft's technology delivers platform-independent multi-plug messaging, universal caching, multi-channel delivery and dynamic event management, which ensures that the right information is delivered to the right place at the right time.

*SpiritSoft is privately held and based in Boston, Mass., with offices in New York and its European HQ in London, UK. Founded in 1997, SpiritSoft is funded in part by Reuters Greenhouse Fund and Catalyst Fund. Key customers include E*Trade, Instinet, LogicWorx, Market Data Corporation, Persistence, Philips, Prebon Yamane, Reuters, Sungard, Tullet & Tokyo Liberty and The Vantra Group.*

Sidebar 1 - Strawberry Growers

Grower

External interaction is based on ebXML with few variants but enough to cause a headache in management of order flow from outside.This is because the Order Management system is fairly old and uses old EDI techniques which didn't have flow defined.

Broker

The Settlement system is bespoke as is the Order Matching application because each client of the Broker has different order placement and order settlement rules that must be followed. The Broker spent a lot of money on a data driven solution that utilised a database to enter rules. Occasionally code must be added to ensure rules are followed correctly.

The Broker is responsible for sending external SOAP requests to the argiculteral UDDI repository to locate growers and try to do business with them. Once located and able to communicate with them the Broker has to "poll" them periodically to get price and availablility information. This causes a lot of additional processing in the system and limits the growth of the Broker.

Buyer

External interaction is based on either ebXML with many variants. This causes Brokers and Growers a headache as they have to confirm to the Buyers protocol as the dominant player. The Buyers also use mobile devices (PDAs) to augment automantic inventory control and this needs to be fed into the system at the point of entry.

Other

Most interaction takes place through browser based technology. These use a Web server that is JSP compliant and acts as a SOAP server, routing requests to the correct application. External access to and from the applications and browser is through a firewall and is based on http.

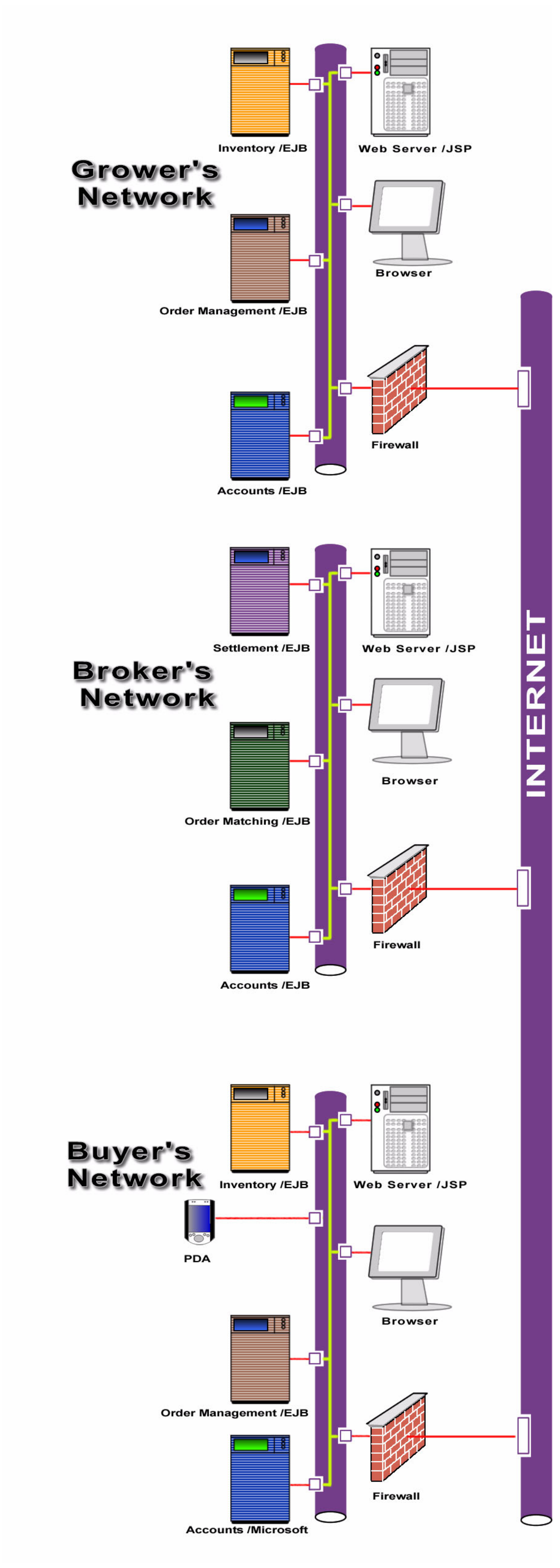

Figure 2: Example Of Webservice - Strawberry Grower

Sidebar 2 - Strawberry Growers Solution

SpiritLite is used to communicate with the PDA using decoupled messaging over http.

SpiritWave is used to communicate as the principle communication interface over legacy messaging at the buyers and over TCP/IP at the growers and broker. It enables the growers to publish changes in availability and price. It enables the growers to offer strawberries using asynchronous messaging and wait for the electronic confirmations separately. It enables the broker to receive information in real-time without the need to "poll" and this in turn leads to a lower cost of ownership for the brokers total IT commitment. The buyer can now publish order requests and wait for the confirmations asychronously allowing the total IT resource to be more effectively utilised.

SpiritBroker is used as a SOAP server routing SOAP messages to the appropriate handlers. This allows all of the players to add components to their IT system without needing to change any of the existign systems. It also plays the role of real-time message transformation and routing between legacy systems and their proprietary formats and other systems. In this way it acts as a neutralising gateway to legacy applications and technologies.

SpiritIntellect provides all of the players with the ability to manage the flow of messages that make up orders. and do so in a way that is personalisable for each player. In effect it mediates between the different business protocols in operation and enables these to change on demand as required. SpiritIntellect also plays a role for the Broker and Buyer allowing them to define alerts for price changes or news on changing weather which may affect their buying habits.

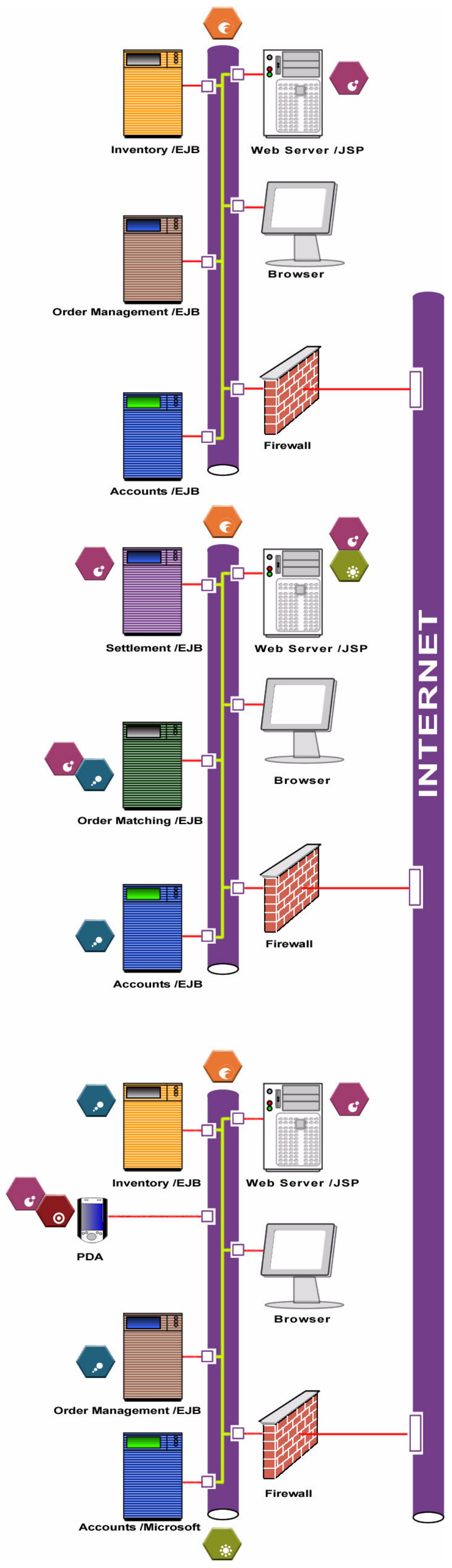

Figure 4 - SpiritSoft Architecture - Strawberry Grower

spiritsoft
go beyond jms

www.spiritsoft.com
Americas Headquarters 100 Medway Road, Suite 203, Milford, MA 01751, USA Tel: 508 473 3227 Fax: 508 473 3672
European Headquarters 10 EastCheap, London EC3M 1AJ, UK Tel: +44 20 7398 5900 Fax: +44 20 7398 5901
info@spiritsoft.com