

A brief history of web services

With many technologies receiving industry cynicism very soon after development, one that has been allowed to grow, mature and maintain its golden child positioning, even through turbulent times, is web services. Research company Radicati expects sales to reach \$6.2 billion in 2008 from \$950 million this year and sees Europe accounting for 39 per cent of all web service deployments this year



Keith Swenson,
Chief Architect,
Fujitsu Software

This industry phenomenon has not been an overnight success – developments have been much slower than expected and six years on, are we there yet? Keith Swenson, chief architect at Fujitsu Software and a pioneer of web services, shares his thoughts on how the market became what it is today, the hurdles overcome and what the future holds.

It seems almost impossible to talk about recent software system technology without the subject of web services coming up. Discussions centre on web service APIs, WSDL interfaces, choreography of those web services with BPEL and various options for transport reliability, security, transactions, context, properties and lifecycles, etc.

It is hard to imagine that there are enterprise products without a web service API today. Much of this is far more hype than help. The extent of the current market hype can be best characterised by noting that virtually all products with any process orientation have announced that they support BPEL either now or in the next version, while the BPEL specification itself has not been finalised and is as much as a year from being accepted as a standard.

How did today's industry become so obsessed with the subject of web services? Like many similar fads that have occurred in history, it did not happen as it seems. The roots of web services go back as far as the 1980s with the emergence of the local area network and the introduction of the client-server model for applications. Whilst your average person on the street had yet to even hear of the Internet, it was already in widespread use in universities and some businesses. People were getting together to discuss how to build hypertext, and what form it might take. The idea of widespread

integration was increasingly being discussed, but it was unclear how to make it a possibility.

The dawn of the '90s brought two distinct steps forward into the world. The first was the development of the CORBA standard in the OMG, an effort to provide relatively fine-grained objects to be connected without regard to location or operating environment. It was a standard to fuel an economy of separate software vendors to be able to participate in supplying reusable components of a large distributed system. The unforeseen problem, and ultimately the downfall of the effort, was that they were standardising call interfaces. The way those calls were communicated around the system could be implemented differently by different vendors, hence there was very little run time interoperability between different implementations. Later IIOP was introduced, but it was too late to save this sinking ship.

The second development was the invention of the URL to stitch resources from different systems together into a web. This was to be a major revolution, but the effect would not be noticed until many years later. The first real browser, Mosaic, was in its infancy in 1992 and widespread adoption did not happen until 1996. The vision of the browser had been around since the late 1960s, but the URL made the browser a reality.

As the URL is a convenient compact bundle of information that tells the browser exactly how to go out and fetch a particular piece of information, the real success came through HTML. It could simply embed the reference to remote information without having to understand anything about how to retrieve the information, without having to understand the meaning of the information within the URL.

HTML roots go back to SGML, which was

in use throughout the 1980s as a platform independent way to encode document markup. Besides being platform independent, the key capability that SGML (and its specialisation HTML) brings to the hypertext world is the ability to extend the expressive power by adding new tags into the mix of old tags. A browser can render all the tags it understands and ignore the tags that it does not understand. While other platform independent ways to describe a document were available at that time, it was this essential flexibility that allowed for the decoupling of the client browser and the web server.

When the browser requests a page, the server can easily serve it up without being too concerned about the version of browser doing the requesting. An older browser will render the page as best it can, ignoring the newer markup that it does not understand. This simplifies the task of the server, allowing hundreds of different versions of browsers to be easily served simultaneously. The first principle of large-scale integration is that change is continuous and uncontrollable, so you must be able to cope with many different versions of a given implementation at the same time.

XML was introduced in 1996 because HTML was not perfectly flexible, because in order to parse it, the browser needs to know a little bit about the meaning of the tags, which is acceptable for documents, but arbitrary data structures might change radically over a short time. There were any numbers of platform independent data formats at that time, however it was the extensible document structure which gave XML its uniqueness.

New tags can be added inside of previously defined tags. There are some rules about how this is accomplished. The meaning of an

existing tag must not be changed to be different from what it had been. A new tag or attribute can be added that extends the meaning, but if the message was received by an agent that does not understand the new tag, the original meaning must still stand. So new tags must be ignorable and a client must ignore new tags that it does not understand. This will allow a client of version 2.1 to talk to a client of version 2.2 without any special negotiation. The 2.1 client sends messages which all the 2.2 client will understand because the meaning of the tags never change. The 2.2 client sends messages back which have extra tags in it, which may be

still lacked widespread interoperability. Equally obvious at that time was the success of the web at weaving information together.

The preliminary specification was presented by a collection of 30 vendors to the IETF at the beginning of 1998. The IETF received it in a less than enthusiastic manner, believing for the most part that SWAP added nothing of significance to the existing IETF protocols. A quintessential question from the audience was "Why would I want to use XML and HTTP to accomplish what I can already do today with RPC?" While web services may seem obvious today, it is important to remember that at that

it actually defines only a standard message format. It defines the envelope tag, the header tag, the body tag, as well as some additional attributes and rules about how to compose a message.

SOAP has become an essential part of virtually all approaches to web services. Even so, you might think that after six years we would have been able to precisely define what information should go in the header and what information in the body; standards groups today still spend a surprising amount of time discussing things like whether the invoked 'operation' should be named in the header or in the body. Many people see the obvious parallels between a SOAP message and an email message and use email conventions to guide them, but SOAP is not like email in many ways, so one might ask if this is a valid basis for decision. These sorts of things will ultimately be set by convention, but that convention is not completely formed.

SOAP defines the message format, how then do you send it? The answer, peculiarly enough, is 'any way you want to'. With it being easy to construct and deliver a SOAP message, vendors jumped right in and came up with their own independent ways to do this that worked perfectly well. However, with SOAP focused on being transport independent, 'Any way you want to' is great for getting things up and going quickly, but a plethora of different transport mechanisms is not what you need in a marketplace of products that are supposed to fit together. There has been a huge focus on consolidating and standardising the approach

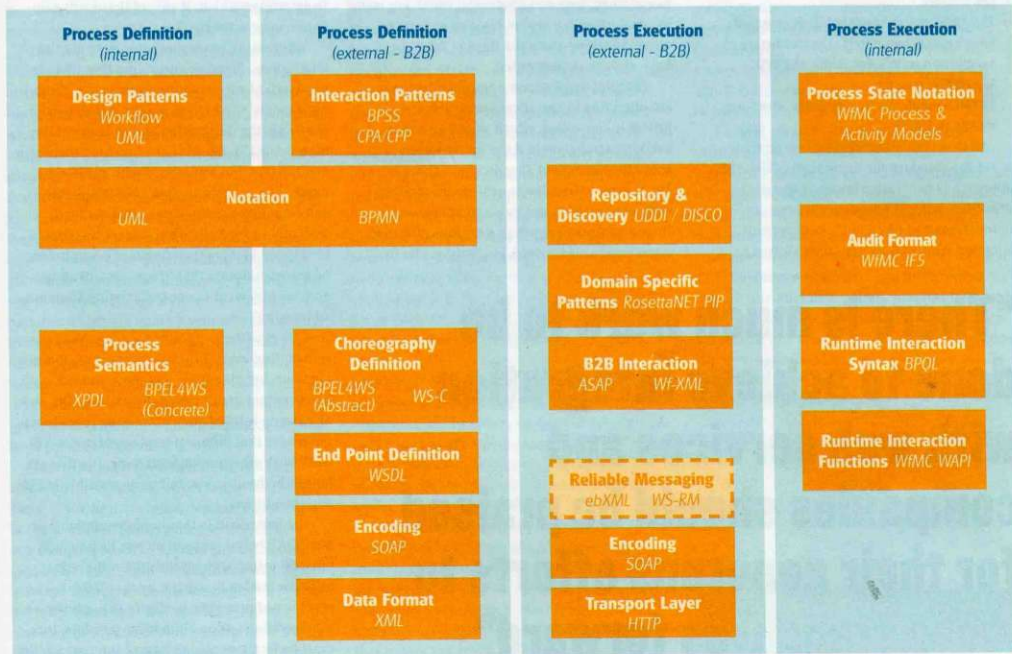
"SOAP has become an essential part of virtually all approaches to web services"

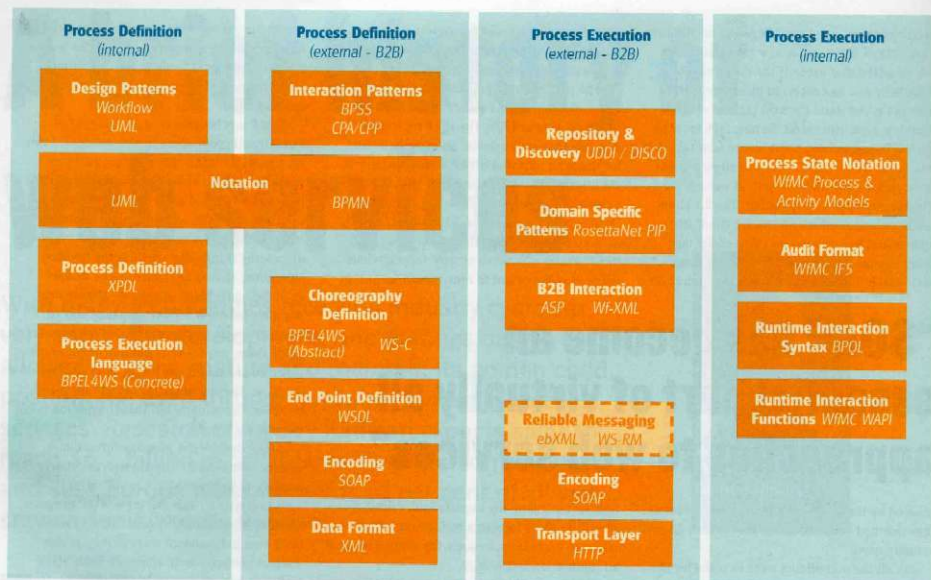
ignored by the 2.1 client and only the subset of tags that had been around at the version 2.1 are actually used.

So, all the ingredients were in place for the first web service proposal - Simple Workflow Access Protocol (SWAP). SWAP was devised by a group who had been working on a proposal hosted on CORBA, who realised that the same results could be achieved using XML over an Internet protocol such as HTTP. CORBA by that time had become very heavy and complex and

time, even among the experts in the field, it was very difficult to foresee a technology which has now, six years later, become the central focus of all integration technology.

SWAP was a bit too ambitious for its time and a few months later Simple Object Access Protocol (SOAP) was introduced which focused, quite appropriately, on the core issue of how to get an XML message from one point to another. Because of its name, you might get the impression that SOAP is a protocol, but





to reliable messaging in recent years, but today there are still three entirely different transport "draft standards" which are viable today:

- AS2 (Applicability Statement 2), the draft specification in the IETF standard by which applications communicate EDI/XML data over the Internet
- The "WS" group of related draft standards which sports two different contradictory specifications, WS-Reliability and WS-ReliableMessaging
- ebXML reliable messaging drafts which are mostly in OASIS

All three are all struggling for the right to set the standard for the industry. The AS2 approach is generally viewed as the most primitive, but it is based on very mature foundations and has a fairly widespread adoption. The "WS" is generally viewed as

the most technically sophisticated. This group has done the most to promote and hype the possibilities of web services. But this group suffers because there seem to be two or more specifications offered for each capability. There is a strong political element to these negotiations between proposals, which are made by vendors with a vested interest in getting others to follow along the lines of what they have already implemented.

The supposedly open proposals themselves are often tied to specifications which are privately controlled, which would give that vendor a stranglehold if the market were to adopt the dependent specification. There is much work to be done to achieve integration with web services and those companies should be praised for their generous efforts to move standards forward, but fortunes lie in

the balance, and the public should look very suspiciously and critically at the proposal before accepting them. ebXML lies somewhere between these two in being an evolutionary approach from real-world experience, but not as purist as the "WS" approach. It is still too early to determine which, if any, of these will gain dominance in the market.

Addressing is another topic sure to raise an eyebrow. It seems surprising that there is no standard <To> tag to put in a SOAP header to specify the destination and so many of the higher level standards have invented their own tag like this, but sometimes with a different name such as <Destination> or <Recipient> and sometimes using names that conflict with other proposals. This subject was kind of skipped by the other standard committees on the assumption that 'it was already done', and 'we just want to use the existing standard addressing'.

Lets give them credit for not always reinventing everything and attempting to work together, but give them low scores for not noticing that nothing actually existed. The lack of a strong ratified standard means there is no guarantee that different implementations will work with other implementations. It will work for all the basic cases, but quite possible fail in edge conditions.

The end result is that implementations of a SOAP based protocol on one technology must be tested with all the other different implementation to see if it works. WS-I was formed primarily to clarify the specific configurations, from all that are possible, that must be implemented to ensure interoperability.

“There is much work to be done to achieve integration with web services and companies should be praised for their generous efforts to move standards forward”

The role of WS-I in the eventual success of web services is too often overlooked.

When evaluating approaches, one thing to remember is that the issue is not about getting the bytes across from location to location, but rather the agreement on what that organisation must do in response. B2B interoperability is not fundamentally a technical issue, but a legal issue. Can one party prove that another party agreed to something? What happens if the message is received by the machine, but then accidentally lost? What if a request for a bid is received and someone responds, but due to a network outage the response does not get delivered for a day? These are fundamental questions that the EDI people have dealt with in the '80s and '90s; and it is a pretty good bet that the design for the '00s is going to incorporate what these groups have learned.

One of the areas that has received a huge amount of attention this year is that of the Business Process Execution Language (BPEL). The range of technologies known as web services is a stack of capabilities, each new layer building upon the last. We find ourselves today trying to get agreement on "choreography" which deals with definitions of the orders that certain messages can be sent and expected to be received in.

As a programming language, or at least an execution language, BPEL is aimed at programmers and works at a very fine grained level of defining a message, sending, receiving and otherwise manipulating those messages. These are the fundamental building blocks of any coordinated exchange of information. Some designers of choreography standards believe that everyone wants to sit down and define XML messages that will be sent back and forth.

This will end up being a lot of detail work. It is true that lack of constraints allows for virtually any pattern to be implemented, but on top of this will be built abstractions or patterns which will be much more useful to humans composing choreographies. In a way, the raw capability of BPEL is analogous to a computer machine code, where a program can jump from any point in memory to any other point in memory; powerful, but difficult to comprehend by mere mortals, so programming languages offer more "structured" programming languages which restrict the possibilities for jumps to patterns which are easy to maintain and debug. In this sense it is probably fair to consider BPEL the "byte codes" of the choreography virtual machine. BPEL programs are then quite likely to be compiled from more human-oriented process descriptions, which are built out of high level commonly useful patterns.

There is no question that BPEL is a first rate effort to define a choreography language. Some of the best computer language experts in the world are participating in this chance at a fresh approach to achieve some daunting goals. But I can't help wondering at why there is so much hype around a language that is not yet fully defined. In fact several vendors are claiming to support it today, never mind the fact that the standard is probably a year away from

being ratified. Industry analysts are telling the marketplace to be sure to bet BPEL capability if you are considering process technology.

Analyst surveys now count support for BPEL as the highest valued process standard, above all other existing standards. This situation is hauntingly similar to another situation just about 20 years ago. The US defence department wanted to reduce the number of failed software products and improve programmer productivity, so it promoted the introduction of the perfect programming language. A committee of the best computer language experts were assembled from both the universities and industry; they worked, toiled and strained over every detail to come up with a language that was precisely defined, with the specification fully elaborated so that every implementation would be as identical as possible. The language became

basic limitation of the 2.1 client is that it only knows about the 2.1 structures and does not know about the 2.2 tags. Because the validating parser does not ignore the unexpected tags, you lose the ability for different levels of clients to communicate, and some people believe that this is the correct way to be, perhaps because they never really understood the reason that XML was chosen. In some cases standard designers have argued that the new tags should have a different namespace and they allow unknown tags, but this means that every release of a protocol must have another namespace for each version, which can be a big burden after a while. The answer, of course, is that the validating parsers are tools that are useful during the implementation and testing of a protocol, but non validating parsers should be used in production environment, or at least the

"BPEL: so much hype around a language that is not yet fully defined"

known as ADA.

Furthermore, they promised that all defence projects going forward would be required to use ADA for software development. There was a huge response, in many ways ADA was a perfect programming language (at least for the times) and for a while ADA was in heavy use, but where is it today? When I look back over the achievements that brought us here, the most significant ones – the URL, HTTP, HTML, the browser, SOAP – were not designed by committees of experts. I don't mean to imply that BPEL is a lost cause. Far from it – what I have seen of it seems to be technically quite strong, but ADA was also technically strong, and I am questioning whether the current over-inflated interest in it is justified.

Where are web services going in the future? I pointed out earlier that XML was chosen because of its unique ability to allow messages to be exchanged between up-version and down-version peers without complicated negotiations. Since 1998 XML Schema has been developed, which has given us a huge leap forward in the ability to define XML Structures including basic data types. The invention of namespaces now allows us to compose a single XML structure from multiple different schema definitions. The namespace keeps the tag names from clashing, but also provides development tools with the ability to retrieve the schema and verify the structure.

Curiously enough there is a recent trend which is working to undermine this interoperability capability. The problem is how the verification is done. The validating parser will verify a structure at level 2.1 of a protocol and it will complain if there are extra tags that it does not know about. Remember, the

definition of "valid" should be relaxed at run time to allow unknown tags from up-version peers.

The trend, then, is for patterns and structures to be built on top of the current choreography level, once it is fleshed out, to enable more complex interactions, with richer semantic meaning, to more finely define what is expected from one party of the other party. Current B2B protocols work on the basis of a static 'a priori' interoperability agreement, which is like a contract for how messages will be handled.

I would expect in the future to see the setting up of this contract to become more and more a part of the everyday exchange. It is possible to progress to the point where the protocol will be able to negotiate terms of interoperability agreements between business partners, within limits preset by each side. Business rules will play a huge role in driving this kind of negotiation. But this kind of negotiation is at least 10 years away. The benefits of this type of internet scale integration will be tremendous. But for now, I see a lot of work to be done to make a few key standards and get market buy in. ☺

www.fusejournal.com
www.no3.com
www.jedi.org
www.cash-open.org