Steffen Staab
University of Karlsruhe
sst@aifb.uni-karlsruhe.de

Web Services: Been There, Done That?

What are Web services? The Stencil Group defines them as "loosely coupled, reusable software components that semantically encapsulate discrete functionality and are distributed and programmatically accessible over standard Internet protocols" (see www.stencilgroup.com/ideas_scope_200106wsdefined.html). Does this sound familiar? Long ago, people learned how to send software components and RPCs over HTTP (for an explanation of RPC and other terms, see the "Glossary" sidebar). Message exchange is increasingly based on XML, but so what?

Do you recognize hype when you see it? Let us measure fame as a Google count of Web pages that include a specified term. For example, measure the Semantic Web—now the topic of an established department in *IEEE Intelligent Systems*—against Web services. (The Semantic Web department in this issue also covers Web services.) The first Semantic Web language document (a working draft on the Resource Description Framework) was issued as part of the World Wide Web Consortium metadata activity in October 1997 (www.w3.org/2001/sw). Using Google's exact search, the term "Semantic Web" yields approximately 136,000 Web pages.

Now consider "Web services." The first UDDI document I could find dates from 1999, and the first W3C XML Protocol Activity preceding the Web Services Activity started in September 2000 (see www.w3.org/2002/ws).[1] The corresponding count delivers a whopping 3,210,000 pages. Compare these numbers with "artificial intelligence," whose count ranks at 1,410,000, "Corba" (Common Object Request Broker Architecture) at 1,650,000, and "TCP" at 7,640,000.

Even given the inevitable fallacies of these numbers, the overall result is clear: Web services have received a lot of hype, the reasons for which are not easily determined. Some of their benefits might even seem to waste away, once we touch on the nitty-gritty details, because Web services per se do not offer a solution to underlying problems such as

- How can I effectively and efficiently distribute computation efforts?
- How can I make a software component really reusable?
- How can I control and monitor these processes?
- How can I effortlessly integrate the components?

The following contributions delve into some of these issues. Wil van der Aalst describes the pitfalls of workflow issues, many of which we've encountered before. V. Richard Benjamins points to the various research in the 1990s into structuring procedural knowledge into problem-solving methods. Amit Sheth and John A. Miller discuss how a low initial entry barrier and simple technology are balanced against the long-term goal of easy integration. Christoph Bussler, Alexander Maedche, and Dieter Fensel strongly support this idea—in particular, by including semantics in their Web Service Modeling Framework. Finally, Dennis Gannon argues that Web services should not follow the well-worn and unsuccessful paths of other distributed-object technology. Rather, they should build on new kinds of applications, such as grid enterprises, which are only possible using technologies such as Web services. Have we been there before? Let's see.

—Steffen Staab

Reference

1. A. Preece and S. Decker, "Intelligent Web Services," *IEEE Intelligent Systems*, vol. 17, no. 1, Jan./Feb. 2002, pp. 15–17.

Don't Go with the Flow: Web Services Composition Standards Exposed

Wil van der Aalst, *Eindhoven University of Technology*

The recently released BPEL4WS is said to combine the best standards for Web services composition, such as IBM's WSFL and Microsoft's XLANG. (For an explanation of these and other acronyms, see the "Glossary" sidebar.) BPEL4WS allows for a mixture of block- and graph-structured process models, thus making the language expressive at the price of being complex. Although BPEL4WS is not such a bad proposal, it is remarkable how much attention this standard has received, while more fundamental issues and problems such as semantics, expressiveness, and adequacy have not received the attention they deserve.

Of course, having a standard is a good idea, but there are too many of them—and most die before they mature. Con-

1094-7167/03/$17.00 © 2003 IEEE
Published by the IEEE Computer Society

sider the growing list of acronyms: PDL, XPDL, BPSS, EDOC, BPML, WSDL, WSCI, ebXML, BPEL4WS—and these are just some of the acronyms referring to various standards in the domain. Another problem is that these languages typically don't have any clearly defined semantics. The only way to overcome these problems is to critically evaluate the so-called standards for Web services composition. In other words, don't just go with the flow.

Web services composition

Two trends are coming together in e-business that are creating both opportunities and pressures to automate business processes across organizational boundaries. One is the technology push created by enabling technologies taking XML-based standards and the Internet as a starting point. The other trend is improving the efficiency of processes from a business perspective.

After the dotcom crash, there was a pressing need to use Internet technology's potential by automating business processes across enterprise boundaries. Web services aim to exploit XML technology and the Internet by integrating applications than can be published, located, and invoked over the Web. A typical example of a Web services application is the Galileo system, which connects more that 42,000 travel agency locations to 37 car rental companies, 47,000 hotels, and 350 tour operators.

To truly integrate business processes across enterprise boundaries, merely supporting simple interaction using standard messages and protocols is insufficient. Business interactions require long-running interactions that are driven by an explicit process model. This raises the need for Web services composition languages such as BPEL4WS,[1] WSFL,[2] XLANG,[3] WSCI, and BPML. These languages are also known as Web services flow languages, Web services execution languages, Web services orchestration languages, and Web-enabled workflow languages. Before discussing such languages, I focus on the typical technology on which they are building.

Figure 1 shows the relation between Web services composition languages and other standards such as SOAP, WSDL, and UDDI. SOAP is a protocol for exchanging information in a decentralized, distributed environment using typed message exchange and remote invocation. It is an XML-based protocol that consists of

- An envelope that defines a framework for describing what is in a message and how to process it
- A set of encoding rules for expressing instances of application-defined data types
- A convention for representing RPCs and responses (SOAP can potentially be built on top of any transport layer, such as an HTTP-based infrastructure)

WSDL is an XML format for describing network services based on a standard messaging layer such as SOAP. A WSDL document defines services as collections of network endpoints, or *ports*. WSDL separates the abstract definition of endpoints and messages from their concrete network deployment or data format bindings. This lets us reuse abstract definitions, including *messages*, which are abstract descriptions of the data being exchanged, and *port types*, which are abstract collections of operations. The concrete protocol and data format specifications for a particular port type constitute a reusable binding. Associ-

Glossary

B2B	Business to Business
BPEL4WS	Business Process Execution Language for Web Services
BPMI	Business Process Management Initiative
BPML	Business Process Modeling Language
BPSS	Business Process Schema Specification
Corba	Common Object Request Broker Architecture
ebXML	Electronic Business Using Extensible Markup Language
EDI	Electronic Data Interchange
IBROW3	Intelligent Brokering Service for Knowledge-Component Reuse on the World Wide Web
IDL	Interface definition language
OMG	Object Management Group
PSM	Problem-solving method
RPC	Remote procedure call
SOAP	Simple Object Access Protocol
UDDI	Universal Description Discovery and Integration
WfMC	Workflow Management Coalition
WSCI	Web Service Choreography Interface
WSDL	Web Services Description Language
WSFL	Web Services Flow Language
WSMF	Web Services Modeling Framework
XLANG	Web services for business process design
XPDL	XML Process Definition Language

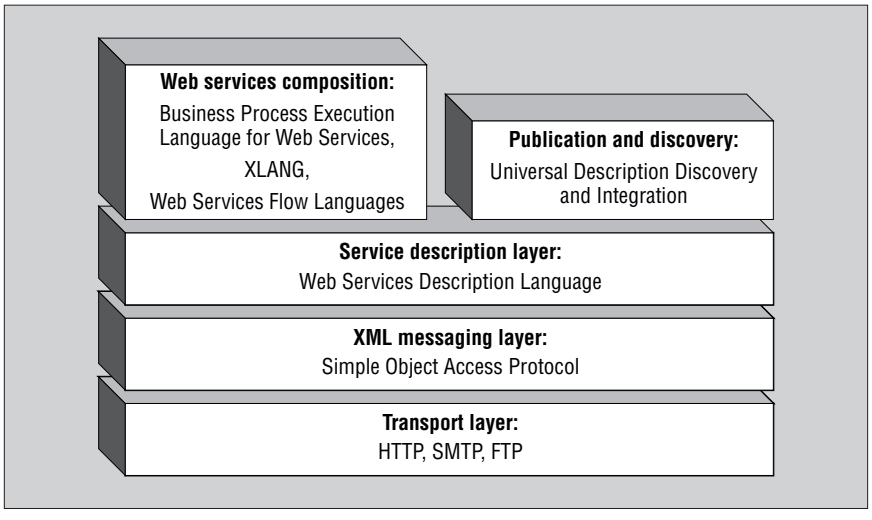


Figure 1. Overview of Web services technology.

ating a network address with a reusable binding defines a port, and a collection of ports defines a service.

UDDI defines a set of services supporting the description and discovery of businesses, organizations, and other Web services providers; the Web services they make available; and the technical interfaces we can use to access those services. Simply put, we can use UDDI to build "yellow pages" for Web services. Consensus currently seems to exist on the use of SOAP, WSDL, and UDDI, so I assume these standards will remain in place.

Web services composition languages build directly on top of WSDL. A language such as BPEL4WS both provides and uses one or more WSDL services. A WSDL service is composed of ports that provide operations. Each operation receives a message (one way), receives and sends a message (request response), sends and receives a message (solicit response), or sends a message (notification). WSDL services and the corresponding operations are glued together to provide composed services. To glue such services together, a process model must specify the order in which the operations execute. A Web services composition language provides the means to specify such a process model.

An important difference between WSDL and a language such as BPEL4WS is revealed when we consider the states. WSDL is essentially stateless because the language is unaware of states between operations. The only state notion supported is the state between sending and receiving a message in a request-response or solicit-response operation. Any technology supporting a Web services composition language will have to record states for processes that are more complex than a simple request response. Only by recording the state can we determine what should be done, thus enabling long-lived business transactions. This has triggered the development of languages such as BPEL4WS, WSFL, XLANG, WSCI, and BPML.

Overview of so-called standards

The BPEL4WS specification[1] builds on WSFL[2] and XLANG.[3] XLANG is a block-structured language with basic control flow structures such as **sequence**, **switch** (for conditional routing), **while** (for looping), **all** (for parallel routing), and **pick** (for race conditions based on timing or external triggers).

Unlike XLANG, WSFL is not limited to block structures and allows for directed graphs. The graphs can be nested but must be acyclic. Iteration is only supported through exit conditions—that is, an activity or subprocess iterates until its exit condition is met. The control flow part of WSFL is almost identical to the workflow language that IBM's MQ Series Workflow uses. This might be surprising, given that this workflow language is very different from most languages. For example, the "Death-Path Elimination" allows for the so-called "Synchronizing merge pattern." This way, routing is not restricted to explicit AND-joins and XOR-joins, as in most workflow products. This is a nice feature, but it is quite exotic and most systems don't support it.

Although the correspondence between the WSFL standard and IBM's workflow product might surprise people not involved in the standardization process, we can easily explain it by the fact that the same set of people (most notably, Frank Leymann) have defined both languages. We can make similar comments for XLANG and Microsoft's BizTalk Orchestrator. XLANG is based on Microsoft's current middleware solution and therefore hardly qualifies as a "standard."

Unfortunately, BPEL4WS, WSFL, and XLANG are not the only recently proposed standards. Sun, BEA, SAP, and Intalio have introduced another candidate for Web services composition: WSCI. Intalio also initiated the Business Process Management Initiative (BPMI.org), which developed BPML. OASIS and UN/CEFACT support ebXML. Part of ebXML is BPSS, which is yet another standard similar in scope to BPEL4WS, WSFL, XLANG, WSCI, and BPML. The abundance of overlapping standards for Web services composition is overwhelming. Some people refer to these competing standards without clear added value as WSAH—Web Services Acronym Hell.

Outside the Web services domain, other initiatives are attempting to standardize the specification of executable business processes. Most notable is the initiative of the Workflow Management Coalition. Since 1993, the WfMC has been active in standardizing both a workflow process definition language and the interfaces between various workflow components. In August 2002, the WfMC released XPDL[4] to support the exchange of workflow specifications between different workflow products. According to Jon Pyke, WfMC Chair and CTO of Staffware, XPDL is consistent with BPEL4WS but goes far beyond the standards for Web services composition. Clearly, many people working on standards for Web services composition did not benefit from the experiences in the workflow domain. Therefore, "been there, done that" comments are justified.

However, workflow vendors clearly have not adopted WfMC standards. Some systems can export to XPDL, but none can import XPDL from another system and still produce meaningful results. This is partly because after work on workflow standards for more than a decade, still no consensus exists on the workflow constructs that must be supported and their semantics. It is remarkable how many different interpretations of a join construct exist in contemporary workflow languages: "Wait for all (AND-join)," "Wait for first and reset (XOR-join)," "Wait for first and block until all have arrived," "Wait for all to come," and so forth.

Comparing BPEL4WS, XLANG, WSFL, XPDL, and WFM products

With respect to Web services composition languages, software vendors such as IBM, Microsoft, Sun, BEA, SAP, and Intalio have been the main drivers of development. This has resulted in an abundance of standards having overlapping functionality. When you look at the standards in more detail, you can see clearly that they are often based on existing products, just as WSFL is almost a copy of the MQ Series Workflow language. Standards that involve multiple software vendors are often a compromise between competing viewpoints. Consequently, such standards tend to be imprecise or unnecessarily complex. WfMC's XPDL is an example of a standard that is imprecise, thereby letting vendors have their own interpretation of it (and making it useless). BPEL4WS joins viewpoints from both WSFL and XLANG, thus making the language complex.

Given these observations, looking for objective measures for comparing Web services composition languages is useful. For the control-flow aspect of such languages, we can use some of the results from workflow research.[5] One way to compare standards such as BPEL4WS, XLANG, and WSFL is to use the set of workflow patterns available from www.tm.tue.nl/it/research/patterns.[6] Each of these patterns corresponds to a routing construct often required during workflow design.[7,8] The whole set of patterns has been

Table 1. Comparison of BPEL4WS, XLANG, WSFL, XPDL, and four workflow products.

Pattern	BPEL4WS	XLANG	WSFL	XPDL	Staffware	MQ Series Workflow	Panagon eProcess	FLOWer
1 Sequence	+	+	+	+	+	+	+	+
2 Parallel split	+	+	+	+	+	+	+	+
3 Synchronization	+	+	+	+	+	+	+	+
4 Exclusive choice	+	+	+	+	+	+	+	+
5 Simple merge	+	+	+	+	+	+	+	+
6 Multichoice	+	–	+	+	–	+	+	–
7 Synchronizing merge	+	–	+	–	–	+	+	–
8 Multimerge	–	–	–	–	–	–	–	+/–
9 Discriminator	–	–	–	–	–	–	–	+/–
10 Arbitrary cycles	–	–	–	+	+	–	+/–	–
11 Implicit termination	+	–	+	+	+	+	+	–
12 Multiple instances without synchronization	+	+	+	–	–	–	+	+
13 Multiple instances with a priori design time knowledge	+	+	+	+	+	+	+	+
14 Multiple instances with a priori runtime knowledge	–	–	–	–	–	–	–	+
15 Multiple instances without a priori runtime knowledge	–	–	–	–	–	–	–	+
16 Deferred choice	+	+	–	–	–	–	–	+/–
17 Interleaved parallel routing	+/–	–	–	–	–	–	–	+/–
18 Milestone	–	–	–	–	–	–	–	+/–
19 Cancel activity	+	+	+	–	+	–	–	+/–
20 Cancel case	+	+	+	–	–	–	+	+/–

used to evaluate and compare about 20 workflow management systems.

Table 1 compares three Web services composition languages, XPDL, and four concrete workflow management systems.[6] The first five patterns correspond to the basic routing constructs you can find in any language. The other patterns refer to more advanced constructs that most standards and products don't support. Every "+" refers to direct support—meaning the language has a construct that directly supports the pattern. A "–" refers to no direct support. This does not mean that realizing the pattern through some workaround is impossible. (For example, any of the constructs can be realized using a standard programming language, but this does not imply that there is direct support for all workflow patterns.) Sometimes there is a feature that only partially supports a pattern, such as a construct that imposes certain restrictions on the structure of the process. Support for such features is rated "+/–". Without going into details, we can make several observations.

First, BPEL4WS is indeed a combination of XLANG and WSFL when it comes to supporting the patterns. Second, WSFL and MQ Series Workflow are indeed identical when it comes to process specification. Third, XPDL seems less expressive than BPEL4WS. (In a way, we can view XPDL as the greatest common denominator of existing workflow languages rather than the least common multiple.) Finally, Web services composition languages and workflow management systems have relevant differences when it comes to supporting routing constructs. Of the four workflow management systems listed, only FLOWer (a workflow and case-handling system) is block structured like XLANG. The other three systems (Staffware, MQ Series, and eProcess) are graph-based like WSFL and XPDL.

The Web site of BPMI.org, one of the organizations proposing a Web services composition standard, states that

> BPMI.org defines open specifications such as the Business Process Modeling Language (BPML) and the Business Process Query Language (BPQL) that will enable the standards-based management of e-Business processes with forthcoming Business Process Management Systems (BPMS), in much the same way SQL enabled the standards-based management of business data with off-the-shelf Database Management Systems (DBMS).

The goal of obtaining standards similar to SQL for Web services is ambitious.

As history shows, such standards do not originate from vendors pushing their own products. Recall that the Entity-Relationship model by Peter Chen and the Relational Model by Eduard Codd enabled languages such as SQL. Although there are well-estab-

lished process-modeling techniques combining expressiveness, simplicity, and formal semantics (such as Petri nets and process algebras), the software industry has chosen to ignore these techniques. So, the world is confronted with too many standards, mainly driven by concrete products or commercial interests. The only way to stop this is to ignore standardization proposals that are not using well-established process-modeling techniques. This will force vendors to address the real problems rather than create new ones.

Acknowledgments

I thank Arthur ter Hofstede, Bartek Kiepuszewski, Marlon Dumas, and Petia Wohed for contributing to the results mentioned in this essay.

References

1. F. Curbera et al., *Business Process Execution Language for Web Services (Version 1.0)*, IBM, July 2002, www-106.ibm.com/developerworks/webservices/library/ws-bpel.
2. F. Leymann, *Web Services Flow Language (WSFL 1.0)*, IBM, May 2001, www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf
3. S. Thatte, *XLANG: Web Services for Business Process Design*, Microsoft, Redmond, Wash., 2001, www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm.
4. *Workflow Process Definition Interface—XML Process Definition Language (XPDL)*, WFMC-TC-1025, Version 1.0 Beta, Workflow Management Coalition, 2002, www.wfmc.org/standards/docs/TC-1025_10_xpdl_102502.pdf.
5. W.M.P. van der Aalst and K.M. van Hee, *Workflow Management: Models, Methods, and Systems*, MIT Press, Cambridge, Mass., 2002.
6. W.M.P. van der Aalst et al., *Workflow Patterns*, QUT tech. report FIT-TR-2002-02, Queensland Univ. of Technology, Brisbane, Australia, 2002; www.tm.tue.nl/it/research/patterns (also to appear in *Distributed and Parallel Databses*).
7. W.M.P van der Aalst et al., *Pattern-Based Analysis of BPML (and WSCI)*, QUT tech. report FIT-TR-2002-05, Queensland Univ. of Technology, Brisbane, Australia, 2002.
8. P. Wohed et al., *Pattern-Based Analysis of BPEL4WS*, QUT tech. report FIT-TR-2002-04, Queensland Univ. of Technology, Brisbane, Australia, 2002.

Web Services Solve Problems, and Problem-Solving Methods Provide Services

V. Richard Benjamins, *Intelligent Software Components, S.A.*

In the fall of 1996, on a flight home from the Knowledge Acquisition Workshop, some colleagues and I were outlining a new research project (an FP4 Esprit project) that aimed to link knowledge technology to the Web. The Web was gaining importance, and knowledge technology was looking to join the bandwagon. We were talking about the IBROW3 project, an Intelligent Brokering Service for Knowledge-Component Reuse on the World Wide Web (see www.swi.psy.uva.nl/projects/IBROW3/home.html).

The project envisioned reusable heterogeneous components located at repositories on the Web, which a *broker* (or software agent) would configure into a distributed program to solve a particular problem. The broker therefore had to know about the problem and the individual components' capabilities. The brokering process's result would be a distributed system that could, for example, classify edible and poisonous mushrooms using a public mushroom database and a classifier, both available on the Web.

The components that the IBROW project considered were *problem-solving methods* and ontologies. A PSM is a generic description of a reasoning process, independent of the domain to which it is applied. A PSM might solve classification tasks, for example, by classifying mushrooms, diseases, or financial transactions. Before we can apply a PSM to a particular domain, we have to check its assumptions, which describe the domain model's required characteristics.

For example, the *establish-and-refine* PSM, which B. Chandrasekaran introduced in the late 1980s, requires hierarchically organizing the domain knowledge. The PSM's assumptions capture the interaction between the PSM and domain knowledge explicitly such that reusing each of them in different situations becomes less troublesome. One main motivation behind research on PSMs and ontologies in the 1990s was to increasingly reuse and share code rather than develop code from scratch for each new problem.

Web services

We also see this motivation in Web services—pieces of software available on the Web that people can access through a standard protocol and execute remotely. Furthermore, when used together, Web services can deliver a complex functionality.

The three major ingredients for Web services are a description of the service, which uses the WSDL; the XML protocol SOAP, through which users access the services; and a UDDI directory, where you can find what Web services are offered and where. (For an explanation of these and other acronyms, see the "Glossary" sidebar.) With those ingredients in place, anybody can use a Web service, regardless of the programming language in which the service was originally defined.

Web services thus tackle the problem of heterogeneous sources and make them interoperable. Technologies such as Corba, RPC, and EDI had the same objectives, but those solutions needed their proper (that is, specific or dedicated) infrastructure, with the corresponding costs and implementation efforts. We can thus explain the success of Web services by viewing them as a technology based on maximal decoupling (and thus maximal reusability) available over an existing economic infrastructure (the Internet). Their power is not so much in their technology (the idea of RPC is nothing new) but rather that they offer a Web-native XML-based solution. So, we can rapidly design, implement, and deploy Web services.

These characteristics make Web services an interesting candidate for integrating heterogeneous enterprise applications. Organizations spend a lot of money making their internal systems interoperate with their partners' and providers' enterprise applications—and sometimes even with their internal applications. Web services offer an interesting alternative because they wrap existing (legacy) systems and let them communicate through SOAP over existing infrastructures (extranet or intranet). This is probably why research analysts at companies such as the Gartner Group and McKinsey are monitoring developments in this area.

The Semantic Web

Having said all these nice words, I must admit that current Web services technology solves only part of the problem. Distributed systems developers now have a common platform for rapidly developing or integrating complex software by configuring existing services. Through UDDI, they can look up the basic services they need expressed in WSDL, which they then can access by

Table 2. Resemblance between IBROW, on the one hand, and the Semantic Enabled Web Services project and the Semantic Web in general, on the other hand.

IBROW3 (Esprit FP4); IBROW (IST, FP5)	The SWWS project, Semantic Web	Comments
The Web is changing the nature of software development to a distributive plug-and-play process. The components concerned are problem-solving methods (generic algorithms) and ontologies.	Web services are orchestrated into complex services.	In IBROW, problem-solving methods (PSMs) and ontologies were the components being configured, versus Web services today.
PSMs and ontologies	Web services	PSMs and ontologies, when connected to the Web, deliver services. PSMs deliver reasoning services, while ontologies enable intelligent query answering.
IBROW integrates research on heterogeneous databases, interoperability, and Web technology with knowledge-system technology and ontologies.	Semantics and enterprise application integration	Web services can wrap heterogeneous sources to make them interoperable. Semantic agreement is currently hard coded in the wrapper.
The project aims at providing intelligent reasoning services on the Web as opposed to the more common information services.	From information overload to task delegation	The current paradigm of information retrieval causes much information overload. We need to delegate tasks to agents such that people focus on the interesting information.
The broker needs to reason about characteristics of PSMs, for example about their competence: Unified Problem-Solving Method Description Language (UPML)	Annotation of services, WSMF, DAML-S	UPML is used to characterize ontologies and PSMs in terms of their competence (capabilities) and requirements.

programming SOAP. However, a person must do all this work, and without much support. As more Web services become available, information overload will make finding the service you need difficult.

To overcome these problems, we must first recognize that current Web services technology is basically a syntactical solution and that the semantic part is still lacking. A Web service is described in WSDL, outlining what input the service expects and what output it returns. To exploit their potential (beyond enterprise application integration), Web services must be able to orchestrate themselves into more complex services. Thus, we need ways to combine individual Web services into a distributed, higher-level service.

The Web Service Flow Language, which can express the sequencing of individual services, is taking the first steps. WSFL lets the user decide which Web services to combine and in what order. However, we still need a framework that semantically describes services, such that software agents can locate, identify, and combine the services.

This is where the Semantic Web comes in. It semantically describes (annotates) content available on the Web such that software agents can understand and process the information. In this manner, Web services should form part of the Semantic Web (www.semanticweb.org). They would need a semantic characterization of the services they offer as well as a middleware for service discovery and orchestration, to hide complex technology from users. A user can thus consume a high-level functionality without noticing that a complex process is going on under the hood. This is a main objective of the Semantic Enabled Web Services project (SWWS, swws.semanticweb.org)—to develop such intelligent middleware along with a language to semantically express Web services capabilities.

If we look back at the original IBROW project (www.ibrow.org), we see that it resembles the SWWS project in particular and the Semantic Web in general. Table 2 illustrates this, showing some phrases and terms from the IBROW project and the corresponding terms used in current Semantic Web research projects.

We are now entering FP6, the Sixth European Research Framework Program. In addition, the IBROW projects (one in FP4 in 1998 and the current one in FP5 from 2000–2003) are about to finish. Since these projects began, many things have changed. The Web has drastically changed how we communicate, do business, and obtain information (where Google is the ruling champion). A revolution of new technologies has overtaken the IBROW project, such as Semantic Web technology and Web services. However, IBROW has played a crucial role in identifying relevant issues and major challenges and in bringing together the key people and groups that nowadays lead the European effort in Semantic Web services research.

The notion underlying Web services is nothing new. In this sense, it is "been there, done that." However, the infrastructure we currently count on is much more advanced than several years ago. In IBROW, we spent much effort making the underlying infrastructure work and creating the (heterogeneous) content. This hampered us from focusing completely on the real issues, such as semantic interoperability. In the current technological situation, coupled with a strong business pull (PSMs were more of a technology push from the knowledge-engineering community), we soon will have Web-accessible methods that solve real business problems through knowledge-intensive or intelligent Web services.

Acknowledgments

I would like to acknowledge the IST projects IBROW (IST-1999-19005), Esperonto (IST-2001-34373), and SWWS (IST-2001-37134).

Web Services: Technical Evolution yet Practical Revolution?

Amit Sheth and John A. Miller,
University of Georgia

Web services are here to stay. Both business and technical considerations will provide them with the staying power and tailwind to survive the hype. Despite their lack of a major technical breakthrough, Web services will initially succeed owing to the right confluence of evolutionary technological choices, low barriers and entry costs, and their strong standards-based approach. In the long run, however, they must be empowered by semantics and coupled with Semantic Web technologies to fulfill broader, longer-term promises.

Evolution in the right direction: Reasons for likely success

Progress in software componentry has been ongoing. The Web is becoming more than just a collection of documents; applications and services are coming to the forefront. History shows that incremental technological progress can produce dramatic effects, such as the advancement from FTP to Archie to Veronica to the Web. Web services might have a similar fate.

Several technical and technology-driven business advantages present themselves, boding well for a quicker and more successful adoption of Web services.

Low initial complexity

The first consideration is the low initial complexity (although the complexity of fully mature technology might be substantially higher), with low entry barriers and costs. Although the Web now consists of many components and technologies—several of which were added as the Web gained wider acceptance and matured—it indeed started with a remarkably small, simple core technology. That simple core—free availability and low- or no-cost initial adoption—played an important role in its growth and acceptance.

Unlike the Web, the Corba architecture was complex from the start (considering its various services and facilities) and required seasoned developers to use it. Also, most businesses had to rely on object request brokers that were not free, which made starting pilot projects—something large businesses always do before adopting any new technologies—difficult.

However, like the Web, the basic components of Web services are relatively simple, and the learning curve is low. XML is already known to the likely adopters and accepted as the data exchange standard by enterprises. Standards such as SOAP and WSDL are simple to use and learn. (For an explanation of these and other acronyms, see the "Glossary" sidebar.) Although UDDI as a whole (with all the details of white, yellow, and green pages) is not simple, the basic parts needed to get started are relatively easy to learn and use. Also, Web services can be developed with essentially no initial technology cost. Furthermore, just as the Web is used not only for publishing or sharing data but also as an application infrastructure, the complexity of Web services can be incrementally increased as more functionality is added.

The hype

The second consideration is the hype surrounding Web services, and the industry's ability to sustain interest over time. Consider XML. Although it has been significantly hyped—for example, it only addresses the syntactic interoperability issues, not semantic ones—it has succeeded. This is because it solves a limited yet important and well-understood business problem, and there is a groundswell of commercialization efforts around the use of XML technology. Some enterprise software such as Enterprise Resource Planning quickly adopted XML, broadening its adoption by businesses. In comparison, corresponding commercial activities did not support the hype of expert systems and other artificial intelligence technologies.

In the case of Web services, various software segments, including application servers and Enterprise Content Management,[1] have found them worthy enough to adopt. This gives the hype of Web services an ability to sustain with follow-through development and adoption.

Standards

The third consideration is the use of more widely accepted standards. XML already enjoys wide adoption in enterprises. Standardization efforts are well aligned and are structured to act much faster than what happened with Corba. Some in industry also see Web services as a natural follow-up to Electronic Data Interchange and ebXML, which already have significant commercial success.[2]

Loosely coupled architectures

The fourth consideration is that Web services are a good choice for loosely coupled architectures.[3] This means not allowing object references or requiring statefulness. Although the Corba architecture was more suitable for intra-enterprise environments, the technical features and choices of Web services make them more reusable and thus more appropriate for interenterprise and global environments.

Languages

Finally, Web services do not use languages that look like programming languages (that is, there's no interface definition language). This is what the industry is trying to accomplish with Web services.

Incremental advances in software componentary

The introduction of the object-oriented programming paradigm in the 1980s promised to make reusable software components a reality. Although there were certainly some gains in software reuse and programming productivity, the change was not substantial. The 1990s saw further attempts to increase software reuse, focused on the idea of software components and extended to distributed components. A notable effort in this area was Corba. It let components on different machines, using different operating systems and even different languages, communicate. From one viewpoint, Web services represent an evolution of Corba. We contend that Web services are more convenient than Corba and that eventually incremental improvement will lead to widespread use.

One difficulty in using distributed technology such as Corba is dealing with languages. Corba tried to eliminate this problem by moving the language into the background by defining an IDL. Although this is a good idea, an IDL looks much like C++, and developers must also understand language bindings to use Corba. If there was a way to replace language-like IDLs with higher-level specification, surely communication could be simplified using meaningful messages. Fortunately, Web services, defined as "self contained, self-describing modular applications that can be published,

located, and invoked across the Web,"[4] automatically get the Web-wide scope rather than primarily the enterprise-wide scope (at least initially) for Corba. In the near term, we see the use of a few well-targeted Web services within and across enterprises for well-targeted B2B applications and as integration points between enterprise software. Examples include integrating a document management system and portal management system within an enterprise, or interenterprise interfaces between different parts of a supply chain or ERP solution, which already uses XML for data exchange.

Intermediate to long term: Processes, QoS, security, and semantics

For the intermediate term, we predict distinct advances in

- Web processes
- Technical support for quality of service
- Technical support for security

Although we expect researchers to address QoS and security in the intermediate term, we expect support for Web processes to take longer, because initial research is just starting on these topics.[5] Because Web services are components, if individual services are limited in their capability, we can compose existing Web services to create new functionality in the form of Web processes. Web service composition is the ability to take existing services (or building blocks) and combine them to form new services.[6] In carrying out this composition task, we should be concerned about the efficiency and the QoS that the composed process will exhibit when it executes. This task of composing services to create efficient Web processes is analogous to designing a workflow.

Web service composition is an active area of research, with academic and industrial research groups proposing many languages. IBM's WSFL[7] and Microsoft's XLANG[8] were two of the earliest languages to define standards for Web services composition. Both extended WSDL, W3C's standard language used to describe the syntactic aspects of a Web service. BPEL4WS[9] is a recently proposed specification that represents the merging of WSFL and XLANG. It combines WSFL's graph-oriented process representation and XLANG's structural construct-based processes into a unified standard for Web services composition. Another effort in this area is ebXML, which lets enterprises conduct business over the Internet using an open XML-based infrastructure.

In contrast to these commercial XML-based standards, researchers are developing a unique Web service markup language called DAML-S.[10] According to the group of researchers working on DAML-S, it supplies Web service providers with a core set of markup language constructs for describing the properties and capabilities of their Web services in unambiguous, computer-interpretable form. DAML-S markup of Web services will facilitate the automation of Web service tasks including automated Web service discovery, execution, interoperation, composition and execution monitoring.[10]

If Web services are to provide not only an enterprise-wide but also a global architecture for application interoperability and integration, we will need to enhance Web services and processes with semantics. Many applications will require a Web process created from composing several Web services. In the intermediate term, we can create enterprise-scale Web processes (or service compositions) by adopting workflow management technology. However, to achieve Web services' full Web-wide and global potential, semantics holds the trump card. Using semantics involves describing resources with formal, machine-readable description. Resources in this case are Web services, and given that they wrap applications, they must be described both functionally and operationally.

As already recognized by DAML-S,[10] Web Service Modeling Framework initiatives,[11] the Work Group on Web Services at a recent Semantic Web workshop,[12] and others, semantics can play a critical role in developing Semantic Web services. This can lead to better discovery of Web services for reuse in a global, Web-scale environment that is not limited to one enterprise or a static collection of enterprises and where current syntax-based techniques do not work. Strong support for Web services discovery is also a prerequisite to developing Web processes[13,14] and addressing various issues of composition, interoperability, and execution.[15,16] There is demonstrable progress in developing semantics-based solutions when dealing with data, such as to achieve better interoperability and integration of information resources. However, the challenges of dealing with applications that are enabled by Web services are even more difficult. They will involve substantial further research and engineering that consider both functional and operational perspectives. The Large Scale Distributed Information Systems Lab's Meteor project is one of the projects looking at developing semantics-based solutions to QoS, discovery, and composition issues in the context of Web processes (see http://lsdis.cs.uga.edu/proj/meteor/SWP.htm).

References

1. R. Perry and R. Lancaster, *Enterprise Content Management: Expected Revolution or Vendor Positioning*, The Yankee Group, Boston, 2002.
2. A. Kotok, " The E-Business Continuum: Web Services, ebXML and EDI," WebServices.Org, June 2002, www.webservices.org/index.php/article/articleview/479/1/24.
3. D. Austin et al., "Web Services Architecture Requirements," World Wide Web Consortium, 2002, www.w3c.org/TR/wsa-reqs.
4. D. Tidwell, "Web Services—The Web's Next Revolution," IBM tutorial, 29 Nov. 2000, www-105.ibm.com/developerworks/education.nsf/webservices-onlinecourse-bytitle/BA84142372686CFB862569A400601C18?OpenDocument.
5. J. Cardoso and A. Sheth, *Semantic e-Workflow Composition*, tech. report, Large Scale Distributed Information Systems Lab, Dept. of Computer Science, Univ. of Georgia, Athens, Ga., 2002.
6. G. Piccinelli, *Service Provision and Composition in Virtual Business Communities*, tech. report HPL-1999-84, Hewlett-Packard, Palo Alto, Calif., 1999; www.hpl.hp.com/techreports/1999/HPL-1999-84.html.
7. F. Leymann, "Web Service Flow Language (WSFL) 1.0," IBM, Armonk, N.Y., 2001, www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf.
8. S. Thatte, "XLANG: Web Services for Business Process Design," 2001, www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm.
9. F. Curbera et al., "Business Process Execution Language for Web Services," 2002, www-106.ibm.com/developerworks/webservices/library/ws-bpel.
10. A. Ankolekar et al., "DAML-S: Web Service Description for the Semantic Web," *Proc. Int'l Semantic Web Conf.*, Springer-Verlag, Berlin/Heidelberg, 2002, pp. 348–363.
11. D. Fensel and C. Bussler, "The Web Service Modeling Framework WSMF," 2002, http://informatik.uibk.ac.at/users/c70385/wese.

12. A. Sheth and R. Meersman, "Amicalola Report: Database and Information Systems Research Challenges and Opportunities in Semantic Web and Enterprises," *ACM SIGMOD Record*, vol. 31, no. 4, Dec. 2002.

13. S. Narayanan and S. Mcllraith, "Simulation, Verification and Automated Composition of Web Services," *Proc. 11th Int'l World Wide Web Conf.*, W3C, 2002, pp. 77–88.

14. J. Cardoso et al., *Modeling Quality of Service for Workflows and Web Service Processes*, tech. report, Large Scale Distributed Information Systems Lab, Dept. of Computer Science, Univ. of Georgia, Athens, Ga., 2002.

15. J. Cardoso et al., "Semantic Web Services and Processes: Semantic Composition and Quality of Service," tutorial at Federated Conferences (CooPIS, DOA, ODBASE), 2002; http://lsdis.cs.uga.edu/lib/presentations/SWSP-tutorial-resource.htm.

16. S. Chadrasekaran et al., *Composition Performance Analysis and Simulation of Web Services*, tech. report, Large Scale Distributed Information Systems Lab, Dept. of Computer Science, Univ. of Georgia, Athens, Ga., 2002.

Web Services: *Quo Vadis*?

Christoph Bussler, *Oracle Corporation*
Alexander Maedche, *FZI Research Center for Information Technologies at the University of Karlsruhe*
Dieter Fensel, *Leopold Franzens Universität Innsbruck*

Web services have been touted for some time now as the technology-based silver bullet solution for many significant integration problems in information technology. These integration problems are of so different a nature due to the specific required functionality that we can't avoid asking right now, "Web services: *Quo vadis*?"

Here, we illustrate the immense discrepancy between the magnitude of the integration problems and the simplistic functionality that Web services provide. We propose the WSMF as the future direction of Web services so that they can cope even with the most real complex integration problems. (For an explanation of this and other acronyms, see the "Glossary" sidebar.)

Web services at the crossroads

Let's spotlight the state of the art of Web services and the situation of the "movement" (or "hype," as some might be more than happy to state). For the following reasons, we see Web services as being at the crossroads:

- They have been touted as the silver bullet for a (too) wide array of integration problems, currently being addressed by complex integration solutions such as B2B integration technology1 as well as simple technology such as Java for remote server invocation.
- Many competing, conflicting, and overlapping standards exist that address Web service functionality (see www.oasis-open.org/cover/sgml-xml.html).
- Many research projects are repackaging existing work as Web services work.
- Many start-up companies, including Cape Clear (www.capeclear.com), Actional (www.actional.com), and Intalio (www.intalio.com), provide technology implementing Web services technology.
- Major infrastructure companies (such as BEA, IBM, Microsoft, and Oracle) already provide implementations of Web service technology.
- Broad journalistic coverage of Web services exists in the trade press, such as *eAI Journal* (www.eaijournal.com) and ebizQ (www.ebizq.net).
- Web services have no underlying real conceptual integration model and are only defined as an implementation technology, such as SOAP (see www.w3.org/TR/soap12-part1/ and www.w3.org/TR/soap12-part2).
- No one has analyzed available and proven solutions such as EDI translation technology[2] or reported on lessons learned from deployments that could advance Web services technology functionality (more than 300,000 Electronic Data Interchange deployments exist worldwide—see www.disa.org/x12org/about/faqs.cfm).

This situation cannot continue indefinitely. We must consolidate the Web services space, focusing Web services technology on a well-defined and well-manageable set of integration problems. This includes the consolidation of the multitude of Web service standard proposals. Current Web services expectations are too high and too varied for one technology to solve all the given integration problems.

The silver bullet problems

Here we discuss some of the silver bullet problems, clearly showing the vast spectrum that Web services are said to solve.

First, SOAP was originally the acronym for the Simple Object Access Protocol. *Nomen est omen* (the term reveals its intent): the underlying paradigm follows the client-server model. The SOAP specification in conjunction with WSDL (see www.w3.org/TR/wsdl) clearly enables the definition of the external interface of a server in a particular way but not of its clients, violating the general peer-to-peer approach that most integration solutions require. The only pattern this supports is the one-way invocation with and without results between the defined roles of client and server.

In this sense, SOAP, in conjunction with WSDL, clearly implements yet another RPC model. This is also explicitly called out in the SOAP specification. Because it is based on the ubiquitous HTTP protocol in conjunction with XML as the message syntax, computer system boundaries are easy to overcome and the notion of the "better RPC" or even "better distributed-object-management technology" was born, leading to the wide awareness of Web services in the developer community.

Second, because SOAP messages can be transported over HTTP to implement the runtime invocation, bridging computer system boundaries is clearly easy owing to the commodity of the HTTP protocol. If only packaged applications like Enterprise Resource Management systems, such as Oracle or SAP (www.sap.com), were to expose WSDL interfaces, then integrating them with Web services would be easy, fast, cheap, manageable, and so on (better along all possible dimensions).

In such a scenario, Web services would be the "better Enterprise Application Integration solution," overcoming the current costly integration technology deployments. This is because the whole integration world would be a nice homogeneous sea of SOAP messages implementing WSDL operation invocations.

Finally, because we can easily exchange SOAP messages over the Internet, the proposal of using Web services as the only and superior way to implement B2B interactions between companies (interenterprise B2B communication) is not far off. The vision in this case is to very rapidly replace EDI, SWIFT (Society for Worldwide Interbank Financial Telecommuication, www.swift.com), and other existing B2B standards

(some of which have existed for over 30 years).

These three example integration problems alone span an impressive array of serious requirements for an integration solution that Web services would have to address:

- *Efficiency*: To scale on an industrial basis, Web services execution must be very efficient.
- *Expressiveness*: B2B interactions in supply chain scenarios are complex, requiring an expressive set of supported integration concepts.
- *Security*: Interactions within as well as across enterprises must be secured to prevent security attacks of all types, and nonrepudiation must be provided for reliable record keeping.
- *Reliability*: Remote and distributed communication must be reliable, and messages must be sent exactly once to ensure dependable interactions.
- *Manageability*: Interenterprise communication changes frequently, requiring easily manageable technology.

These requirements pose a high demand on a technology that addresses their implementation.

On the other side, we have the current state of Web services technology. The following standards currently define Web services: SOAP, WSDL, and UDDI (see www.uddi.org). This standards triple, even if fully implemented, is far from addressing even a subset of the requirements we've listed, because it is based on the primitive notion of synchronous remote invocation following a simple client-server model. Complex integration problems, such as B2B integration problems based on a peer-to-peer paradigm, are not at all in the scope of the standards triple owing to missing integration concepts, missing reliability protocol definitions, or missing security concepts.

Because Web services have caught on very impressively across the developer and business community, it is worthwhile exploring potential future developments to capitalize on the state Web services have reached.

Directions leading from the crossroads

We can only ask where to go from here if there are alternative paths going forward. Possible directions for Web services include it being

- The better RPC
- The better plumbing for application-to-application or B2B integration
- The better intelligent agent platform
- All of the above

There seems to be the implicit agreement in the community that Web services should be "the" new and "the" better technology for integration (especially for B2B integration). For instance, together, the vast array of industrial-standard proposals appears to provide an almost complete set of concepts required for implementing complex B2B integration. Examples of such integration include supply chain management, health care organization integration, or financial transaction processing.

Web services that can address complex integration problems require conceptual modeling based on a well-defined set of integration concepts.

Zapthink (www.zapthink.com/reports/poster.html#) has a poster that shows 135 of the 450 (at its time of publication) industrial standards and standards proposals classified and related to each other. They define business documents, security services, transport protocols, packaging, trading partner management, and many other relevant standards.

However, it's not just industry that is working on a complete set of standards addressing the complete integration space. Academic research is also tackling the vast array of problems in interenterprise integration, as we've seen at workshops such as Technologies for E-Services (TES 2002) or the Workshop on Web Services, E-Business, and the Semantic Web (WES 2002), or in certain research papers.[3–5]

Even more important, the big "however" is that the need for 135 (or 450) standards indicates a broken conceptual model, because the standards conflict, contradict, overlap, compete, or disagree with each other significantly. Another big "however" is that these efforts haven't really considered semantic unification at all. Only a few contributions (standards and research) highlight the need to solve the semantic mismatch problem that inherently exists in interenterprise integration and that no real integration solution can do without.

So, we need a comprehensive conceptual-integration model that combines all aspects of a solution for the interenterprise as well as intra-enterprise integration problem. From that we can derive a comprehensive modeling and execution framework that can address even the most complex integration problems completely.

Such a framework must provide a holistic and general conceptual-integration model that characterizes all integration problems. From that we can derive a set of expressive modeling constructs that model Web services, their invocation, and other requirements such as composition and security, addressing even the most complex integration scenarios. Furthermore, such a framework must provide an expressive and scalable mediation approach that can tackle the semantic unification problem through powerful mediation.

The future of Web services: WSMF

Web services that can address complex integration problems require conceptual modeling based on a well-defined set of integration concepts. This set of concepts must be able to represent all relevant aspects of integration, including document formats, processes, security, trading partner management, mediation, and discovery.

Because integration always occurs between two or more parties that are exposing services, any integration solution is "in between" the parties. This calls for an integration framework that clearly identifies system boundaries and modes of interaction between all elements. Furthermore, it must define the execution model for the conceptual model so that the application of concepts is uniform.

Last but not least, we need a methodology that suggests best practices for the modeling task when integrating services of several parties. The methodology suggests appropriately using integration concepts.

The WSMF is an important step toward such a conceptual model in conjunction

with an integration framework and methodology.[6] It provides all the integration concepts necessary for modeling even the most complex integration scenarios. It also provides a framework that defines the meaning of the concepts and suggests a methodology.

The major aspect of WSMF is, besides the mere definition of complex Web services, a strong support for mediating data as well as interaction behavior (message exchange patterns). In general, different services expose business data following different schemas and expect different message exchange patterns. For two or more services to interact, the differences in data structure and meaning as well as in message exchange behavior must be mediated. WSMF recognizes a significant set of mismatch patterns and provides mediation support. Only after the mediation problem is solved does integration become possible.

For Web service integration and Web services mediation to scale, the integration concepts must be represented as a formal language so that particular Web services definitions are self describing, machine interpretable, and machine executable. Ontologies are a suitable technology for this purpose, owing to their ability to represent semantics in the form of concepts.

The formal representation of services, their data, and their behavior are a precondition for automatic Web services registry, discovery, and composition. Only a strictly formal approach allows automatic and meaningful Web services composition. UDDI is far from providing such a formal mechanism today, and WSMF can serve as an important input to this necessary development.

With WSMF as a foundation, service-oriented architectures and computing become possible where all elements that require integration are represented as services. Once a service is formally defined, it is registered and can be discovered and composed to achieve its integration for business goals such as supply-chain automation. WSMF provides the formal foundation that allows service-oriented architectures to be executable in a complex business world.

We therefore strongly suggest that Web services will choose the path toward an integrated, complete conceptual-integration model supported by a framework defining the execution semantics and a methodology for successful definition of integration.

References

1. C. Bussler, "The Role of B2B Engines in B2B Integration Architectures," *ACM SIGMOD Record*, vol. 31, no. 1, Mar. 2002.
2. M. Sherif, *Protocols for Secure Electronic Commerce*, CRC Press, Boca Raton, Fla., 2000.
3. F. Casati, M. Sayal, and M.-C. Shan, "Developing E-Services for Composing E-Services," *Proc. 13th Int'l Conf. Advanced Information Systems Eng.* (CAISE 01), Lecture Notes in Computer Science, no. 2068, Springer-Verlag, Berlin, 2001, pp. 171–186.
4. C. Bussler, "The Application of Workflow Technology in Semantic B2B Integration," *Distributed and Parallel Databases*, vol. 12, nos. 2–3, Sept.–Nov. 2002, pp. 163–191.
5. A. Maedche and S. Staab, "Services on the Move: Towards P2P-Enabled Semantic Web Services," to be published in *Proc. 10th Int'l Conf. Information Technology and Travel & Tourism* (ENTER 2003), Springer-Verlag, Berlin, 2003.
6. D. Fensel and C. Bussler, "The Web Service Modeling Framework WSMF," *Electronic Commerce Research and Applications*, vol. 1, no. 2, 2002.

From Web Services to Grid Services

Dennis Gannon, *Indiana University*

There are two common criticisms of the Web services model. The first is that Web services define a mechanism for doing RPCs on an implementation of some XML-specified interface, using a slow protocol called SOAP. (For an explanation of these and other acronyms, see the "Glossary" sidebar.) This concept appears to be a second-rate reinvention of OMG's Corba or Java remote method invocation.

The second criticism is that the World Wide Web works, and Corba and the other distributed-object technologies are all hopeless failures. Web services are just trying to drive the Web down the same path to doom as these other RPC disasters.

As is the case with most folk wisdom (and other less benign forms of oversimplification), these statements contain some truth. In fact, attempting to use the Web services framework as a replacement for Corba or trying to use it to redo what Web technologies already do well is folly. What the Web services architecture brings to the table is a highly extensible, message-oriented middleware for building distributed, composable services. To illustrate this point, I look at how it is being used to build a component architecture for applications and services that makes ubiquitous grid computing a reality.

Grid computing

The concept of grid computing grew out of attempts to build large-scale distributed applications that group remote supercomputers, databases, and online instruments into tools used by groups of collaborators. Examples include the Particle Physics Data Grid and GriPhyn (www.griphyn.org), which involves a widely distributed team of physicists working to analyze the data streaming out of large international particle physics projects. Another example is the Network for Earthquake Engineering Simulation Grid (www.neesgrid.org), which is a national virtual collaboratory for earthquake engineering. A third example being organized is the Linked Environment for Atmospheric Discovery, which will provide the tools to do "better than real-time" detailed prediction of severe storms, such as tornadoes, based on the adaptive coupling of instruments, databases, and massive supercomputer simulations. Disciplines such as astrophysics, genomics and proteomics, and chemical engineering are building many other grid applications and testbeds.

To enable a distributed team of collaborators to build these applications, grids need an infrastructure of ubiquitous services that manage the resources provided to them. These services include

- *Security services*: Grid users need a way to authenticate themselves, and services are needed that reveal what the user is authorized to do. Privacy is also a serious concern for many grid collaborations.
- *Scheduling services and resource brokers*: If a grid application needs to use a distributed set of resources (such as advanced instruments, supercomputers, and databases) in a linked computation, the user must schedule these resources for concurrent use.
- *Information and data services*: Many large scientific applications require numerous data archives that are distributed around the world. These must be cataloged, indexed, and provided to the user to make information access transparent and ubiquitous.

- *Workflow services*: Grid applications usually require a complex set of interactions between many services and resources. For example, data must be extracted from instruments and then farmed out to data mining services or used as input to numerous simulations whose output is correlated to other sensor data. Orchestrating these tasks is the workflow engine's job.
- *Monitoring, messaging, and logging services*: Grids are complex and dynamic. At any given time, some component will be down or temporarily unavailable. Hence their applications must rely on autonomic services that keep track of alternate resources, redundant computations, and better network routes.

Many more standard grid services exist. The research community defining these standards is called the Global Grid Forum (see www.gridforum.org). Consisting of about 30 working groups and research groups and about 600 individuals, GGF meets three times a year. It occupies a place in the grid-computing world similar to that of the IETF for the Internet and W3C for Web standards. GGF depends on both the IETF and W3C because it builds on standards both groups are developing.

In GCF's early days, the standards were considered in isolation and often implemented in very different ways by different organizations. However, the emergence of Web service standards has changed that. Ian Foster, Carl Kesselman, Jeffrey M. Nick, and Steven Tuecke proposed the Open Grid Service Architecture as an extension of Web services to build a true component framework for the Grid.[1] By looking at OGSA, we can see where many of us believe Web services standards will evolve.

Web services

A conventional Web service consists of

- A set of *types*, which are usually described with XML schema documents
- *Messages*, which are XML elements consisting of a set of typed and named parts
- *Operations*, which can have an *input* message, a response *output* message, and possibly a *fault* message

An operation with an input and output message is called a *solicit response* operation, and an operation with only an input message is a *one way* message. Web services also contain *port types*, which group together a set of operations; *bindings*, which are associations between port types and transport and encoding protocols; and *ports*, which associate a name, binding, and network location.

WSDL provides an extensible schema for encoding all these elements to define a specific Web service (see www.w3.org/TR/wsdl). A complete WSDL document of a service is usually enough to generate the code needed to access that service—that is, it is a *service reference*. The standard protocol most services use is SOAP over HTTP; however, that is not a requirement. Because operations can be one-way and messages can be almost arbitrary documents, it is possible to implement a service over just about any transport protocol such as email, Corba Internet InterOperability Protocol, or a peer-to-peer protocol such as Sun's JXTA.

The Web Services Definition Language provides an extensible schema for encoding all these elements to define a specific Web service.

Another advantage of Web services is that the specification says little about the way services are actually implemented. Open-source and commercial toolkits exist for building Web services from standard Java, Enterprise JavaBeans, C/C++, Python, and C# using .NET.

The GGF Open Grid Service Infrastructure working group, led by Steve Tuecke and David Snelling, has taken advantage of WSDL's extensibility to define a grid service component model. OGSI defines a standard set of port types that grid services support and that applications can expect. One port type that every grid service must support is called the *grid service*, which provides a basic, essential introspection mechanism for services. Each grid service supports a set of *Service Data Elements*, XML documents that describe the service metadata and its state. The grid service port provides every grid service with two types of operations. One operation lets clients search the service's SDEs to answer questions such as

- What other port types do you support?
- What other SDEs do you support?
- What status and state information can you provide?

The other operation involves service lifetime management.

In his PhD dissertation, Roy Fielding described the Representational State Transfer model, based on the principles that have helped the Web achieve success.[2] REST emphasizes a few verbs applied to many nouns. The verbs in the Web are the HTTP operations, such as GET. The nouns consist of the network of URLs that constitute the Web. The REST model assigns most of an operation's semantics to the data, rather than to the operation's name. The grid service specification uses the message and document delivery model of Web services framework to achieve a similar goal. The grid service port provides a simple mechanism to convey information through service data elements rather than through complex interfaces of methods and parameters.

Every grid service is represented by a *Grid Service Handle*, which is a universal resource identifier that can be resolved into a specific *Grid Service Reference*, which might be the WSDL for the service instance. Using a separate URI handle lets services be easily replicated or maintained and still have a globally unique identity.

The other standard grid service port types include

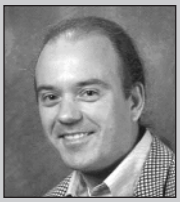
- *HandleMap*: This service provides the mapping between the Grid Services Handle and a current Grid Service Reference.
- *Registry*: This service binds service instance metadata to a registry.
- *Factory*: This can create instances of other services and generate stateful, transient instances of grid applications from a stateless, persistent service.
- *Notification*: Several standard ports are provided to allow asynchronous, publish-and-subscribe notification between services and clients.

Wil van der Aalst is a full professor of information systems and head of the Information and Technology section of the Department of Technology Management at the Technische Universiteit Eindhoven. He is also a part-time full professor at the Computing Science faculty at the Department of Mathematics and Computer Science at the same university. His research interests include information systems, simulation, Petri nets, process models, workflow management systems, mining, verification techniques, enterprise resource planning systems, computer-supported cooperative work, and interorganizational business processes. He holds an MSc in computing science and a PhD in mathematics, both from the Technische Universiteit Eindhoven. He is a fellow and management team member of the research institute BETA. Contact him at Eindhoven Univ. of Technology, P.O. Box 513, NL-5600 MB Eindhoven, Netherlands; w.m.p.v.d.aalst@tm.tue.nl.

V. Richard Benjamins is the director of R&D at Intelligent Software Components (iSOCO) in Madrid, Spain. His research interests include areas such as knowledge technologies, artificial intelligence, knowledge management, the Semantic Web, and ontologies. He received his MS and PhD in cognitive science from the University of Amsterdam. He serves on many international program committees and has been co-chair of many international workshops (including at IJCAI and ECAI) and conferences (EKAW). He is member of the *IEEE Intelligent Systems* editorial board. Contact him at richard@isoco.com.

Amit Sheth is a professor of computer science and the director of the Large Scale Distributed Information Systems (LSDIS) Lab, at the University of Georgia. He also founded Taalee, an enterprise software and Semantic Web technology startup based on the research at the LSDIS lab. He received his BE from B.I.T.S., Pilani, India, and his MS and PhD from Ohio State University. He is on the editorial board of six journals, has served on over 70 program and organization committees, and has chaired or cochaired six international conferences or workshops in the areas of the Semantic Web, digital libraries, multidatabase systems, and parallel and distributed information systems. Contact him at the Dept. of Computer Science, 415 Graduate Studies Research Center, Univ. of Georgia, Athens, GA 30602-7404; amit@ca.uga.edu; lsdis.cs.uga.edu/~amit.

John A. Miller is a professor of computer science at the University of Georgia and is also the Graduate Coordinator for the Computer Science department. His research interests include database systems, simulation and workflow, and parallel and distributed systems. He received his BS in applied mathematics from Northwestern University and his MS and PhD in information and computer science from the Georgia Institute of Technology. He is an associate editor for *ACM Transactions on Modeling and Computer Simulation* and *IEEE Transactions on Systems.* Contact him at the Dept. of Computer Science, 415 Graduate Studies Research Center, Univ. of Georgia, Athens, GA 30602-7404; jam@ca.uga.edu.



Christoph Bussler is a principal member of the technical staff at the Oracle Corporation's Integration Platform Architecture Group, based in Redwood Shores, Calif. He is responsible for the overall architecture of Oracle's next-generation integration platform technology addressing B2B, A2A, and ASP integration. He received his BA and MS in computer science from the Technical University of Munich, Germany, and a PhD in workflow management from the University of Erlangen-Nuremberg, Germany. He is a member of the IEEE Computer Society and ACM. Contact him at chris.bussler@oracle.com; http://hometown.aol.com/chbussler/index.html.

Alexander Maedche is the head of the Knowledge Management research department at the FZI Research Center for Information Technologies at the University of Karlsruhe. His research interests include knowledge discovery in data and text, ontology engineering, learning and application of ontologies, and the Semantic Web. He has taught courses at the University of Karlsruhe in the areas of knowledge discovery in databases, data and text mining, ontology engineering, and applied computer science and has given several tutorials on ontologies. He received a diploma in industrial engineering and his PhD in applied informatics, both from the University of Karlsruhe. He is a member of the IEEE and German Society of Computer Science. Contact him at FZI, Univ. of Karlsruhe, 76131 Karlsruhe, Germany; maedche@fzi.de; www.fzi.de/wim.

Dieter Fensel works at the University of Innsbruck, Austria. His research interests include ontologies, the Semantic Web, Web services, knowledge management, enterprise application integration, and electronic commerce. He is an editor of *Knowledge and Information Systems: An International Journal* (KAIS), *IEEE Intelligent Systems*, the *Electronic Transactions on Artificial Intelligence* (ETAI), and *Web Intelligence and Agent Systems* (WIAS). He has been involved in several national and internal research projects—for example, the IST projects COG, Esperonto, H-Techsight, IBROW, Multiple, Ontoknowledge, Ontoweb, SWAP, SWWS, and Wonderweb. He has been the project coordinator of Ontoknowledge, Ontoweb, and SWWS. Contact him at dieter.fensel@uibk.ac.at.

Dennis Gannon is a professor in and chairs the Computer Science Dept. at Indiana University. His current work is on designing software component architectures for distributed scientific applications and studying the architecture of grid systems. He received a BS from the Univerisity of California, Davis, a PhD in computer science from the University of Illinois, and a PhD in mathematics from the University of California, Davis. He is one of the cofounders of the Common Component Architecture project (now supported by the DOE Center for Component Technology for Terascale Simulation Software), the Java Grande Forum, and the Global Grid Forum. He is also the Science Director for the new Indiana Pervasive Technologies Labs and the Chief Computer Scientist for the NCSA Alliance. Contact him at gannon@cs.indiana.edu.

In addition to OGSI, there is a GGF Open Grid Service Architecture working group, which is exploring grid uses cases and defining the specific standard services that are needed for a new generation of grid applications. Other working groups are looking at specific topics such as security. Another important advantage of Web service messaging's document model is that security can be end-to-end rather than link level. In other words, we can encrypt and sign a message that might

pass through several intermediaries before reaching its destination. In standard RPC models, the security is often at the level of the supporting stream protocols, and the contents of method arguments are exposed at any intermediate steps in the processing.

Any component architecture must have a concept of component composition. The Web services community has produced a new draft standard, called the BPEL4WS, which is one candidate.[3] A BPEL4WS process defines a composite Web service described the way you might build a flowchart for an algorithm. It is composed of a set of activities that include invoking its composite services, waiting for input messages, copying data, and throwing exceptions.

Like any programming language, BPEL4WS has a rich control structure to sequence operations. While it is not yet known if BPEL4WS will also prove to be the appropriate composition tool for grid services, it does seem to be a strong candidate.

A final area of concern might be the relation of Web services and grid services to the Semantic Web. Others have observed that Semantic Web markup languages can play an important role in allowing agent technology to automatically use Web services.[4] Within the Grid Forum, there is a research group arising out of the Semantic Grid of David De Roure and his colleagues (see www.sematnicgrid.org) that is actively considering this question. ❐

References

1. I. Foster et al., "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," *Grid Computing: Making the Global Infrastructure a Reality*, F. Berman, A.J.G. Hey, and G. Fox, eds., John Wiley, New York, 2003.
2. R.T. Fielding, *Architectural Styles and the Design of Network-Based Software Architectures*, PhD dissertation, Dept. of Computer Science, Univ. of California, Irvine, Calif., 2000.
3. F. Curbera et al., "Business Process Execution Language for Web Services, Version 1.0," *IBM developerWorks*, 13 July 2002, www.ibm.com/developerworks/library/ws-bpel.
4. S.A. McIlraith, T.C. Son, and H. Zeng, "Semantic Web Services," *IEEE Intelligent Systems*, vol. 16, no. 2, Mar./Apr. 2001, pp. 46–53.