Practical Web Services

Charles Petrie • *Stanford University*

I've been asked to give a keynote at the IEEE Semantic Web Services in Practice conference (http://datam.i2r.a-star.edu.sg/swsip09), so I'm sharing my initial thoughts on the subject. I'll add more detail at the talk, but I hope that readers of this column will give me feedback before then.

Defining Terms

The title of this column might seem almost like an oxymoron to some, but we need to understand the terms better to have a discussion. Regular readers know I'm fond of precise definitions, so let's start deconstructing the terms.

I use "Web services" in the narrow scope I've previously defined ("No Science without Semantics," July/Aug. 2007), which includes but isn't limited to WSDL (www.w3.org/TR/wsdl), and excludes REST (www.w3.org/TR/ws-arch/#relwwwrest) but includes SA-REST (http://knoesis.org/research/srl/standards/sa-rest). There's a key difference between Web services and well-known technologies such as RPCs that justifies a distinction: Web services advertise a description of the service that's machine readable over standard Internet protocols so that a human and possibly a program can understand how to use the service, at least to the extent of knowing what messages are legal.

To clarify, I'm not talking about Web-based services. You can easily order a book from Amazon using your browser. That's not a Web service. Perhaps it uses a Web service in the background, but then where's the added value of the Web service technology for the user? And where, then, is the functional difference that would justify the distinction of a new term such as "Web service?"

This technical distinction is independent of the implementation detail of the communications protocol used, such as SOAP (www.w3.org/TR/soap). And yes, this definition of "Web service" is different from that of the W3C's, which also calls REST a Web service. However, the W3C definition is either so broad or so protocol-specific that it doesn't warrant a designation as a new technology. Worse, not focusing on the service descriptions, the feature that distinguishes Web services as a technology, creates implementations with impoverished service descriptions that are no better than just using remote procedure calls (RPCs). OK, I hope I've beaten that into the ground in previous columns.

It's the word "practical" that requires a lot of deconstruction: practical to whom and for what purpose? Web services should provide a "service" to someone, so it's key to understand the service consumer and use case.

Impractical Enterprise Services

Large enterprises have extensively implemented WSDL Web services for years as a software engineering technique. This has the immediate advantage of converging upon a stack of technologies required to implement them, which has the advantage of making programmers more of a commodity. Do you know WSDL, XML, and SOAP? Can you use axis2? Apache? Good — you're hired, for now. That's practical for the enterprises that employ them.

WSDL also facilitates the concept of software components, which could lead to the programming dream of reusable, maintainable software. Well, it could, were these components' functions better described. Unfortunately, programmers who use these functions within an enterprise typically treat them like a new version of subroutines. The next programmer hired to work on one of these components hasn't much more of a clue how to use it than he or she did with any other piece of software.

Actually, it's worse than this. The services I've seen are internal software components with the WSDL automatically generated by other programs, resulting in tens of pages of WSDL code useful only to other specific programs. And this is because entire processes have been converted to a Web service. Of course, it would be impossible to provide an understandable description of such a huge, monolithic service, thus negating the very feature of Web services that distinguish them from RPCs.

It's equally impossible for some other program to understand how to use this component automatically, a really long-range vision of software engineering that would prove practical for dynamic software configuration and repurposing. WSDL per se isn't terrific for these purposes, though it seems an incremental improvement over ad hoc subroutines. So, there's some teeny tiny bit of practicality here, just not as much as there might be. (With what? If you don't already know, thank you for your patience.)

OK, what about Web services on the open Internet? There are few, if any, practical Web services.

Amazon services are practical to the extent that a third-party retailer can use them, and such retailers do. But they have to study Web documentation. This is because the documentation is lacking in the WSDL itself. For instance, a programmer might like to know Amazon's product categories. In principle, this would be in the WSDL.

It's not: it's only on the human-only readable Web. If Amazon changes its product categories, the programmer would know only by reading the online documentation, were it to actually change. Again, the chief advantage of Web services isn't being used. And you might notice that Google has moved away from Web services to better programming tools, such as Ajax.

Programmers need a service description language so they can see what a software component written as a Web service does and how to use it.

Impractical End-User Services

Practical means the service does something useful, it's easily accessible with some standard software client, and it's easy to understand how to use the service based on the machine-readable description. Where are these services? If you know of any, now's your chance to speak up.

What about end users? Only a nerd masochist would use the Amazon Web services to buy books.

To be really useful, an open Web service would be able to be discovered easily by some easy-to-use search engine, perhaps Seekda (http://seekda.com). Now, this is potentially a good tool. Try, for example, searching for "hotel reservation." You get a list of WSDL services. Click on one and you get the list of operations of the service. Click on one of those, and it asks you to fill in the strings that will compose the message and be sent to the service. This is almost practical.

Except you don't have a clue what you're being asked to enter. Click, for example, on the "Reservations-Service," which is one of the services returned in the search. Oh, wait, there's no description yet. Well, just pick the first one in the results list. Its description is "seems to be an internal service." And if you click on the "Use Now" link, you have no idea what the operations do, individually or together. If you click on one of them, you're asked to enter strings that correspond to fields that clearly want you to enter some secret codes. Even the previous "Reservation-Service" has operations with names like "GetRGInfo" with a single message field called "nRGID."

Seekda is possibly the best product of this kind out there. But you see the problem, don't you? And we have known about this problem, and talked about it, almost since the beginning of the century. The solution is supposed to be semantics (the answer for which I've asked you previously to be patient, if you didn't already know).

Practical Services Need Semantics

For either programmers or end users, Web services have to have descriptions that can be easily and precisely understood by other people, if not machines. We need sufficient semantics to make Web services practical ("The Semantics of Semantics," Sept./Oct. 2009).

From where will these semantics come? One possibility is end users. Seekda has a nice facility that allows people to contribute a description of a service. Suppose that Seekda also let people enter descriptions of operations and message fields. That would go a long way. Suppose these descriptions could be formalized to some extent. That might start to be practical.

There are a lot of traps on the road to formalization. One is thinking only in terms of categories. When you do that, you say this is an instance of a "travel" service.

The first thing that's wrong is that anyone can say any service fits in any category. Maybe it's really a porn site. Second, this doesn't help you know what affect this service really has. Does it ultimately book a flight for you? And under what conditions? We've gone down this path before. It was called UDDI, and we knew it would fail from the start for fundamental reasons (http://logic.stanford.edu/talks/gap).

Letting the user community converge on descriptions Wikipedia-style might be full of traps, but it's better than what we have. And someone should try that. Of course, what's needed is a business mode — some set of incentives that would cause such development. Ideas? Anyone?

While we're talking about users, how about writing Web services? Where is Web 2.0 for Web services? They're just not going to catch on until we make it easier for users to write services, and this includes some kind of description language. And I don't mean situational calculus fluents. I mean a language in which it's easy to say that the service lets you reserve a car if you're 18 years old, with a credit card and valid driver's license on a weekday between 7 a.m. and 8 p.m. What would that language look like? Quiz later.

Some people (a phrase widely used by politicians) say that some set of top-down semantics developed by academics will solve this problem for everyone on the Internet. I don't think I need to say more about this.

Christoph Bussler and I have argued that, sadly, open Web services are a long way off ("The Myth of Open Web Services," May/June 2008). Please, please prove us wrong.

Practical Enterprise Services: A Challenge to the SOA Architects

Back to enterprises: there's a real business need here. Programmers need a service description language so they can see what a software component written as a Web service does and how to use it. Otherwise, it's just more software. Web services with descriptions could be used dynamically, and even among enterprises, eliminating tedious process construction by programmers from different companies. That's how it's done now, and it won't scale. That's not practical. But Web services without good semantic descriptions aren't practical either.

Users of enterprise services need such descriptions even more. That's why Bussler and I argued that "industrial service parks" will offer Web services with useful descriptions before they develop on the open Internet. At least within an enterprise, there's the possibility of converging upon useful descriptions, by fiat. And indeed, these enterprises' customers should demand them.

But where are these enterprise Web services now? Sigh.

Right now, Web services have been left to the programmers, and they're doing what I described earlier — that is, hard-coding again, but with all of the overhead of Web services, both in coding, implementation, and execution. A good Java programmer can beat a Web service designer at a complex task once, and maybe even in changing the code to adapt to new requirements. Furthermore, no customer would want, or be able, to use the enterprise services as they exist today.

I challenge enterprise architects to rethink how enterprise services should work if they're to be practical — that is, used by their customers as well as reused by their own programmers. Yes, it will cost resources, but there will be no successful use of Web services without a massive effort. In industry, we call this an "opportunity." But it's an outright challenge to the people who say they're working on a "service-oriented architecture."

Speaking of challenges, I've previously mentioned the Semantic Web Services Challenge (SWSC), of which I was the founding chair. This, I'm happy to say, has moved forward. It's now sponsored and run by the new EU Semantic Evaluation At Large Scale (SEALS) project. The SWSC remains a good test of technologies for Web services that purport to be practical. We have a sandbox of Web services, informal descriptions of them, and problems to be solved. If you think you have a good language for annotating/describing Web services, come see if they're practical for solving our problems, which are practical indeed. It's hard — that is, it's a challenge. But the SWSC is a good testbed out of which we might discover some good answers.

Service Descriptions Should Be Enforceable

My colleague Bussler points out that semantics are increasingly recognized as important for software engineering. Some of the semantics needed for Web services are more than just a description of the message types — the pre- and postconditions for using the service. This is now part of the Eiffel programming language (http://en.wikipedia.org/wiki/Eiffel_(programming_language)). Why have programming languages moved on and Web services have not?

Bussler and I think an important answer is execution. Today, service descriptions are entirely separate from the code. In Eiffel, the descriptions are used by the compiler to ensure that the code execution conforms to the description. Using POEM-style computational logic for writing both service descriptions and code is a long-range answer. But doing something more Eiffel-like would be a major step forward from the current state of the art. We academics tend to work on ontologies that should, somehow, describe services and leave it at that. Not very useful, is it? Modeling without execution rarely is.

I've been largely discussing WSDL Web services. But all of the issues apply to SA-REST as well. It might well be that SA-REST is a simpler technology that will catch on with at least end users. If not, enterprises and WSDL will fade away. Maybe. But the fundamental issues of developing description languages that people, if not machines, can easily understand remain just as difficult for SA-REST as for WSDL. The core issue is the nature of the service descriptions, not the service technology.

Web services as they stand today aren't very practical. The only good news is that most enterprise software, as it's done today, isn't very practical either, especially as enterprises have to exchange more information among themselves with larger and more dynamic systems. Web services with semantic and enforceable descriptions that would enable dynamic interoperability over the Internet, would be more practical.

The Future with Practical Web Services

There's a huge potential for enterprise Web services. For example, customization is a problem for enterprises that provide big software systems. Why should Fred install version two when he's done a lot of customization work on version one? How much testing will be required to get a new customized version of the system to work? And what does "work" mean?

There's another way to do this, which we've outlined in the Policy-Oriented Enterprise Management (POEM) project at Stanford (http://logic.stanford.edu/POEM). Given a good formal description of the Web services that compose versions of a system, as well as of the user's business logic, in theory, you can mathematically construct a provably correct set of new processes that do what the old ones did. Yes, this is a long ways off, but it's possible. Elimination of new version testing would be a very practical use of Web services. But even with today's software, there's a huge potential for enforceable good service descriptions.

Imagine that individuals as well as enterprises could set up virtual supply chains as needed, just by stating requirements and having a discovery and composition engine arrange the right connections among services. This would be practical with the right service descriptions. Compare this with what developers, sometimes from different companies, have to do today to write a BPEL process.

There's a future for some form of Semantic Web services in which we can all program the world using super browsers that might be thought of as "world wide wizards" (http://news-service.stanford.edu/news/2007/may2/petrie-050207.html). We already have almost all of the necessary technology. We need to realistically understand, collectively, the barriers to this vision and work to overcome it to make Web services practical as a step toward this vision. If we don't address those issues and find a way to get industry investment in semantic descriptions, then Web services remain, unfortunately, as they are now: impractical.

Charles Petrie has been a senior research scientist at Stanford University since 1993. His research interests include concurrent engineering, virtual enterprise management, and collective work. Petrie has a PhD in computer science from the University of Texas at Austin. He is EIC emeritus and a member of *IC*'s editorial board. Contact him at petrie@stanford.edu.