Toward an Acceptable Definition of Service

Steve Jones, *Capgemini*

XML and Web Service standards are helping architects improve the definition of service and provide a consistent framework for realizing service-oriented architecture's promises.

Methodologies such as Extreme Programming abhor it,[1] the Rational Unified Process requires it,[2] and it remains the way that developers document most systems and approve changes—via verbose, word-processor-created documents. While XP maintains that the truth is in the code, the reality is that modern computing languages don't have concrete mechanisms for the definition of service beyond a simple interface. They leave nonfunctional attributes either to a deployment or maintenance document or ignore them completely as something assumed. RUP takes a different approach and guides the team to document a system's nonfunctional attributes, but it too fails to provide a framework for the definition of service. Both methodologies claim that they work with service-oriented architecture, but neither was designed with SOA in mind. Both predominately focus on a project's delivery, not on the system's initial conceptualization.

Numerous frameworks define enterprise architecture from published approaches such as the Zachman framework,[3] proprietary approaches, and, of course, IEEE standards.[4] To various degrees, these approaches consider services; however, most are simply looking for a box in which to place elements and ensure that everything is considered, rather than taking a service-driven view as to what the boxes should be. Even when the approach's basis is service, tooling support is limited and ineffective, with the project team and the stakeholders communicating using an unstructured textual format, usually in Microsoft Word.

For service definition to succeed, there must be tooled support for the automatic flow of information among architects, designers, developers, maintainers, and, most importantly, the client. For SOA to succeed, it must close the tools gap (see Figure 1).

Standards are the only way

In the past, numerous efforts have tried to bridge the gap between the different communities to produce a single definition of service. It's a testament to their failure that verbose word documents continue to be king. The last few years have seen a move toward standardization in the software marketplace. This drive toward standardization and, indeed,

0740-7459/05/$20.00 © 2005 IEEE

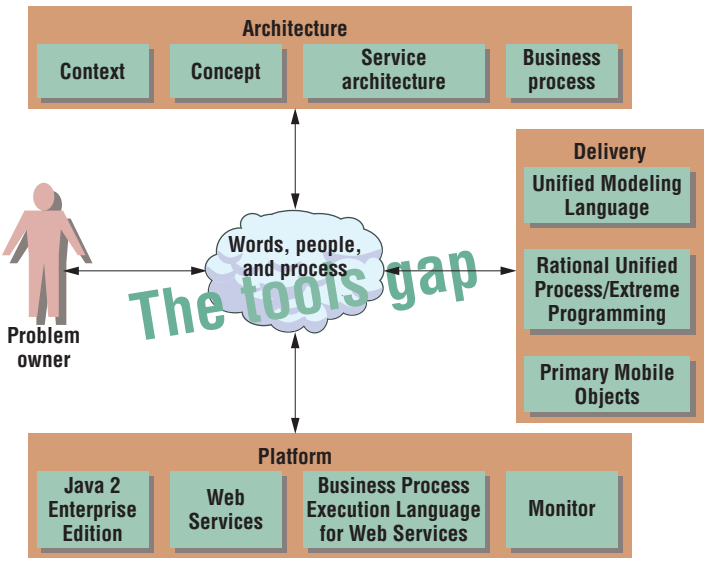

Figure 1. The tools gap between various structure information sets. The gap occurs because the tools don't communicate information to each other but require people to interpret the unstructured information.

commoditization has come from both vendors and end users to raise the bar for defining software systems. The computing industry and the IEEE in particular have a great history of creating marketplaces by enabling interoperability around standards. For SOAs to become truly successful, they must be based on agreed standards.

Standards haven't always had to come from such august bodies as the IEEE. A standard is an element that the industry widely accepts as such. However, it's often driven by standards organizations that bring together multiple parties to agree on a common way forward. Such standards have supported, or even ignited, the major technology explosions of the last 25 years (see Figure 2). The Wi-Fi sector is a classic example of standards creating a market that wouldn't have been possible without them.

We project architects must remember, however, that standardization is only successful when the industry actively supports it. CORBA is just one example of a near-standard that didn't reach full success because one major organization, Microsoft, failed to adopt it. To succeed, a standard doesn't have to be technically or academically the best way of undertaking a task. It doesn't even need origins in a recognized standards body; it needs only to become the de facto method for achieving its goal. Some would argue that the 802.11x standards aren't the best,[5] but surely none can argue that without such standards, Wi-Fi hotspots wouldn't be prevalent around the world. Standardization is the only way we can acceptably define nonfunctional aspects; otherwise, service interoperability would be impossible and would prevent market creation. If Sun can't read the service-level agreement (SLA) for Vodafone's service, how can Sun possibly meet it?

What constitutes service?

When determining the required standards to define a service, the first stage is to determine what those definitions must describe. A service has many characteristics that an architect must consider and can specify as required. An important point at this stage is to dispel the fallacy that all services require the same level of definition. A quote service, which returns a quote from a Unix fortune file, doesn't require security, reliability, transactional in-

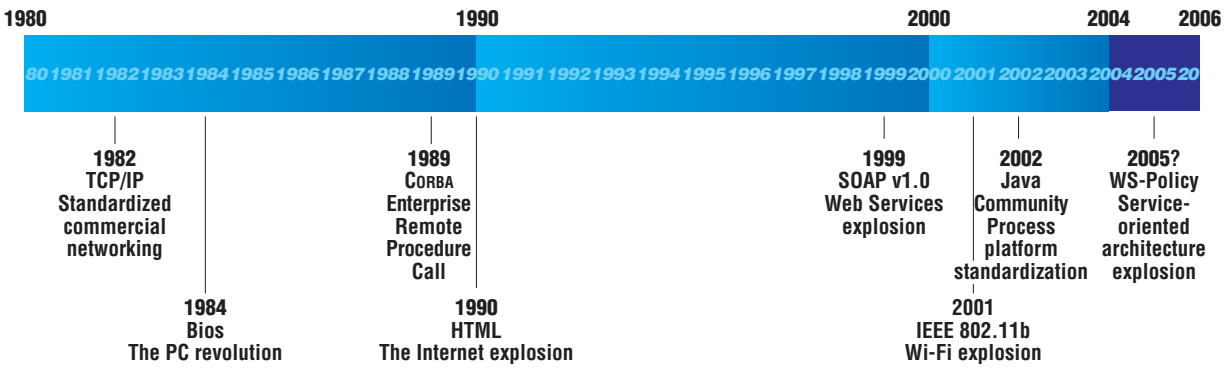

Figure 2. A timeline of standards and their market success.

tegrity, performance assurance, or anything beyond a simple interface definition. Such a service isn't part of a business's core functionality and therefore doesn't require an SLA for the business to deliver its goals. This last element is the key to understanding the scope of defining a service. A service's intention is to undertake certain functions to provide value to the business; its specification isn't just the direct service it provides but also the environment in which it undertakes those functions.

A by-product of this is that services architecture isn't just about technology, which is SOA's focus, but also a service's nontechnology aspects. This article considers those elements that we can realize via technology, but it's critical to remember that services can have many attributes and effects that technology can't define or measure. Indeed, any system is likely to use many nontechnology services. A *service* therefore is a discreet domain of control that contains a collection of tasks to achieve related goals. In a good service architecture, these often relate to business departments or subdepartments and their tasks. We can also decompose service into finer-grain services in the same way that organizations are decomposed to enable them to function. The IEEE provides a service to all of its members; the computing service is a lower-level service that works toward meeting the overall organization's goals (see Figure 3). Thus, in our definitions of service, we must consider not only the SLA at the lower levels but also the higher-level agreements' impact on those lower services.

The scope of service definition

The following is a nonexhaustive list of the numerous categories in which a service could require specification:

- performance,
- capacity,
- business organization,
- risks and issues,
- ownership,
- reliability,
- security,
- business impact,
- tolerance,
- service contract (preconditions, postconditions, and invariants), and
- dependencies.

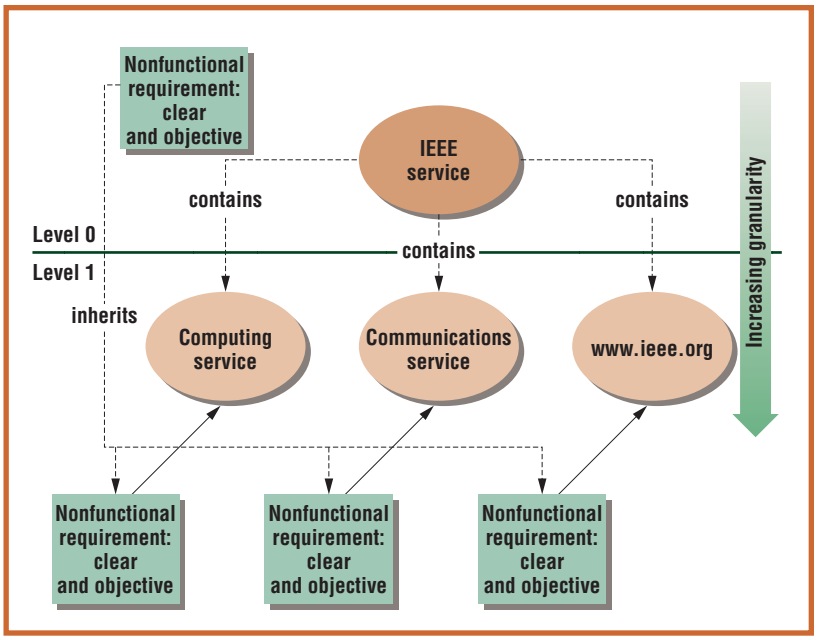

Figure 3. An example of a Level 0 and Level 1 service model. Level 0 is the highest level contextual service view and Level 1 is the first conceptual service view.

What's therefore required is a standards set whose deterministic elements we can define, manage, and monitor. Some elements will remain beyond computing's scope for some time to come because they're "soft" elements that require human judgment to be applied.

One interface, multiple policies

A car is a great example for a service definition. All cars offer the same basic functionality of "What does it do?" and "How do I get it to do that?" Counting only manual (stick-shift) cars gives an almost identical specification in terms of the interface of consumers on the services. We can describe the service by the following simple service definition.

Figure 4 shows two different views of the same information. The right side is a simple UML class diagram. The left side is that class's Web Service visualization. This clearly simplifies what a car does, but it also exposes the major functionality that the user requires.

This consumer-oriented view of service is central to SOA and differentiates it from object orientation. In OO, an object represents what it is, but in SOA, a service represents how its consumers wish to use it. So, how do we define a service?

Web Services Description Language—the basis for service

During the last two years, the Web Services Description Language[6] has become the stan-

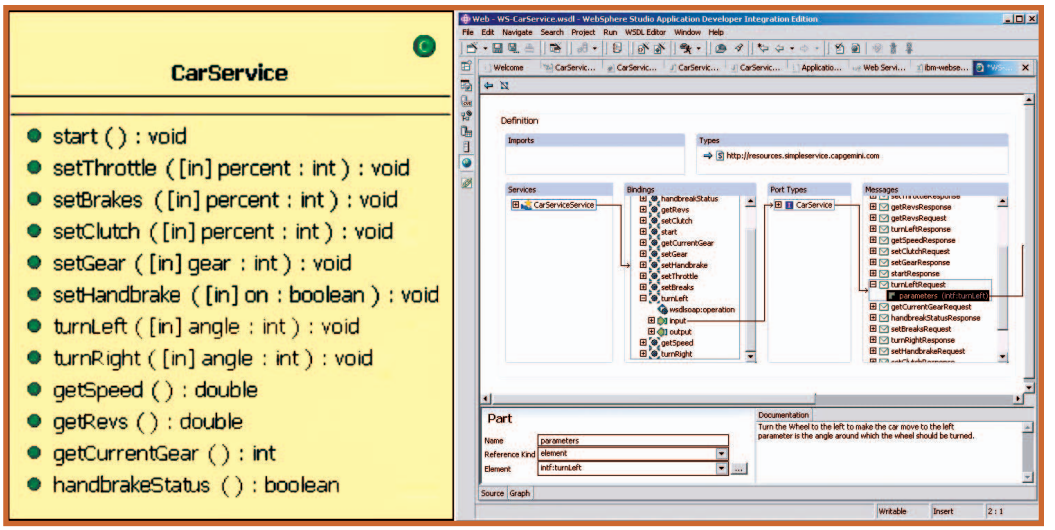

Figure 4. Two representations of a service: a UML class diagram and a Web Service visualization.

dard for defining not only services that use SOAP[7] but also generally any network service. This change, led by the two Java 2 Enterprise Edition software vendors BEA and IBM, lets us describe new technology artifacts as well as existing systems and services. This evolution is critical to SOA's wider acceptance because it doesn't mandate additional development to turn existing estates into service-based architectures. IBM's latest product offering, which comes under the impressively long name of IBM WebSphere Business Integration Server Foundation, enables architects to define a service-based architecture that lets mainframe-based CICS (customer information control system) applications interact with SOAP-based Web Services. BEA's Weblogic Platform offers similar facilities, providing a common container in which customers can control services. Along with these companies, Microsoft, Sun, Oracle, and SAP are working together with service architecture delivery organizations[8] and analyst groups[9] to create a more standardized view of SOA. These organizations are all betting that WSDL will become the de facto standard for describing services in the next few years and that defining existing systems using WSDL will help enterprises add agility to their IT environments. WSDL is an example of a standard that might fall short of academic purity but is enabling the industry to develop the next evolution of IT systems. By being major software vendors' and corporations' agreed standard, it's already enabling collaboration networks, and its value[10] has already outstripped any previous attempt at interenterprise collaboration, including electronic data interchange.

Web Services Invocation Framework—A mechanism for defining every service. Another mechanism for creating a standard is to release it into open source and have the market move around it. This is a more and more popular move with elements such as Linux BEA's BeeHive, Sun's Solaris,[11] and IBM's Web Services Invocation Framework.[12] Although WSIF is a Java-only technology, all the Java vendors are considering and, in many cases, implementing it. WSIF uses WSDL facilities to provide non-SOAP end points. Using WSIF, you could apply the service extensions and decorations I propose in this article not only to "traditional" SOAP Web Services but also to services that non-SOAP infrastructures provide, such as adaptors, mainframes, message-oriented middleware, and application server containers. This makes the Web Services approach to service definition more flexible because you can apply service definition retrospectively to existing systems.

WS-Policy. This is a standard for service decoration proposed by IBM, Microsoft, BEA, and SAP, which seeks to define a framework for attaching attributes to services. The proposal's goal statement explains,

> *WS-Policy provides a flexible and extensible grammar for expressing the capabilities, requirements, and general characteristics of entities in an XML Web Services-based system. WS-Policy defines a framework and a model for the expression of these properties as policies. Policy expressions allow for both simple declarative assertions as well as more sophisticated conditional assertions.*[13]

```
<wsp:Policy xmlns:wsse="..." xmlns:wsp="...">
 <wsp:ExactlyOne>
    <wsse:SecurityToken wsp:Usage="wsp:Required" wsp:Preference="100">
       <wsse:TokenType>wsse:Kerberosv5TGT</wsse:TokenType>
    </wsse:SecurityToken>
    <wsse:SecurityToken wsp:Usage="wsp:Required" wsp:Preference="1">
       <wsse:TokenType>wsse:X509v3</wsse:TokenType>
    </wsse:SecurityToken>
 </wsp:ExactlyOne>
</wsp:Policy>
```

Figure 5. WS-Security example.

This isn't the complete answer, but it's already helping new projects define how they consider their environments.

The initial approach is to use WS-Policy to define the security standards that the service enforces. This means that we already can migrate away from verbose textual documents to XML-based metadata to specify the encryption type, authentication mode, and audit to apply to the security policy.

The code in Figure 5 is straight from the draft specification and lays down how a service will specify the different forms of security tokens it will accept as authentication. By providing a simple interface for these elements, Capgemini is already working with clients, especially in the finance sector, to automatically define the security policies that apply to their external interactions. By providing a simple standard to act as a container for a service's decorations, WS-Policy is set to become one of the most important standards in the IT landscape. Where previous standards have focused on the desire to define a service's data interface, WS-Policy sets out to define not the service's "what" but its "how."

The car policy. In the car metaphor I described earlier, the security policy is the car key. The service definition doesn't change, but it's now constrained by a security policy that requires a recognized user to authenticate using a token (the key). Depending on the type of car, other nonfunctional constraints will also be applied to the service. For instance, the Ford Focus service will have a maximum speed of 120 mph and five gears plus reverse. The Aston Martin DB9, however, will have a maximum speed of 190 mph and six gears plus reverse.

Thus, using WS-Policy and WSDL, we'd have functionally the same interface but would have constrained the service definition by placing range restrictions in the schema and applying a security policy to the service as a whole.

The impact of invocation. One area where the Web Service standards are lacking is defining the impact of calling a particular function on a service. This is a key discussion area between the service delivery organizations and the tool vendors to determine the best way to define the impact of calling a service. The thought behind this builds on Bertrand Meyer's Design by Contract[14] to move toward a contractual definition of service. This includes the traditional contracts, precondition, postcondition, and invariant, and nonfunctional contracts that the service undertakes. Some elements of these contracts can be met by defining the range of the inputs that can be passed into a service. These range restrictions are simple to do in XML and will reduce the impact of unexpected "out of bounds" systems errors.

For example, with the car, it's impossible to depress the accelerator beyond zero degrees, as that's when the driver's foot is flat against the floor. You can't change the gear while the clutch isn't depressed, and the car shouldn't be able to move if it's not turned on. You could assign other restrictions—for instance, the time at which a service can be called—by using a policy that executes a business process that returns a success or failure criterion. Manners exist for resolving this, but we need a wider standard to ensure that all vendors implement service definition commonly.

Migrating from Word to tools. The primary justification for moving from a document-oriented approach to service definition to a contract-based one is to minimize the effort required to define and then, more importantly, enforce that definition. The migration I discuss here is in

Table 1

Range constraints for a car

Field	Definition	Range
Throttle	The throttle angle (mandatory)	60% to 0%
Brakes	The brake angle (mandatory)	60% to 0%
Clutch	The clutch angle (mandatory)	60% to 0%
Gear	The gear for the car (optional)	Depends on the car; from –1 to 5 is normal
Handbrake	Whether the handbrake is applied (optional)	Boolean
Steering angle	Overall steering angle; 0% is straight (mandatory)	Integer, –70 to 70
Speed	The car's speed (optional)	Double, depending on the car; range of –50 to 200
Revolutions	The current engine revolution (optional)	Double, depending on the car; range of 0 to 10,000

```
<simpleType name="PedalAngle">
  <restriction base="double">
       <maxInclusive value="60"></maxInclusive>
       <minInclusive value="0"></minInclusive>
  </restriction>
</simpleType>
```

Figure 6. The range restriction on a car expressed in an XML schema.

terms of process and projects, not the ability to port unstructured information into a structured and controlled environment. As I've shown earlier in this article, tools from vendors such as IBM are enabling architects and requirements analysts to enforce these service definitions using tools rather than documents and reviews. In a world where delivering and running services is ever more global, it's critical to ensure that the next generation of systems is built on a clear, unambiguous definition of service.

This migration's first phase is already here; with tools from IBM, Microsoft, BEA, SAP, Sun, and Oracle, architects and developers are using WSDL and WS-Security to define services. They define a service's raw contract in terms of its input range, data input, and the processes it can invoke as well as the credentials required to perform that invocation. This reduces the effort required to define services in Word documents. A good example of this is the impact of WSDL's range definition techniques, which minimize the need to excessively document the data models required for interaction.

Table 1 shows a textual definition of how we could define a car's constraints. We can replace this with an XML schema, a fragment of which is shown in Figure 6. We then use this XML schema to generate the project's associated service and code and to provide a direct mapping from the architectural and analysis constraints into that service's implementation and running. The contract can't be broken because we can fully trace it from the client requirements to system delivery.

This is a simple mechanism for maintaining these attributes' definitions in the project and a standard way to enforce modifications. For instance, if a new car's pedal range is only 40 to 10 degrees, it would just specify an extension of the pedal angle, which enforces a tighter restriction. We could then simply test that against the rest of the application, tracing the change's impact on all those elements that reference the previous XML artifact.

This is the start of a long road, one that will only be enabled by tools that let all the architects, analysts, and developers communicate around a common information model using standards that help define services. No one should expect to write the XML documents directly but to create those documents in a tooling environment that translates information between a project's different roles. A key challenge that tool and infrastructure vendors face is to ensure that the next generation of tools isn't just improving today's approaches but is helping to properly realize SOA's promises.

Next steps for standards and tools. Building on this first move from definition in Word toward a more formal definition of service, several standards are looking to augment the Web Service standards. They want to add even more decorations to services that will reduce the amount of documentation required and increase systems' traceability and manageability. The standards are too numerous to mention, including many competing standards. The key measure of their success will be the extent to which the tool vendors adopt them. When we can define and measure a service's main characteristics—namely, interface, contract, security, performance, capacity, and reliability—we will have dramatically reduced the effort of defining services via documentation.

But to succeed, the tools need to present this common information to each of a development project's major actors. This rendering of multiple views on a common information model is the basis of developments such as Model Driven Architecture.[15] Figure 7 details

the problems facing these tool vendors as they look to provide such an environment.

Fundamentally different users require different information subsets and different views on those subsets. It's unlikely in the short to medium term (one to three years) that an effective standard will enable multiple vendors to interact in such a manner. We should expect that some elements are possible between vendors—for instance, mapping analysis models to developers—but others, including the architectural models, will probably require proprietary solutions over this period. That isn't to say that we should avoid such solutions. The benefits of a common definition of service can't be underestimated: they're a key component for enabling organizations to gain a consistent picture of their IT estate.

Many tools are enabling us to move partway down the road toward a common definition of service. We envisage much change in the coming years in how organizations will be able to define and manage their environments by implementing a service architecture that the standards outlined in this article have defined. If service architecture in general and SOA in particular are to become a reality for most software projects, they'll need support from a tooling framework that enables all of a project's actors to interact in their own manner and for the tools to provide translations between the different descriptions of those actors.

This is the start of SOAs becoming a reality; it's an exciting time to be involved with software systems.

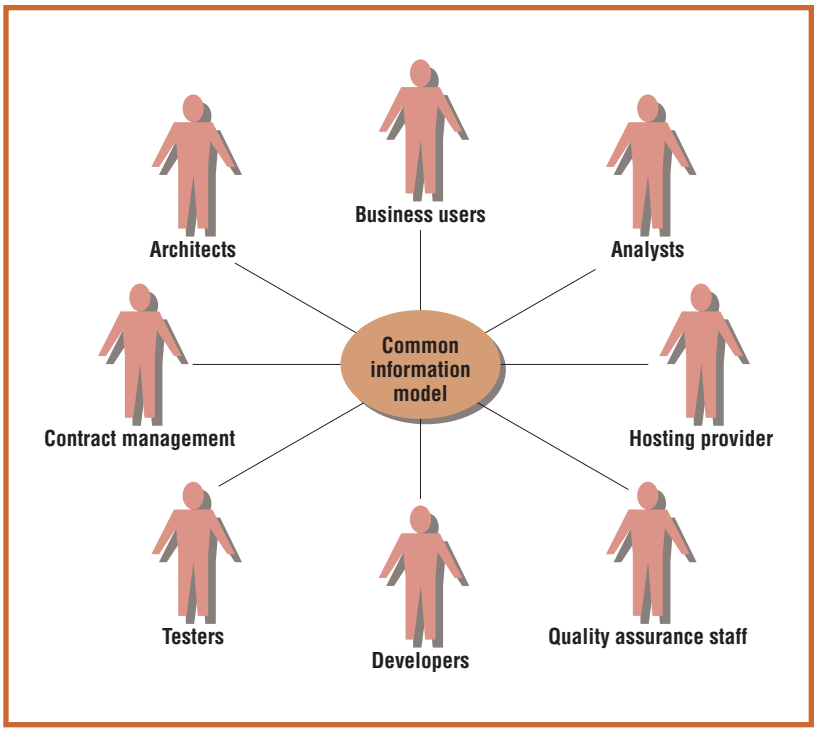

Figure 7. A common information model, or a central model that provides multiple views to ensure that everyone sees the right thing without interpretation.

Referenc es

1. K. Beck, *Extreme Programming Explained: Embrace Change*, Addison-Wesley Professional, 2004.
2. "Rational Unified Process," IBM, Mar. 2005, www-3.ibm.com/software/awdtools/rup.
3. J.A. Zachman, "A Framework for Information Systems Architecture," *IBM Systems J.*, vol. 26, no. 3, 1987, pp. 276–292.
4. *IEEE Std. 1003.1, Standard for Information Technology—Portable Operating System Interface (POSIX)*, IEEE, 2004.
5. J.-P. Saindon, "Techniques to Resolve 802.11 and Wireless LAN Technology in Outdoor Environments," *Security Magazine*, 8 May 2002; www.security-magazine.com/CDA/ArticleInformation/features/BNP__Features__Item/0,5411,77206,00.html.
6. E. Christensen et al., *Web Services Description Language (WSDL) 1.1*, World Wide Web Consortium (W3C) note, Mar. 2001; www.w3.org/TR/wsdl.
7. *SOAP Version 1.2*, World Wide Web Consortium (W3C) recommendation, June 2003; www.w3.org/TR/soap.
8. M. Op 't Land, "Usability of Capgemini's Integrated Architecture Framework (IAF) Compared with the Extensible Architecture Framework (xAF)," *Proc. Dutch Nat'l Architecture Congress 2004* (LAC 2004), 2004; www.lac2004.nl/docs/fvbg2hdsb83/Track8/M.%20Op%20%27t%20Land.pdf.
9. "SOA Blueprints," Middleware Research, Mar. 2005, www.middlewareresearch.com/soa-blueprints.
10. G. Papadopoulos, "Rules of the Game: Moore's Law Meets Gilder's Law Meets Metcalfe's Law," *Proc. Accelerating Change Conf. 2003*, 2003; www.sun.com/executives/perspectives/rules.html.
11. "Open Source and Open Learning," Sun, Mar. 2005, www.sun.com/2004-0720/feature.
12. B. Lublinsky, "Web Services Invocation Framework," *WebSphere Developer's J.*, June 2003; www.findarticles.com/p/articles/mi_m0MLX/is_6_2/ai_104209589.
13. A. Anderson, "IEEE Policy 2004 Workshop, 8 June 2004, Comparing WSPL and WS-Policy," *Sun Labs*, 2004; www.policy-workshop.org/2004/slides/Anderson-WSPL_vs_WS-Policy_v2.pdf.
14. B. Meyer, "Applying 'Design by Contract,'" *Computer*, vol. 25, no. 10, 1992, pp. 40–51.
15. *First European Workshop on Model Driven Architecture with Emphasis on Industrial Application*, 2004; http://modeldrivenarchitecture.esi.es/mda_worksProc&Pres.html

About the Author

Steve Jones is the head of Capgemini's Enterprise Java group and was the executive sponsor of Capgemini's membership in the Java Community Process. He also is an enterprise architect, consulting on technical and implementation strategy. His research interests include high-volume event-driven systems and using standards-based Java business process technologies to increase the flexibility of companies' infrastructures. He's an active member of industry expert groups, including those on Web services and business process standards. Contact him at 76 Wardour St., London, W1F 0UU UK; steve.g.jones@capgemini.com.