

Web Services Interoperability Specifications

A proposed conceptual framework for analyzing Web services interoperability issues provides a context for studying existing standards and specifications and for identifying new opportunities to provide automated support for this technology.



*Hamid R.
Motahari Nezhad
and Boualem
Benatallah*

University of New South Wales

Fabio Casati

Hewlett-Packard Laboratories

Farouk Toumani

LIMOS-ISIMA, France

Web services are becoming the technology of choice for realizing service-oriented architectures (SOAs). Web services simplify interoperability and, therefore, application integration. They provide a means for wrapping existing applications so developers can access them through standard languages and protocols.

Standardization simplifies interoperability: Instead of interacting with heterogeneous systems, each with its own transport protocol, data format, interaction protocol, and the like, applications can interact with systems that are more homogeneous. A standards-based approach helps reduce both development and maintenance costs for integrated systems. More specifically, Web services standards foster support of loosely coupled decentralized interactions.

We propose a conceptual framework that provides a context for analyzing existing Web services technologies to better understand their goals, benefits, and limitations. Using this framework to view the different approaches to interoperability can help to identify what kind of tool support is needed to leverage these technologies.

INTEROPERABILITY ISSUES

Several dimensions characterize SOA interoperability issues.

Integration layers

Because they are analogous to computer networks, we believe it's useful to study Web services interoperability issues in terms of layers to address various parts of the problem at different levels of abstractions. Typically, the structuring in layers goes from the lower levels, which are more horizontal—needed by most or all interactions—to higher layers, which build on top of the lower ones and might or might not be needed depending on the application. We identify the relevant integration layers to study the specifications for service interoperation, shown in Figure 1 in the context of a B2B interaction.

Messaging. The basis of any interoperability specification is the definition of a protocol for transporting information, regardless of the information content's syntax and semantics. In Web services, SOAP is the most common protocol at the messaging level. A service provider can further require (or allow) that messaging have certain properties. For example, the interaction can be reliable or unreli-

dination layer or the business-protocol layer is a somewhat arbitrary decision.

According to our criteria, specifications that are more horizontal belong in lower layers. For example, we consider trust-negotiation protocols to be at the business-protocol level. However, if, with time, a specific trust-negotiation protocol became widely accepted and adopted in many interactions, we would likely consider it as part of the basic coordination level.

Business services versus middleware services

Generally, interoperability involves both business and middleware Web services. Business services (such as an invoice-processing service) are the principal message sender or receiver in service interactions. Supporting interactions among business services might require third parties or intermediate services to provide service discovery, management, coordination, transactionality, and so on. As Figure 1 shows, these middleware services sit between business services and facilitate their interoperability. Some specifications are intended for consumption only by the middleware. For example, the SOAP middleware and its prebuilt libraries support the sending and receiving of messages, so developers don't need to know these details to implement Web services.

At the other end of the spectrum are the specifications and languages at higher abstraction levels. For example, business service developers use business protocol languages to create specifications attached to service descriptions, and they must be aware of other services' protocol specifications to implement their service so it can properly interact with others. In general, few specifications are consumed only by developers and of interest only to business services. Indeed, especially as technology matures, middleware will likely increasingly provide more support at higher abstraction levels.

Languages versus specifications

The difference between specifications and languages (or metaspifications) is an obvious but sometimes underestimated distinction that can help put standardization proposals into perspective. In some cases, standardization aims to establish a language that developers can use to define specifications. For example, service developers or other standardization consortia can use WSDL, WS-BPEL, and WS-Policy to provide specifications for interfaces, business protocols, and policies. In other cases, standardization efforts directly provide specifications. For example, interoperability protocols such as SOAP and WS-Transactions are specifications, not languages that developers can use to create other specifications.

As in most interoperability frameworks, Web services

efforts provide specifications for the lower layers of the interoperability stack, and they provide languages for the higher layers. This is consistent with the observation that higher-layer specifications are more domain- or even service-specific. Using a common and standardized language—for example, WS-BPEL for specifying business protocols—facilitates understanding of the service behavior as well as the development of tools that facilitate service creation and interoperability—for example, by supporting business protocol development and analysis.

Developers can describe a service by associating it with a set of specifications at the different layers. Standardization consortia directly provide some of these specifications—such as the services supporting HTTP, SOAP, and WS-Transactions—whereas domain-specific consortia (such as auto-makers) or the service developers

themselves develop others—for example, a service can support the interface `InvoicePayment` or the business protocol `InsuranceQuotation`.

Going forward, when using the term “specification,” we also include metaspifications, unless otherwise stated.

Human versus automated consumers

A service's interoperability information has two types of consumers (and usage scenarios): human users and applications. In general, human users need the interoperability description to determine whether to interact with a service and to decide how to develop clients that can correctly interact with that service. They can consume both formal and informal descriptions. Indeed, a service description often contains an information part specified in natural language. For example, WSDL includes a formal description of the message formats and operation signatures, along with descriptive information defining what each operation does.

Applications can use service descriptions to support interactions among services in various ways—for example, to verify that messages are exchanged in accordance with the defined protocol or to provide for nonrepudiation for operations with this requirement. To do this, applications typically read service specifications provided in some standard language, such as WS-BPEL protocol specifications. In general, applications can only consume formal specifications. Service-development applications also can access formal specifications to automate aspects of the development life cycle—for example, to generate stubs and skeletons from WSDL specifications or to assist users in evaluating the similarities and differences between two services, at least at the syntactic and structural levels.

As technology matures, middleware will likely increasingly provide more support at higher abstraction levels.

In contrast, applications that support binding, especially dynamic binding, aren't really interested in reading the specifications, so they don't care about the degree of formalization. Indeed, applications only care whether a service supports a certain (nonmeta) specification. For example, a shipping company client will need to know whether a particular service supports SOAP 1.2 and a certain interface—for example, `ShippingIntf`—developed by the shipping industry standardization consortium.

It's unlikely that applications will be interested in reading and parsing the WSDL file at runtime to, for example, view which operations are supported and assess whether they're semantically and syntactically equivalent or similar enough to the desired operation. If the service doesn't support the same interface and protocols syntactically, structurally, and semantically, there's no point in dynamically binding to them. With the exception of the binding information they provide, reading the specifications of the interface and protocols the service supports won't help.

Ongoing research efforts aim to increase the automation of service interoperability—for example, by letting client applications read the details of a provider service specification (such as a service's WSDL file) at binding time, and, if they find that a discovered service isn't compatible, letting them automatically resolve these differences.² Specifications targeted at humans aren't always intended for business services—and business service specifications aren't always intended for human use. SOAP, for example, is semiformal and meant to be understood by humans, but only so they can implement the middleware tools that support it.

Life-cycle activities

These activities classify interoperability specifications in terms of the service life-cycle activity they support. We broadly define three phases: binding, interaction, and management.

In the *binding* phase, an application (or developer) analyzes candidate service descriptions to determine whether they can engage in interactions. In general, a client needs all interoperability specifications at binding time because it needs to know whether a service supports a certain specification. This holds true at all integration layers. In addition, the client needs to verify whether the service's nonfunctional properties are within the desired boundaries. Service policies, using the WS-Policy standards family, usually declare most of this information, including support for certain specifications and nonfunctional properties. Thus, specifications allowing the definition of service policies are relevant to binding.

Specifications targeting humans aren't always intended for business services—and business service specifications aren't always intended for human use.

The *interaction* phase starts after service binding and incorporates all activities allowing service invocation, the exchange of concrete messages (along with the verification that messages are exchanged as declared), and the coordination of a set of interrelated messages with desirable properties such as reliability, security, and transactions. The specifications in this dimension include all those below the policies and nonfunctional properties layer in the service integration layers.

The *management* phase is concerned with monitoring service interactions, service provisioning, and managing service configurations and relationships. Management specifications provide for the definition of visible interfaces for service tracking, accounting, auditing, supervision, and control of service execution. Web Services Distributed Management (WSDM) and WS-Resource are two specifications with

such design goals.

SERVICE INTEROPERABILITY APPROACHES

Currently, there are two main SOA approaches for service-interoperation specifications: the WS-* family and ebXML (www.ebXML.org).³

Industrial software vendors are the primary contributors of WS-* specifications. These vendors develop incremental, modular specifications, introducing them in a bottom-up fashion with simple, horizontal specifications serving as the basic building blocks. The specifications stack is gradually extended with specifications at a higher level of abstraction. For example, WS-Security and WS-Reliability build upon SOAP to provide secure and reliable messaging.

The United Nations Center for Trade Facilitation and Electronic Business (UN/CEFACT) and the Organization for the Advancement of Structured Information Standards (OASIS, www.oasis-open.org) lead ebXML development, aiming to provide a comprehensive suite of specifications catering to all aspects of business-to-business interactions, including contractual relationships. These groups are developing specifications such as collaborative business processes, contracting, and secure and reliable messaging as parts of a single solution.

We use these dimensions as guidelines to discuss several interoperability approaches. Due to space limitations, we summarize our results in the following subsections and leave out several other interoperability specification families, including semantic Web services (www.daml.org/services) and specifications such as electronic data interchange (EDI) for standardizing business document exchanges and RosettaNet for standardizing product descriptions and business process interactions.⁴⁻⁶

Integration layers

Here, we summarize the results of analyzing interoperability specifications with regard to the integration layers. The “Service-Oriented Architecture Standardization Efforts” sidebar highlights features of the main specifications at each integration layer.

At the messaging layer, the WS-* and ebXML approaches use SOAP as the basic messaging protocol—they allow other protocols, but SOAP clearly has the strongest influence. However, they differ in the representation and in how they provide more advanced features. ebXML messaging (ebMS) encompasses a set of specifications for reliable messaging and message-level security on top of SOAP. In contrast, WS-* approaches

are more flexible, allowing the modular addition of more advanced features.

Although modularization is often a good practice, several groups of industrial vendors have presented overlapping and sometimes competing proposals for the same features (for example, WS-Reliability and WS-ReliableMessaging), thereby generating heterogeneity and hampering effective standardization. Similarly, in the security area, several competing standards—XML-based specifications such as the XML Key Management Specification (XKMS), WS-Trust, and WS-Federation, for example—provide features such as establishing trust and service federation on top of WS-Security.⁷

Service-Oriented Architecture Standardization Efforts

Here, we summarize standardization proposals at the different levels of the service integration stack. We refer to specifications proposed by industry groups as industrial specifications (IS), specifications approved by consortia such as W3C and OASIS as consortia-based standards (CS), and specifications that are under consortia review as CS-R.

WS-* family

The WS-* family is a group of specifications developed mostly by industrial software vendors.

Messaging

SOAP 1.2 (CS), a transport-neutral XML-based message-exchange protocol, is the messaging layer’s basic building block for more complex specifications. Other specifications at this level include

- WS-Addressing (CS-R), which defines mechanisms for addressing service end points and incorporating addressing information into SOAP message headers;
- WS-Reliability 1.1 (CS), which extends SOAP headers to guarantee message delivery based on acknowledgment messages;
- WS-Reliable Messaging (CS-R), which has similar functionality to WS-Reliability, but is built on top of WS-Addressing and uses WS-RM Policy to express reliability policy;
- SOAP-Attachments (CS), which attaches MIME-type documents to SOAP messages;
- WS-Security 1.0 (CS), which extends SOAP specification to provide confidentiality, integrity, and authentication; built on top of W3C’s XML-Encryption and XML-Signature; and
- Security Assertion Markup Language (SAML) 2.0 (CS), a language for defining security assertions (authentication and authorization) and mechanisms for exchanging them.

Basic coordination

Specifications at this level build upon each other to manage message exchange between two or more partners:

- WS-Coordination (IS) provides a framework for middleware-based coordination of some services’ activities;
- WS-Transaction (IS), which consists of WS-AtomicTransaction (which supports tightly coupled atomicity, consistency, isolation, and durability) and WS-BusinessActivity (which relaxes isolation and atomicity), uses WS-Coordination to coordinate transaction activities;
- Business Transaction Protocol (BTP) 1.0 (CS), consisting of atoms (loosely coupled, relaxes isolation) and cohesion (relaxes isolation and atomicity), develops its own coordination method;
- WS-Trust (IS), built on WS-Security, enables brokered or peer-to-peer issuance, exchange, validation, and dissemination of security tokens within a trusted domain;
- WS-SecureConversation (IS), built on top of WS-Security and WS-Trust, defines mechanisms for establishing and sharing secure context and deriving session keys in secure contexts;
- WS-Federation (IS), which shares attributes and authentication data over trusted domains, builds on WS-Trust; and
- Liberty Alliance (IS), which is built on SAML and WS-Security, has similar goals to WS-Federation.

Business-level interfaces

The Web Services Definition Language 1.1 (2.0) (CS) is the common language at this level. It defines a service as a set of end points implementing a common interface—that is, a set of operations associated with messages. WSDL can’t specify message exchange sequences.

Another issue is that new versions of a specification can appear during development of the specification layers, necessitating the layers' revision. This problem is inherent in relatively young technologies. At the basic coordination layer, because all ebXML transactions are binary (even multilateral collaborations are implemented as a set of bilateral collaborations), no specification for coordinating a set of trading partners exists. The WS-* approach includes several specifications for coordinating service operations. For example, WS-Transactions uses WS-Coordination to coordinate the execution of business transactions among a set of services.

At the business-level interfaces and protocols level, the WS-* approach aims to standardize a set of lan-

guages for defining interfaces and protocol specifications. Aside from such languages, these specifications don't make assumptions or impose constraints on how to define interfaces. This gives services more flexibility, but leaves them open to syntactical and semantic heterogeneities.

Languages such as the Web Service Choreography Interface (WSCI), Web Services Choreography Description Language (WS-CDL), and WS-BPEL build on WSDL to allow business protocol specification.⁸ WS-CDL and WS-BPEL are drawing the most interest from the industry. These two languages differ mainly in that WS-BPEL presents protocols from a service viewpoint, whereas WS-CDL describes the entire

Business-level protocols

Three specifications define the allowed message-exchange sequences between partners:

- The Web Services Choreography Interface 1.0 (CS) incorporates a set of constructs into WSDL definitions to specify business processes.
- The Web Services Choreography Description Language (WS-CDL) 1.0 (CS) proposes a set of constructs for defining the global choreography of services.
- The Web-Business Process Execution Language (BPEL) (CS-R) defines private and public business processes.

Policies and nonfunctional properties

In the WS-* family, policy- and nonfunctional property-related specifications include:

- WS-Policy (IS), a framework and generic syntax for defining Web services policies (capabilities and requirements);
- WS-PolicyAttachment (IS), which defines two mechanisms for attaching a policy to a service: incorporating it into subject elements and external policy attachment.
- WS-PolicyAssertions (IS) identifies a set of general policy assertions such as text encoding using WS-Policy syntax;
- WS-SecurityPolicy (IS), which defines a set of general policy assertions for security properties supported by WS-Security; and
- XML Access Control Markup Language (XACML) 2.0 (CS), which expresses fine-grained access-control policies for XML-based resources.

ebXML (CS)

Specifications in the ebXML family aim to be more comprehensive than their WS-* family counterparts and focus more on business-to-business applications.

Messaging

The main messaging specification in the ebXML family is ebXML Messaging 2.0 (CS). ebMS is based on SOAP and SOAP-Attachments specifications. It extends SOAP to guarantee message delivery, provides message-level security using SSL, XML-Signature, and XML-Encryption, and supports both synchronous and asynchronous messaging.

Basic coordination

Business-process schema specifications (BPSS) 1.01 (CS) provide basic coordination support for bilateral atomic transactions between business partners. It provides no support for two-phase commit.

Interfaces

Two XML-based specifications guide interface development:

- Core components 1.90 (CS) is a set of domain vocabularies, rules, and guidelines for defining data types, naming conventions, and complex data structures.
- Collaboration protocol profiles (CPP) 2.0 (CS) represents an XML-based syntax for expressing capabilities, requirements, and specifications of a trading partner represented in a profile.

Protocols

At the protocol level, BPSS 1.01 (CS) provides a set of customizable business-process specifications stored in a business library. In addition, collaboration protocol agreements (CPA) 2.0 (CS) model two parties' conditions and agreements for collaboration.

Policies and nonfunctional properties

At this level, CPP 2.0 (CS) doesn't provide a specific language for defining policies. Policies could be defined as general capabilities and requirements of partners in the CPP.

Table 1. Business and middleware service standardization efforts.

Approach	Business services	Middleware services
WS-* family	WSCI, WS-CDL, BPEL, WSDL, and WS-Policy	SOAP, ASAP, WS-Addressing, WS-Reliability, WS-Security, SAML, WS-SecurityPolicy, XACML, UDDI, WSDM, WS-Eventing, WS-Notification, WS-Trust, WS-Federation, WS-Coordination, WS-Transactions, BTP, and Web Services for Remote Portlets (WSRP)
ebXML	CPP and CPA	ebXML Registry, ebXML core components, and BPSS

Table 2. Language and specification standardization efforts.

Approach	Languages	Specifications
WS-* family	WSDL, BPEL, WSCI, WS-CDL, WS-Policy, WS-SecurityPolicy, and XACML	SOAP, ASAP, WS-Addressing, WS-Reliability, WS-Security, SAML, WS-Coordination, WS-Transactions, BTP, WS-SecureConversation, WS-Trust, and WS-Federation
ebXML	ebXML core components, CPP, and CPA	ebMS (messaging service) and BPSS

choreography of message exchanges among multiple partners. They both provide a protocol definition model that is conceptually based on process models (flowcharts and activity diagrams, with constructs such as sequences, forks, and joins) rather than on state machines or sequence diagrams, which are the other formalism generally adopted for defining protocols in the different areas of computer science.

In contrast, ebXML recommends customizing a set of common core components for defining the documents to be exchanged. This approach achieves interoperability collaboratively and offline. ebXML supports the definition of message-exchange sequences in business-process schema specifications (BPSS) and collaboration protocol profiles and agreements (CPP and CPA). Developers can use this combination of specifications to specify choreographies among multiple services. These detailed specifications cover many aspects of the interaction, far beyond the choreography's definition. For example, they include features for defining security and reliability requirements as part of the protocol definition, as well as other message exchange policies.

Service providers can define certain properties pertaining to the interactions at the interface or protocol levels. For example, a service provider might declare that an interface operation is transactional—that is, it can be undone. This means that the operation's properties are the same regardless of the business protocol in which the operation is used.

An alternative is to define operation properties at the protocol level, meaning that depending on the context in which the operation is invoked, the operation might—or might not—have certain properties or requirements, perhaps overriding analogous specifications defined at the interface level. This option provides more flexibility—which, as usual, increases the specification's complexity.

In terms of nonfunctional properties, the WS-* approach uses several policy or property definition languages to define service policies. The WS-Policy family is one example. However, WS-Policy doesn't mandate which policies can be defined and how. This is typically left to proposals put forward by various standardization consortia, depending on the kind of policies to be defined and on the applicative domain. For example, the recently published WS-RM Policy specification provides a set of predefined policy assertions for WS-Reliable Messaging. Although Oasis is more concerned with security and access-control policies, developers can use its XML Access Control Markup Language (XACML) to express service policies. ebXML doesn't support explicit formalization of service policies in a fragmented manner. Developers can express policies as service capabilities and requirements in the CPP and CPA.

Other dimensions

Table 1 summarizes the results of our analysis of interoperability specifications with regard to the business versus middleware services dimension; Table 2 summarizes the results of our analysis of interoperability specifications with regard to the languages versus specifications dimension. In this latter dimension, we can classify some specifications in both categories. For example, SAML provides both a language for defining security assertions and a specification for exchanging assertions among services. However, most standards are either a language or a specification. For example, SOAP and WS-Security are specifications, and WSDL and WS-BPEL are languages.

Table 3 classifies interoperability specifications with regard to life-cycle activities. Although all specifications are relevant at binding time, the table emphasizes specifications that are used mainly or only at this time. The

Table 3. Life-cycle activity standardization efforts.

Approach	Activity	Name/status/source	Key features
WS-* family	Binding	WS-Policy (IS)	Uses service policy specifications to match services and check compatibility during binding
		All other specifications	Checks the support for other specifications at binding time
	Interaction	Messaging	Messaging standard used for message exchange
		Basic coordination	Coordinates the exchange of a set of messages
	Management	WSDM 1.0 (CS)	Consists of two parts: a framework for defining interfaces to manage resources using Web services (MUWS), and a framework for management of Web services (MOWS) as resources according to MUWS
		WSRF (CS-R)	Provides methods for representing and accessing stateful resources as Web services; used by WSDM to model resources as services
ebXML	Binding	Core components 1.90	The library stores a set of common (customizable) information models and business processes; partners agree on (core-component-based) external interaction specifications offline; partners match candidate profiles (CPP) and generates contracts (CPA); CPAs are broader than protocols (they include legal issues, how long to store documents, and so on)
		BPSS 1.01	
		CPP 2.0	
		CPA 2.0	
	Interaction	ebMS 2.0	Synchronous and asynchronous messaging, reliability, message-interaction-level security
		BPSS 1.01	Specification of bilateral business transactions (basic coordination)
Management	Not addressed		

management aspect is somewhat newer in Web services. This isn't surprising, because management problems begin to arise once the basic technology is in place and service providers have deployed and are maintaining many Web services. The Grid computing efforts have greatly influenced specifications in this area.⁹ In this respect, Grid and Web services technologies are converging into a set of common specifications under the WS-Resource framework. ebXML approaches don't focus on management aspects.

With respect to the human versus automated consumers dimension, all lower-layer specifications in the WS-* family are generally intended for human users, although software vendors can use them to implement middleware components that support interactions according to these specifications.

Higher-level specifications, such as WSDL and WS-BPEL, are intended for both human users and applications. Service developers can read WSDL to understand how to develop client interactions with a service, and tools can automatically generate service skeletons from WSDL specifications. Similarly, humans use BPEL specifications at development time, but tools also use them at runtime to verify that interactions actually occur in accordance with the specifications.

The situation is similar for ebXML: Humans use ebMS core components and parts of BPSS that cater to coordination and transactional properties to develop middleware that allows interactions according to these specifications. In contrast, CPP, CPA, and parts of BPSS that provide template business-process definitions are

designed for both developers and applications to read, manipulate, integrate, and reason about.

OPEN ISSUES

The Web Services Interoperability Organization (www.ws-i.org) is trying to organize the crowded space of interoperability specifications by providing guidelines on how to implement services that comply with standards and how to use combinations of WS specifications to achieve interoperability. Currently, most WS-I efforts focus on developing interoperability profiles for generally adopted specifications such as SOAP; WSDL; universal description, discovery, and integration (UDDI); and WS-Security. Facilitating integration for services technologies requires addressing several other issues as well.

Adoption and usage

SOAP, WSDL, and WS-Security are currently the most widely adopted and used specifications.^{10,11} Although most platforms support UDDI, it isn't widely used. The general consensus is that this will change in the near future, with some dynamic binding possible as more services become available. Increasing interest in modeling service process and composition has led developers to support WS-BPEL to the extent that several free and open source implementations are available.

Many development and integration environments that provide security integration solutions also support SAML. Notably, some development environments support XML-Encryption and XML-Signature instead of WS-Security. Recently, OASIS adopted WSDM for

implementing Web service management solutions. Some business integration solutions support ebXML specifications, but as yet their adoption has been limited.

Leverage among specifications

Interrelationships among specifications aren't always clear. Typically, specifications in the WS-* family build upon each other, but the situation is blurred at higher levels. For example, BPEL is based on WSDL, but its usage relationships with WS-Transactions and WS-Coordination for providing transactional properties aren't clearly stated. ebXML, vertical specifications like RosettaNet, and semantic Web services approaches have evolved toward reusing (or allowing the use of) the WS-* family of specifications, such as SOAP and WSDL, at the lower levels.

Convergence

Different industry groups are pushing competing proposals for standardizing the same aspect, and formal bodies are supporting yet another version.¹⁰ Until now, this hasn't been an obstacle to the widespread adoption of Web services technology because software vendors have essentially focused on a few specifications. By identifying different facets of service interoperation, our interoperability dimensions will be useful as a framework for developing the standardization roadmap and will provide guidelines for service developers to use in determining their path through the complex maze of available specifications.

Although standardization is crucial for realizing interoperation in SOAs, it's by no means sufficient, especially at the higher abstraction levels. In our view, the effective use and widespread adoption of Web services technologies require new frameworks and systems for the conceptual modeling, analysis, management, simulation, and testing of service models and abstractions. These frameworks can provide a basis for matching service specifications or checking compatibility and replaceability among services as well as conformance with standards based on high-level models.

In addition to analysis, such frameworks can facilitate the resolution of heterogeneity and incompatibility through semiautomated adapter development.^{2,12} A key aspect of this work is the homogeneity of the (high-level) models at the different levels, while abstracting as much as possible from the details of each language or specification. In fact, modeling all aspects using the same underlying concepts and notations—for example, state charts or activity-based models—where appropriate, and adding extensions to cater to interoperability specificities, would make interoperability analysis easier at all levels and would even let developers verify that the different specifications for the same service are provided consistently. ■

References

1. F. Curbera et al., "Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI," *IEEE Internet Computing*, Mar./Apr. 2002, pp. 86-93.
2. B. Benatallah et al., "Developing Adapters for Web Services Integration," *Proc. Int'l Conf. Advanced Information System Eng. (CAISE)*, 2005, Springer-Verlag, pp. 415-429.
3. S. Patil and E. Newcomer, "ebXML and Web Services," *IEEE Internet Computing*, May/June 2003, pp. 74-82.
4. C. Bussler, *B2B Integration: Concepts and Architecture*, Springer, 2003.
5. G. Alonso et al., *Web Services: Concepts, Architectures, and Application*, Springer, 2004.
6. D.J. Kim et al., "A Comparison of B2B E-Service Solutions," *Comm. ACM*, Dec. 2003, pp. 317-324.
7. M. Naedele, "Standards for XML and Web Services Security," *Computer*, Apr. 2003, pp. 96-98.
8. C. Peltz, "Web Services Orchestration and Choreography," *Computer*, Oct. 2003, pp. 46-52.
9. I. Foster et al., *The Open Grid Services Architecture*, v.1.0, GGF informational document, Jan. 2005; www.gridforum.org/documents/GWD-I-E/GFD-I.030.pdf.
10. S. Vinoski, "WS-Nonexistent," *IEEE Internet Computing*, Nov./Dec. 2004, pp. 94-96.
11. N.S. Latimer et al., "2002-2003 Web Services Development, North America: User Wants and Needs," Gartner, July 2003; www.gartner.com/DisplayDocument?doc_cd=115909.
12. B. Benatallah, F. Casati, and F. Toumani, "Representing, Analyzing, and Managing Web Service Protocols," to be published in *Data and Knowledge Eng. J. (DKE)*; <http://dx.doi.org/10.1016/j.datak.2005.07.006>.

Hamid R. Motahari Nezhad is a PhD student in the Computer Science and Engineering School at the University of New South Wales. Nezhad received an MS in computer science from Amirkabir University of Technology, Iran. He is a student member of the IEEE and the Australian Computer Society. Contact him at hamidm@cse.unsw.edu.au.

Boualem Benatallah is an associate professor in the Computer Science and Engineering School at the University of New South Wales. He received a PhD in computer science from University of Grenoble, France. He is a member of the ACM and the IEEE. Contact him at boualem@cse.unsw.edu.au.

Fabio Casati is a senior researcher at Hewlett-Packard Laboratories in Palo Alto, California. He received a PhD in computer science from Politecnico di Milano, Italy. Contact him at fabio.casati@hp.com.

Farouk Toumani is a senior lecturer at LIMOS-ISIMA, France. He received a PhD in computer science from the National Institute of Applied Sciences, France. Contact him at ftoumani@isima.fr.