

A Model for Network Services on the Web

Reiner Kraft
IBM Almaden Research Center
San Jose, CA, U.S.A.

Abstract—Service oriented architecture (SOA) is gaining more momentum with the advent of network services on the Web. A programmable and machine accessible Web is the vision of many and might represent a step towards the semantic Web. This paper reflects upon existing Web services implementations and presents an abstract containment model as a basis to model Web services. A common model for Web services helps to compare different implementations in a principled way, introduces a common terminology, and provides a foundation for system designers to build additional functionality (e.g., security, work-flow) on top. Examples are shown on how existing Web services implementations can be represented using this model.

Keywords—Web, Web services, model.

I. INTRODUCTION

WEB services promise to promote the vision of a machine accessible Web, which can be used as a platform to conduct e-commerce and provide interoperability across organizations leveraging from a global Web infrastructure. The idea behind Web services is to use the Web as a platform for application to application communication. Programmatic interfaces that are made available over the Web are referred to as *Web services*. The goal of the W3C's Web services activity [1] is to develop a set of technologies that help to standardize protocols to foster interoperability. IBM and Microsoft jointly announced a new organization designed to ensure the interoperability of Web services, called the *Web Services Interoperability Organization (WS-I)*[2] to provide implementation guidance and support for customers deploying Web services, promote consistent and reliable interoperability among Web services, and articulate a common industry vision.

The term “Web service” and synonyms are used in a variety of different contexts and situations. There are already various existing implementations [3] [4] [5] that all use different terminology to refer to similar concepts. To compare different implementations in a principled way a common model is needed that can be used to represent these implementations. This further will help to avoid ambiguities between terms in different terminologies.

The paper presents an abstract model to represent Web services. This model can be used by system designers as a basis to add functionality (e.g., security, work-flow) to Web services. Access control as a critical security requirement is already integrated in the model in form of access control processors that make authorization decisions for a Web service or related components. In addition, Metadata represents an integral part of the model to further facilitate automatic processing through software agents.

The paper's main contribution is to define an abstract formal model for Web services, and justify design decisions and rationale. The proposed model does not enforce any particular implementation. Examples show how the proposed model can be used to represent different implementations such as Microsoft's .NET XML Web services [3] and IBM's Web Services Toolkit (WSTK) [6].

II. THE WEB SERVICE MODEL

The paper presents a model and proper definitions for components of the service oriented architecture. First, a careful definition is needed for the term “*Web service*” itself, since it is used in a variety of different contexts. Synonyms (e.g., “*XML Web service*”, “*network service*”, or simply “*service*”) are broadly used. The *Web Service Description Language* (WSDL) [7] introduces some general definitions that are abstract and separated from their concrete network deployment or data format bindings. However, the paper adopts a slightly different terminology used in object-oriented programming languages (e.g., objects, methods) that sounds more natural and intuitive to use to represent the related concepts. In addition, object-oriented concepts are widely used and well understood. If appropriate there will be a mapping provided on how a term or concept relates to the WSDL terminology.

A. Web service object

The paper prefers the term “*Web service object*” over Web service, since the term “object” intuitively relates more to concepts in object oriented programming, where an object encapsulates data and behavior. A Web service object is accessible over a network by a network endpoint (port). It represents a component with which applications can programmatically interact by exchanging messages.

In WSDL terminology a *Web service object* would refer to a “*service*” as a named set of abstract operations and the abstract messages involved. In Microsoft's .NET platform [3] the term “XML Web service” is used, whereas Apache SOAP [4] and IBM's WSTK [6] use the term “Web service” to refer to the same concept.

Note that a Web service object has an internal set of states during execution. In case subsequent messages are being sent to an object they might result in different responses, i.e. there is no guarantee that methods can have the property of “idempotence”. That means that the side-effects of $n > 0$ identical requests are the same as for a single request. Therefore a request message to an object should not be considered *safe*. Clients have to be aware

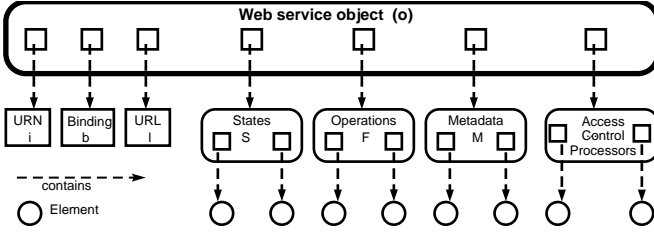


Fig. 1. A Web service object (see definition 1).

of any actions they might take, which may have an unexpected significance to themselves or others. More formally we define a Web service object as follows:

Definition 1: Web service object

A Web service object is a tuple $o = (i, b, l, \Sigma, F, M, A)$, where i is a non-empty string over an alphabet Σ^* representing a globally unique identifier (e.g., URN) [8], b is a string over an alphabet Σ^* representing a network protocol binding (e.g., SOAP [9] over HTTP [10]), l is a string over an alphabet Σ^* representing a network location (e.g., URL [8]). Σ is a finite set of states to represent the internal state of the object at a given time. F is the set of supported operations (*Web service methods*). M is a set of Metadata providing additional description for o . A is a finite set of access control processors $a \in A$ that are associated with o and perform authorization decisions for o . Σ , F , M or A can be the empty set \emptyset . A Web service object is a resource according to RFC 2396 [8].

A *Web service object* (see figure 1) is primarily used to aggregate different Web service methods (see definition 2). It acts therefore as a container for Web service methods, which are computational functions with a well-defined interface. The interface can be automatically derived from F , the set of supported Web service methods. It can be forwarded in a serialized form (e.g., XML) to a client that wants to discover how to use the Web service object. The interface description may be expressed in any interface definition language (e.g., WSDL). A client only needs to know a description of an object’s interface, the protocol-binding, and its network location to interact. The interaction is then performed by exchanging standardized messages depending on the selected protocol binding b (e.g., using the SOAP messaging framework [11] over HTTP). How a Web service object itself is implemented is not relevant and should be hidden to the client. Hiding implementation issues and providing a well-defined interface is the key concept of the Web services architecture.

A Web service object has an internal set of states Σ . Since Σ can be the empty set \emptyset , stateless operations on objects are also supported. In this case a request method may be considered safe or idempotent. Using σ to range over states in Σ the invocation of a Web service method then maps σ to another state σ' , i.e. $\sigma : \Sigma \rightarrow \Sigma$.

A Web service object o can be a member of zero or more Web service collections (see definition 3).

If A is the empty set \emptyset , then there’s no access control processor that makes authorization decisions for o .

We use the notation $o.m$ to refer to a Web service method m that belongs to o , and $o.m(p_1, \dots, p_n)$ represents in addition the parameter list for m .

To illustrate the usage of a Web service object consider an address verification Web service object. It would have an URN for a globally unique identification and a network address (e.g., URL) where it can be located on the network. Clients may use a discovery service (e.g., in form of a UDDI [12] directory) or file based discovery mechanisms to discover the network location where the Web service object is hosted. A service directory may contain also a service description (e.g., WSDL) that lists the interfaces of a Web service object. To invoke $o.m$ we can append a parameter for m to the URL of o . For instance, `http://acme.com/o.asmx` could represent the Web service object o , and `http://acme.com/o.asmx?method=m` might represent an invocation of m (with no parameters). The model does not enforce a particular implementation.

If there’s an access control processor specified for that Web service object, it would make an authorization decision whether an incoming client request would be granted.

B. Web service method

A *Web service method* represents an operation on a Web service object. The paper adopts the term “method” since it is widely used in object-oriented languages to represent behavior of an object in form of a function.

Definition 2: Web service method

A method is a tuple $m = (i, p, r(p), M)$, where i is a non-empty string over an alphabet Σ^* representing a globally unique identifier (e.g., URN) [8], p is a string over an alphabet Σ^* representing a set of input parameters, r is a function $r : \Sigma^* \rightarrow \Sigma^*$ that maps p onto a result string over an alphabet Σ^* representing the output (result) or return value(s) of a computation. p and $r(p)$ may be the empty string ϵ . M is a set of Metadata providing additional description for m . M can be the empty set \emptyset . A method m has to be a member of exactly one object O , so $\exists O | m \in O$. It is a resource according to RFC 2396 [8].

A Web service method represented through an abstract interface can be accessed to perform some computation (see figure 2). In the proposed model a method is also a resource. It may have a set of properties in form of additional Metadata associated with it. On an abstract level given some input string p over some alphabet Σ^* a Web service method will perform a computation based on that input string and may return a result string $r(p)$ over some alphabet Σ^* as a result of that computation. The computation may change current state $\sigma \in \Sigma$ in o to σ' . It may also be the case that the computation diverges and loops forever, i.e. there’s no state σ' . Thus a Web service method may not return a value at all.

Note that input parameter p and the result of the function $r(p)$ in programming languages typically have an associated data type of a type system. The definition supports this, since it allows that any arbitrary string can be used to encode some typed set of parameters. For instance, $p, r(p)$ could be encoded using XML Schemas [13] to associate data

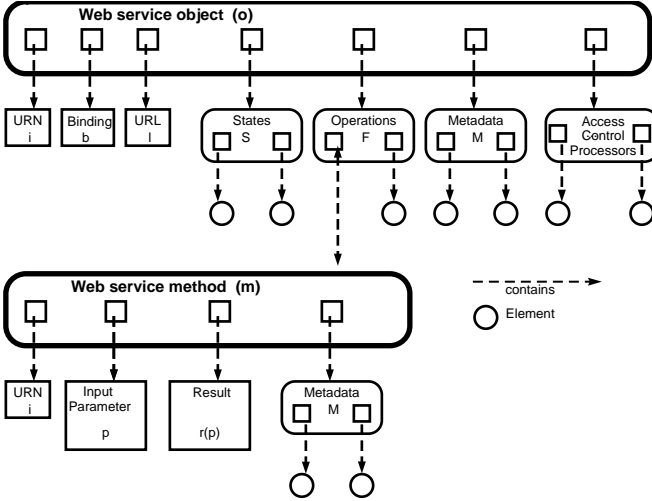


Fig. 2. A Web service method with the associated Web service object (see definition 2).

types to parameters.

To illustrate the usage of types the previously mentioned address verification service may have a Web service method `boolean isValidPostalAddress(String address)`, which given a string over an alphabet that encodes an address, will return the Boolean value `true` or `false` to indicate whether the address is correct or not.

C. Web service collection

A *Web service collection* is used to bundle and organize related Web services objects. It represents an aggregation over a set of (possibly) related Web service objects (see figure 3).

Web service collections provide a convenient way to group related Web service objects, which is useful to facilitate management of authorization specifications and Metadata. Organizing Web service objects into collection hierarchies also allows easier browsing and manipulation compared to what a flat namespace would provide. Hierarchical containment is widely used in current Internet information systems (e.g., Web servers, file systems, URLs) and broadly understood. More formally, we define a Web service as follows:

Definition 3: Web service collection

A Web Service collection is a tuple $c = (i, O, C, M, A)$, where i is a non-empty string over an alphabet Σ^* representing a globally unique identifier (e.g., URN) [8], O is a finite set of (possibly related) Web services objects $o \in O$, C is a finite set of Web service collections (children), and M is a finite set of Metadata providing additional description and semantics for w . A is a finite set of access control processors $a \in A$ that are associated with w and perform authorization decisions for c . O , C , M , or A can be the empty set \emptyset . A Web service is a resource according to RFC 2396 [8].

We allow O to be the empty set \emptyset to be able to refer to a *Web service collection* that can be identified, but has no content yet.

C allows the containment of other Web service collections

as children of c , which results in a hierarchical containment structure (tree) of Web service collections and Web service objects (similar to a file system). C can be the empty set \emptyset .

Associating Metadata with a Web service collection is desirable but not mandatory (M can therefore be the empty set \emptyset).

If A is the empty set \emptyset then there's no access control processor that makes authorization decisions for c . Furthermore, the content of a *Web service collection* is dynamic and may change over the time. Still we can use the identifier i at any time to refer to the same resource with possibly different content representations.

A Web service object may be a member of one or more Web service collections, but this is no requirement. The Web service collection acts as a container for Web service objects and other Web service collections. Note that a Web service collection can have only one parent Web service collection. This ensures a tree structure of the hierarchical containment and simplifies traversal in the namespace.

D. Usage Examples

This section presents some usage scenarios and examples to illustrate how the model can be used to adapt to different needs and environments.

Figure 4 shows how a possible hierarchy of Web service collections, Web service objects, and Web service methods. For instance, we can see that a Web service object can have no methods. Also, a Web service object doesn't need to belong to a Web service collection. Similarly, a Web service collection (C) doesn't have to have any Web services objects. A Web service collection (E) can have children that are either other Web service collections or Web service objects.

As an example, a credit card verification *Web service collection* may be identified using an URI. It could be comprised of many different but related Web service objects o that may span over organizational boundaries and that serve all a particular purpose: One Web service object may

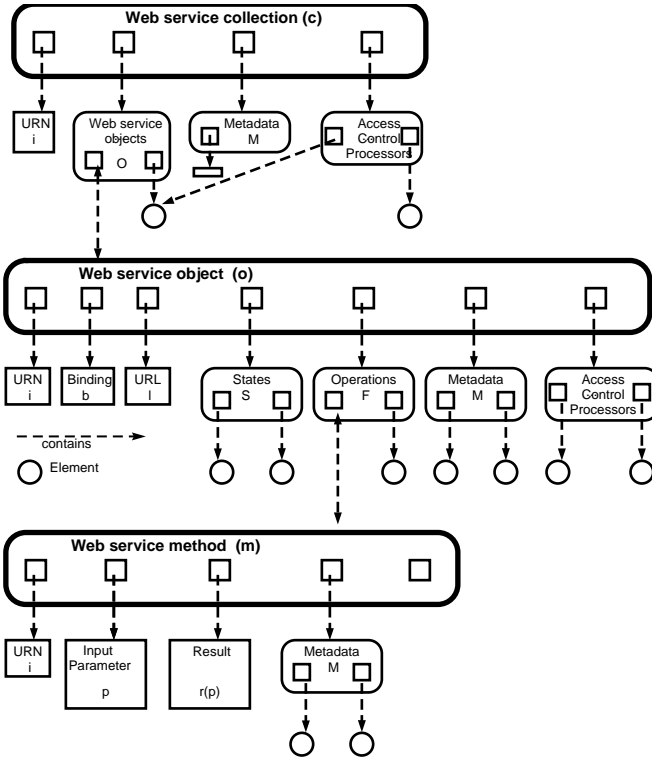


Fig. 3. A Web service collection (see definition 3).

verify a client's address, another one may check the balance of the credit card, and another one eventually would perform a transaction. Additional Metadata (e.g., in form of RDF schemas [14]) could be associated that provides information (e.g., about the Web service provider, contact information, accounting). Decisions for authorizations of incoming requests would be made by an associated access control processor that would either grant or deny that request.

In a simpler scenario an implemented Web service may comprise only one Web service object within one organization with no Metadata and no access control processor associated.

Somewhat more complex would be a Web service collection that comprises several Web service objects. Each of those Web service objects may belong to a different organization. Metadata could describe the semantics of that Web service collection in more detail, and one or more associated access control processors could make authorization decisions for it.

In a more complex example, Web service collections belonging to the same organizational unit but from different departments may be bundled together in a parent Web service collection at a higher organizational level. This might be useful for the specification of authorizations on departmental level, and more general authorizations that are specified at a higher organizational level to propagate further down the hierarchy chain.

Or, all Web services that perform a particular kind of

computation may be bundled into a Web service collection. The overall idea is that Web service objects from any number of sources even spanning organizational boundaries could be combined over the Internet to form dynamic applications for business-to-business commerce. Web service collections do not specify nor force any particular aggregation style or flow of control.

The question arises whether we actually need a Web service collection as an additional abstraction that makes the model more complex, since the notion of Web service collections is currently not supported in any of the major Web service implementations (e.g., MS .NET [3], IBM's WSTK [6]).

A strong motivation for having Web service collections is that it introduces another level of abstraction that might be useful for larger organizations to more efficiently manage authorizations on Web services and trust relationships, i.e. to keep administration efforts lower. Authorizations may be applied to collections, and could be propagated to all its children Web service collections and Web service objects. In general, specifying and applying Metadata or properties to Web service collections allows to exploit the hierarchical tree structure and containment relationship by propagating the information down the hierarchy.

Consider a company running a large amount of different Web services, where Web services spread over different divisions and organizational locations. It would be very time consuming and error prone to update authorizations on a related Web service object manually, if possible at all. In addition, different departments may have different security needs for their own Web services. A Web service collection addresses the need of decentralized management of authorizations that arises from a hierarchical structure and provides a feasible solution for the specification and management of authorizations. In this example, the company may want to bundle all externally available Web services into a Web service collection for security reasons to facilitate authorization management. Another Web service collection could be used to group all internally available Web services objects together.

Furthermore, trust relationships across Web service collections may further simplify authorization management. Trust relationships introduce also additional complexity and it needs to be explored further on how they can be used to enhance the overall model. It might be sufficient for smaller organizations to use only the Web service object abstraction level. The model is flexible to adapt to particular needs.

To sum up there's a trade-off between complexity of the model and practical needs to facilitate administration efforts. Making a Web service collection optional allows organizations to decide on the most appropriate choice to model their individual needs.

E. Example: Microsoft's .NET XML Web Services

The proposed model doesn't favor any particular implementation. It can be used to model many different of the existing Web services implementations, or extend these.

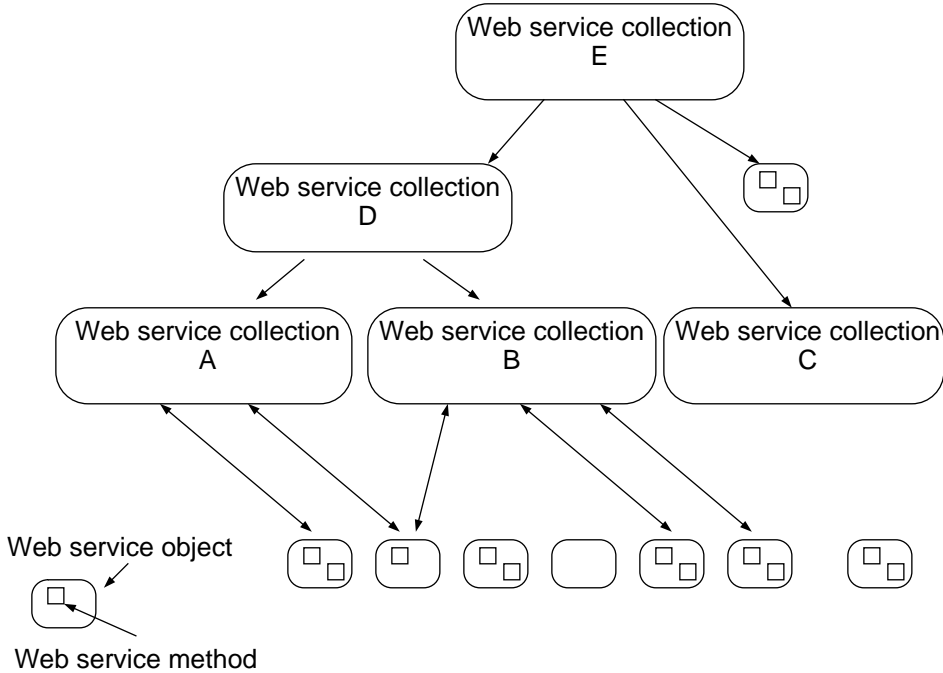


Fig. 4. Example of the Web service model hierarchy

The paper presents two implementations that are both based on a common standardized Web services stack, i.e. XML [15], SOAP[11], WSDL[7], UDDI[12], and shows how the model can be used to represent the particular implementation.

Microsoft's XML Web services architecture based on ASP .NET [3] can be represented in the proposed model as follows:

ASP .NET is based on a common language runtime (CLR). The idea is to develop programming language independent components that all target a common CLR. C# (pronounced C sharp) is an object-oriented programming language that can be used to create XML Web services. Classes in C# have special marked methods, called *WebMethods*, that make a public method accessible as a Web service. ASP .NET uses a special file with the extension ".asmx" that it associates with a class that implements a Web service.

Therefore the ".asmx" file could represent a Web service object $o = (i, b, l, \Sigma, F, M, A)$ of the proposed model, where i is the URN, b (the binding) per default is SOAP over HTTP (alternatively HTTP-GET or HTTP-POST), l would be the location (URL), in that case the URL of the ".asmx" ASP .NET service. The set of states Σ would comprise all internal states of the data members of the C# class. F would comprise the set of *WebMethods* of the ASP .NET service., M would be the Metadata in form of

comments and attributes in the C# code, A would be the empty set (there is no concept of an access control processor).

An ASP .NET *WebMethod* would be mapped to a Web service method of the presented model. According to the definition $m = (i, p, r(p), M)$, i would be the URN based on the used namespace, p a typed parameter list (e.g., `int m(String Hello)`), $r(p)$ the result of the computation (an integer in this example), M may contain some attributes associated to the method.

The notion of a Web service collection is currently not supported in .NET XML Web services.

F. Example: IBM's Web Services Toolkit

IBM provides a toolkit to develop Web services, which is based on Apache SOAP [4] technology and the Java programming language. Any Java class or bean that has public methods can be used to implement a Web service. There is a tight relationship between the Java class and the WSDL that describes the Web service. Instead of marking particular code segments with attributes as in C# (*WebMethod* approach), a WSDL mapping will be created with the assistance of tools (or manually using a text editor) that help to automatically generate WSDL files from existing classes, or create a Java code skeleton based on an existing WSDL file.

Intuitively a Java method would represent a Web service

method in the model and the Java class that contains the method would be the corresponding Web service object. As mentioned earlier WSDL refers to a Web service object simply as a service.

A Web service collection then would be a collection of Java classes or services. However, there is only basic support for a Web service collection (i.e. “services”) in WSDL. WSDL is primarily used to describe Web services and its operations. The Web service collection represents an aggregation of Web services. Emerging standards for composition and aggregation of Web services (e.g., WSFL [16], RDF schemas [14]) may be used to specify and implement Web service collections. Similarly, the same techniques for Web service aggregation may be applied to Microsoft’s .NET XML Web services.

Concluding, the proposed Web services model, along with the presented terminology is flexible enough to model existing Web services implementations. Furthermore, related RPC technologies (e.g., Corba IIOP [17], RPC [18]) can also be represented in the model, which shows its generality and flexibility.

III. FUTURE WORK

Recent informal polls [19] showed that security was the top issue among those considering Web services. When decision makers were asked what are the biggest obstacles to implement Web Services, 45.5% pointed out security and authentication issues. Therefore it would be worthwhile to see how this model can be used to add security functionality, i.e. how an access control processor can be integrated.

In addition, the design spaces of Web service aggregation and operations on Web services need to be explored further. For instance, we can build a composite Web service using a hierarchical containment of different Web service objects. Alternatively, Web services can be invoked in a linear chain and within the chain we can support branching to support parallel execution. Both approaches can be combined, which will lead to interesting architectural aspects. In general, Web service aggregation fosters the reuse of existing Web service components and it would be interesting to investigate on how the proposed model can help to enumerate design choices.

IV. CONCLUSION

The paper introduced a model for components of the service oriented architecture that uses a containment model as a basis to model Web services. A common model for Web services helps to compare different implementations in a principled way, and provides a foundation for system designers to build additional functionality (e.g., security, work-flow) on top. Access control is built in the model to facilitate the integration of security and authorization mechanisms. The proposed model is flexible, extensible, and forces no particular implementation. Examples are shown on how the model can be used in different organizational scenarios. Furthermore, another example illustrated how an existing Web service implementation can be represented using this model.

ACKNOWLEDGEMENTS

I am grateful to Jim Whitehead for his valuable comments and suggestions.

REFERENCES

- [1] W3C Architecture Group, “Web services activity,” <http://www.w3.org/2002/ws/>, last accessed: 3/5/2002.
- [2] Web Services Interoperability Organization, “Resources and guidelines for Web services interoperability,” <http://www.wsi.org/>, last accessed: 3/5/2002.
- [3] Microsoft .NET framework, “.NET framework homepage,” <http://msdn.microsoft.com/netframework/>, last accessed: 3/7/2002.
- [4] Apache SOAP, “Apache SOAP homepage,” <http://xml.apache.org/soap/>, last accessed: 3/7/2002.
- [5] HP e-speak middleware, “HP e-speak homepage,” <http://www.e-speak.hp.com/>, last accessed: 3/7/2002.
- [6] IBM, “IBM’s Web Services Toolkit,” <http://www.alphaworks.ibm.com/tech/webservicestoolkit>, last accessed: 3/7/2002.
- [7] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana, “Web Services Description Language (WSDL) 1.1,” <http://www.w3.org/TR/wsdl>, last accessed: 3/5/2002.
- [8] T. Berners-Lee, R. Fielding, and L. Masinter, “Uniform Resource Identifiers (URI): Generic Syntax,” <http://www.faqs.org/rfcs/rfc2396.html>, August 1998.
- [9] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, and Dave Winer, “Simple Object Access Protocol (SOAP) 1.1,” <http://www.w3.org/TR/SOAP/>, May 2000.
- [10] R. Fielding, J. Gettys, J. Mogul, and H. Frystyk, “Hypertext transfer protocol - HTTP/1.1,” *Network Working Group, Request for Comments*, no. 2068, January 1997.
- [11] Martin Gudgin, Marc Hadley, Jean-Jacques Moreau, and Henrik Frystyk Nielsen, “SOAP Version 1.2 Part 1: Messaging Framework,” <http://www.w3.org/TR/soap12-part1/>, December 2001.
- [12] UDDI, “UDDI homepage,” <http://www.uddi.org/>, last accessed: 3/5/2002.
- [13] W3C, “XML schema,” <http://www.w3.org/XML/Schema>.
- [14] Dan Brickley and R.V. Guha, “Resource description framework (RDF) schema specification 1.0,” <http://www.w3.org/TR/2000/CR-rdf-schema-20000327/>, last accessed: 3/7/2002.
- [15] W3C, “Extensible markup language (XML),” <http://www.w3.org/XML/>.
- [16] Frank Leymann, “Web Services Flow Language (WSFL 1.0),” <http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf>, May 2001.
- [17] William Ruh, Thomas Herron, and Paul Klinker, “IIOP Complete: Understanding CORBA and Middleware Interoperability,” 2000.
- [18] Douglas E. Comer, “Internetworking with TCP/IP, Volume 3, Client/Server Programming and Applications Linux/POSIX Version,” 2000.
- [19] John Fontana, “Top Web services worry: Security,” *Network World*, <http://www.nwfusion.com/news/2002/0121webservices.html?docid=7747>, January 2002.