

Modeling Stateful Resources with Web Services

Version 1.0

01/20/2004

Authors

Jeffrey Frey (IBM) (Editor)

Steve Graham (IBM) (Editor)

Karl Czajkowski (Globus Alliance / USC ISI)

Don Ferguson (IBM)

Ian Foster (Globus Alliance / Argonne National Laboratory) (Editor)

Frank Leymann (IBM)

Martin Nally (IBM)

Tony Storey (IBM)

Steve Tuecke (Globus Alliance / Argonne National Laboratory) (Editor)

William Vambenepe (Hewlett-Packard)

Sanjiva Weerawarana (IBM)

Copyright Notice

© Copyright International Business Machines Corporation and Hewlett-Packard Development Company 2004. All Rights Reserved.

© Copyright The University of Chicago 2004. All Rights Reserved.

Permission to copy and display this "Modeling Stateful Resources with Web Services" whitepaper ("this Whitepaper"), in any medium without fee or royalty is hereby granted, provided that you include the following on ALL copies of this Whitepaper, or portions thereof, that you make:

1. A link or URL to this Whitepaper at this location.
2. This Copyright Notice as shown in this Whitepaper.

THIS WHITEPAPER IS PROVIDED "AS IS," AND IBM AND THE HEWLETT-PACKARD DEVELOPMENT COMPANY (COLLECTIVELY, THE "COMPANIES") MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR TITLE; THAT THE CONTENTS OF THIS WHITEPAPER ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

THE COMPANIES WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR RELATING TO ANY USE OR DISTRIBUTION OF THIS WHITEPAPER.

The names and trademarks of the Companies may NOT be used in any manner, including advertising or publicity pertaining to this Whitepaper or its contents, without specific, written prior permission. Title to copyright in this Whitepaper will at all times remain with the Companies.

No other rights are granted by implication, estoppel or otherwise.

PORTIONS OF THIS MATERIAL WERE PREPARED AS AN ACCOUNT OF WORK SPONSORED BY IBM CORPORATION. NEITHER THE AUTHORS, NOR THE UNITED STATES GOVERNMENT OR ANY AGENCY THEREOF, NOR THE UNIVERSITY OF CHICAGO, NOR IBM, NOR ANY OF THEIR EMPLOYEES OR OFFICERS, NOR ANY OTHER COPYRIGHT HOLDERS OR CONTRIBUTORS, MAKES ANY WARRANTY, EXPRESS OR IMPLIED, OR ASSUMES ANY LEGAL LIABILITY OR RESPONSIBILITY FOR THE ACCURACY, COMPLETENESS, OR USEFULNESS OF ANY INFORMATION, APPARATUS, PRODUCT, OR PROCESS DISCLOSED, OR REPRESENTS THAT ITS USE WOULD NOT INFRINGE PRIVATELY OWNED RIGHTS. REFERENCE HEREIN TO ANY SPECIFIC COMMERCIAL PRODUCT, PROCESS, OR SERVICE BY TRADE NAME, TRADEMARK, MANUFACTURER, OR OTHERWISE, DOES NOT NECESSARILY CONSTITUTE OR IMPLY ITS ENDORSEMENT, RECOMMENDATION, OR FAVORING BY IBM, THE UNITED STATES GOVERNMENT OR ANY AGENCY THEREOF OR ANY OTHER COPYRIGHT HOLDERS OR CONTRIBUTORS. THE VIEW AND OPINIONS OF AUTHORS EXPRESSED HEREIN DO NOT NECESSARILY STATE OR REFLECT THOSE OF IBM, THE UNITED STATES GOVERNMENT OR ANY AGENCY THEREOF, OR THE ENTITY BY WHICH AN AUTHOR MAY BE EMPLOYED.

This manuscript has been created in part by the University of Chicago as Operator of Argonne National Laboratory ("Argonne") under Contract No. W-31-109-ENG-38 with the U.S. Department of Energy. The U.S. Government retains for itself, and others acting on its behalf, a paid-up, nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government.

Abstract

The Web services architecture has been broadly accepted as a means of structuring interactions among distributed software services. Further standardization is now required to facilitate additional interoperability among services. One important area in which further standardization is required concerns interactions with stateful resources. In this paper, we address the constructs used to enable Web services to access state in a consistent and interoperable manner. We introduce the *WS-Resource approach* to declaring and implementing the association between a Web service and one or more named, typed state components. In this approach, we model state as *stateful resources* and codify the relationship between Web services and stateful resources in terms of the *implied resource pattern*, a set of conventions

on Web services technologies, in particular WS-Addressing. When a stateful resource participates in the implied resource pattern, we refer to it as a *WS-Resource*. We describe the means by which a WS-Resource is defined and associated with the description of a Web service interface. We also describe an approach for making the properties of a WS-Resource accessible through a Web service interface, and for managing a WS-Resource's lifetime.

Status

This whitepaper is an initial draft release and is provided for review and evaluation only. The Companies hope to solicit your contributions and suggestions in the near future. The Companies make no warranties or representations regarding the whitepaper in any manner whatsoever.

Table of Contents

1	Introduction.....	5
2	Web Services Background.....	6
2.1	What is a Web Service?	6
2.2	Web Service Environments	7
2.3	A Brief Taxonomy of State and Services	8
2.4	Stateless Implementations, Stateful Interfaces.....	9
3	Modeling State in Web Services	9
3.1	Modeling State: Stateful Resources	10
3.2	The Implied Resource Pattern	11
3.3	WS-Resource Use of WS-Addressing	11
3.4	WS-Resource Relationship Cardinality	14
3.5	WS-Resource Encapsulation	14
4	WS-Resource Lifecycle and Identity.....	15
4.1	WS-Resource Creation	15
4.2	WS-Resource Identity.....	15
4.3	WS-Resource Destruction.....	17
5	WS-Resource Properties	17
5.1	WS-Resource Properties Document.....	18
5.2	WS-Resource Property Composition.....	19
5.3	Accessing WS-Resource Property Values	20
6	WS-Resource and ACID Properties	20
7	WS-Resource Security.....	21
8	Conclusions	21
9	Acknowledgements	22
10	References	22

1 Introduction

The Web services architecture [WS-ARCH] defines a service-oriented distributed computing model in which services interact by exchanging XML documents. The basic elements of the Web services architecture define the syntax for information exchange. Various efforts are now underway to augment this base architecture with additional conventions so that interacting services can accomplish more sophisticated behaviors such as authentication, transactions, and reliable messaging [Web Services] in standard ways.

We introduce here a set of conventions intended to formalize interactions with *state*. The motivation for these new conventions is that while Web service *implementations* are typically stateless, their *interfaces* must frequently allow for the manipulation of state, i.e., data values that persist across, and evolve as a result of, Web service interactions. For example, an online airline reservation system must maintain state concerning flight status, reservations made by specific customers, and the system itself: its current location, load, and performance. Web service interfaces that allow requestors to query flight status, make reservations, change reservation status, and manage the reservation system must necessarily provide access to this state.

Web services successfully implement applications that manage state today. Nevertheless, it is desirable to define conventions for stateful services so that requestors can discover, introspect on, and interact with those underlying resources in *standard* and *interoperable* ways.

In what we term the *WS-Resource approach*, we model state as *stateful resources* and codify the relationship between Web services and stateful resources in terms of the *implied resource pattern*, a set of conventions on Web services technologies, particularly XML, WSDL, and WS-Addressing [WS-Addressing]. When a stateful resource participates in the implied resource pattern, we refer to it as a *WS-Resource*. We describe the means by which a WS-Resource is defined and associated with the description of a Web service interface. We also describe an approach for making the properties of a WS-Resource accessible through a Web service interface, and for managing and reasoning about a WS-Resource's lifetime.

This paper contributes to an ongoing debate within the Web services community concerning whether and how Web services should allow for the representation of state. In this debate, one view is that "Web services ... have no notion of state" [Vogels] and "a service [must be] stateless" [Parastatidis], while others, including ourselves, have argued that the critical role that state plays in distributed computing requires that it be addressed within the Web services architecture [Physiology]. The WS-Resource construct may help reconcile these two positions, by showing how the relationship between Web services and state can be formalized in a straightforward manner that builds on other Web services specifications.

We are concerned in this paper with the concepts and constructs that underlie the WS-Resource approach, not its rendering in terms of specific Web services message exchanges. We propose elsewhere a specific rendering, in the form of a set of specifications called the WS-Resource Framework [WSRF].

The WS-Resource approach is inspired by the work of the Global Grid Forum's Open Grid Services Infrastructure (OGSI) Working Group [Physiology, OGSI-Spec]. We

discuss the relationship between the WS-Resource approach and framework and OGSi elsewhere [OGSI-Refactor].

2 Web Services Background

Before we can define the means by which Web services may be associated with stateful resources, we need to clarify a few terms and concepts.

2.1 What is a Web Service?

The term *Web services* emerged in the year 2000 with the introduction of technologies such as SOAP, WSDL, and UDDI. Contemporaneously, the term *service oriented architecture* (SOA) [Tao] was coined to describe the overall approach of building loosely coupled distributed systems with minimal shared understanding among system components. Much writing and some practice has since increased understanding of these concepts by the community of information technology practitioners.

Whereas the individual component technologies, such as SOAP, WSDL and UDDI are fairly well defined, a universally accepted definition of the term *Web service* remains elusive. The W3C Web services Architecture working group provides the following definition [WS-Arch]:

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

A service-oriented architecture defines a distributed system wherein agents, known as services, coordinate by sending messages. Quoting [WS-Arch] once again:

An SOA is a specific type of distributed system in which the agents are "services." [A] service is a software agent that performs some well-defined operation (i.e., "provides a service") and can be invoked outside of the context of a larger application. That is, while a service might be implemented by exposing a feature of a larger application ... the users of that server need be concerned only with the interface description of the service. "[S]ervices" have a network-addressable interface and communicate via standard protocols and data formats.

It is tempting to interpret the clause "users of that server need be concerned only with the interface description of the service" as implying that a service's behavior is defined solely by the message exchanges supported by the service. But service interface definitions often *imply* the existence of a stateful resources that are used and manipulated in the processing of a Web service request message. For example, the airline reservation system might support three messages, as follows.

- *getReservation*, which returns an XML document describing the reservation.
- *addFlightSegment*, which adds a new flight to the reservation.
- *removeFlightSegment*, which removes a segment from the trip.

This interface implies that the service manages a set of documents describing reservations. Programmers may also infer a reservation identifier from the messages declared in the service's interface. The central tenet of this paper is that it is desirable to represent such relationships between a Web service and state explicitly and in a standard manner, instead of relying on intuitive inferences. We argue that such explicit representation and standardization enhances service interoperability, simplifies the definition of new service interfaces, and enables more powerful discovery, management, and development tools.

2.2 Web Service Environments

Operationally, there are several important facets of a Web service that require further description. These components are illustrated in Figure 1 and explained below.

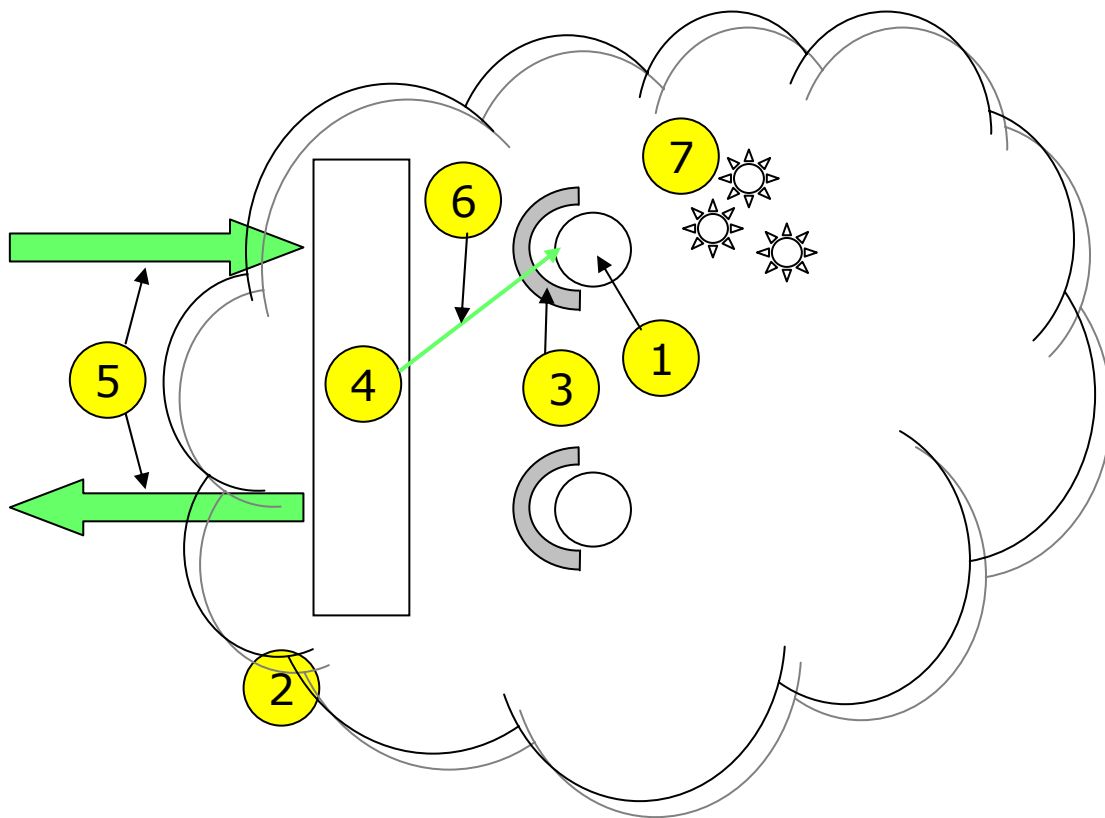


Figure 1 Facets related to a Web service

In Figure 1 a Web service (labeled 1 in the figure) is a software component that performs some function, such as posting a purchase order. Web services may provide operations that access or manipulate the state held in other resources within the system. A Web service is a component deployed within some runtime environment (2), for example a Web application server such as IBM's WebSphere or JBoss. The runtime environment is responsible for executing the code of the Web service and for dispatching messages to the Web service. The environment may also

provide other qualities of service to the Web service, such as security and transactions.

A Web service's interface (3), described by a Web service description language such as WSDL [WSDL 1.1], defines the Web service's capabilities in terms of a collection of operations that may be invoked by other entities (called service requestors) in a distributed system. Each operation is described in terms of message exchanges that define both the format of the message used to invoke an operation and the message format(s) of possible response message(s), including fault messages.

The runtime environment provides a message processing facility (4) that can receive messages (5) from requestors. This component may support one or more network transport protocols, such as HTTP, SMTP, or IIOP. The term *endpoint* is often used for this facet of the runtime environment, as this component is made available to the distributed computing fabric at a particular network address. Other responsibilities of the endpoint component include translating the message envelope into a format understandable by the service (for example, converting on-the-wire XML into a collection of Java objects) and (6) dispatching the message to the target service implementation identified by the address (URL) and other components of the message. Note that whereas Web services are created within (or, as we sometimes say, deployed in) a runtime environment at a particular endpoint address (i.e., the network identity of the message processing facility of the runtime environment), each Web service is itself uniquely identified by an address that combines the endpoint address to which it is deployed plus some additional identity component specific to that Web service.

The Web service implementation is responsible for receiving the message, processing the message—potentially interacting with other services and stateful resources (7)—and, if appropriate for the message exchange, formatting and sending a response message. Many Web services themselves play the role of a service requestor, initiating message exchanges with other Web services.

2.3 A Brief Taxonomy of State and Services

A thorough taxonomy of interface, state, and instances is beyond the scope of this paper. However, to provide context for the material that follows, we provide the following brief overview of possible associations of state with an interface.

- 1) A *truly stateless service* implements message exchanges with no access or use of information not contained in the input message. A simple example is a service that compresses and decompresses documents, where the documents are provided in the message exchanges with the service.
- 2) A *conversational service* implements a series of operations such that the result of one operation depends on a prior operation and/or prepares for a subsequent operation. The service uses each message in a logical stream of messages to determine the processing behavior of the service. The behavior of a given operation is based on processing preceding messages in the logical sequence. Many interactive Web sites implement this pattern through HTTP sessions and cookies.

- 3) A *stateless service that acts upon stateful resources* provides access to, or manipulates a set of logical stateful resources (documents) based on messages it sends and receives.

We are concerned in this paper with the third model. We believe that approaches based on context/headers, WS-Coordination, and WS-Policy provide a model for the second model.

2.4 Stateless Implementations, Stateful Interfaces

When we talk in the third model about a *stateless service* we mean a service whose *implementation* maintains no dynamic state, i.e., state for which the service is responsible between message exchanges with its requestors. Statelessness tends to enhance reliability and scalability: a stateless Web service can be restarted following failure without concern for its history of prior interactions, and new copies of a stateless Web service can be created (and subsequently destroyed) in response to changing load. Thus, statelessness is generally viewed as good engineering practice for Web service implementations.

A consequence of statelessness is that any dynamic state needed for a given message-exchange execution (its *execution context*) must be either:

- provided *explicitly* within the request message, whether directly by-value or indirectly by-name or by-reference, or
- maintained *implicitly* within other system components with which the Web service can interact.

Of course, the Web service may also maintain *static* state (e.g., preconfigured references to other system components) within its implementation.

The third model's characterization of a *stateless service that acts upon stateful resources* acknowledges that a "stateless" Web service will frequently interact with, and cause updates to, dynamic state that is maintained in other system components, such as a database, with the identity of the state element(s) to be either passed in the request message or maintained as static data by the Web service. The interface offered by such a Web service is clearly stateful, in the sense that its behavior is defined with respect to underlying state.

3 Modeling State in Web Services

We have pointed out that a Web service *implementation* can normally be viewed as a stateless message processor, but that the message exchanges that it implements (as defined by its *interface*) are frequently intended to enable access to, and/or update of, state maintained by other system components, whether database, file systems, or other entities.

Given the vital role that access to state plays in many Web service interfaces, it is important to identify and standardize the patterns by which state is represented and manipulated, so as to facilitate the construction and use of interoperable services.

To this end, we introduce an approach to modeling stateful resources in a Web services framework based on a construct that we call a *WS-Resource*. More specifically, we define the means by which:

- a stateful resource is used as the *data context* for the execution of Web service message exchanges (this section);
- stateful resources can be created, assigned an identity, and destroyed (Section 4); and
- the type definition of a stateful resource can be associated with the interface description of a Web service to enable well-formed queries against the resource via its Web service interface, and the state of the stateful resource can be queried and modified via Web service message exchanges (Section 5).

3.1 Modeling State: Stateful Resources

The term *state* is vague and can in principle encompass many different aspects of a computer system, from the value stored in a specific database record to the seek time or even temperature of the disk drive. We focus on what we call a *stateful resource*, which we define to be:

- a specific set of state data expressible as an XML document that defines the type of the resource;
- having a well-defined identity and lifecycle; and
- known to, and acted upon, by one or more Web services.

Examples of system components that may be modeled as stateful resources are files in a file system, rows in a relational database, and encapsulated objects such as Entity Enterprise Java beans. A stateful resource can also be a collection or group of other stateful resources.

Note that this definition concerns how a stateful resource is modeled, not how it is implemented or represented. A specific resource's state may be implemented as an actual XML document that is stored in memory, in the file system, in a database, or in some XML Repository. Alternatively, the same resource may be implemented as a logical projection over data constructed or composed *dynamically* from programming language objects (such as a J2EE EJB Entity Bean) or from data returned by executing a command on a private communications channel to a traditional procedural application or data system.

Multiple independent instances of a given resource type may be created and destroyed. As we describe in Section 4, an instance of a stateful resource may be created via a Web service referred to as a *resource factory*.

As we describe in Section 5, the type of a stateful resource is defined by a single XML Global Element Declaration (GED) in a given namespace. This GED defines the type of the root element of the resource's XML document and hence the type of the stateful resource itself.

When a stateful resource instance is created, it is assigned an identity by the entity that created it. Applications using the resource may assign the resource additional identities (aliases). The identity of a stateful resource instance may be consumed by one or more Web service implementations as stateful resource-related context under which the execution of a given message exchange is performed. The use of resource identity as part of a message execution context is discussed in the next section.

3.2 The Implied Resource Pattern

Having defined how we can model elements of state as stateful resources, we now turn to the question of how stateful resources are referred to by a Web service's clients. We define the term *implied resource pattern* to describe a specific kind of relationship between a Web service and one or more stateful resources.

The *implied resource pattern* refers to the mechanisms used to associate a stateful resource with the execution of message exchanges implemented by a Web service.

- The term *implied* is used because the stateful resource associated with a given message exchange is treated as implicit execution context for the message request. By implicit, we mean to say that the requestor does not provide the identity of the resource as an explicit parameter in the body of the request message. Instead, the context used to designate the implied stateful resource is encapsulated in the WS-Addressing endpoint reference used to address the target Web service at its endpoint.
- We use the term *pattern* to indicate that the relationship between Web services and stateful resources is codified by a set of conventions on existing Web services technologies, in particular XML, WSDL, and WS-Addressing [WS-Addressing].

A WS-Addressing endpoint reference is an XML serialization of a network-wide *pointer* to a Web service. This pointer may be returned as a result of a Web service message request to a factory to *create* a new resource or, alternatively, from the evaluation of a search query on a registry of resources, or as a result of some application-specific Web service request.

WS-Addressing standardizes the endpoint reference construct used to represent the address of a Web service deployed at a given network endpoint. An endpoint reference may contain, in addition to the endpoint address of the Web service, other metadata associated with the Web service such as service description information and *reference properties*, which help to define a contextual use of the endpoint reference. The reference properties of the endpoint reference play an important role in the implied resource pattern.

Note that other patterns for enabling access to stateful resources are possible. For example, a Web service could maintain the resource identity as static service state, thus obviating the need to pass that identity in the WS-Addressing endpoint reference. However, we do not recommend the use of this pattern, as it implies a one-to-one mapping from Web service endpoints to stateful resources and thus a need for a unique Web service endpoint for each stateful resource. Instead, we maintain an architectural distinction between the identity of a WS-Resource and its corresponding Web service, in order to support a strict separation of the Web service from stateful resource implementations and thus to permit a one-to-many mapping from Web service endpoints to stateful resources.

3.3 WS-Resource Use of WS-Addressing

When a stateful resource participates in the implied resource pattern, we refer to it as a *WS-Resource*.

Let us examine the WS-Addressing-related conventions used by the implied resource pattern. We show in Figure 2 a WS-Addressing endpoint reference conformant to the conventions of the implied resource pattern.

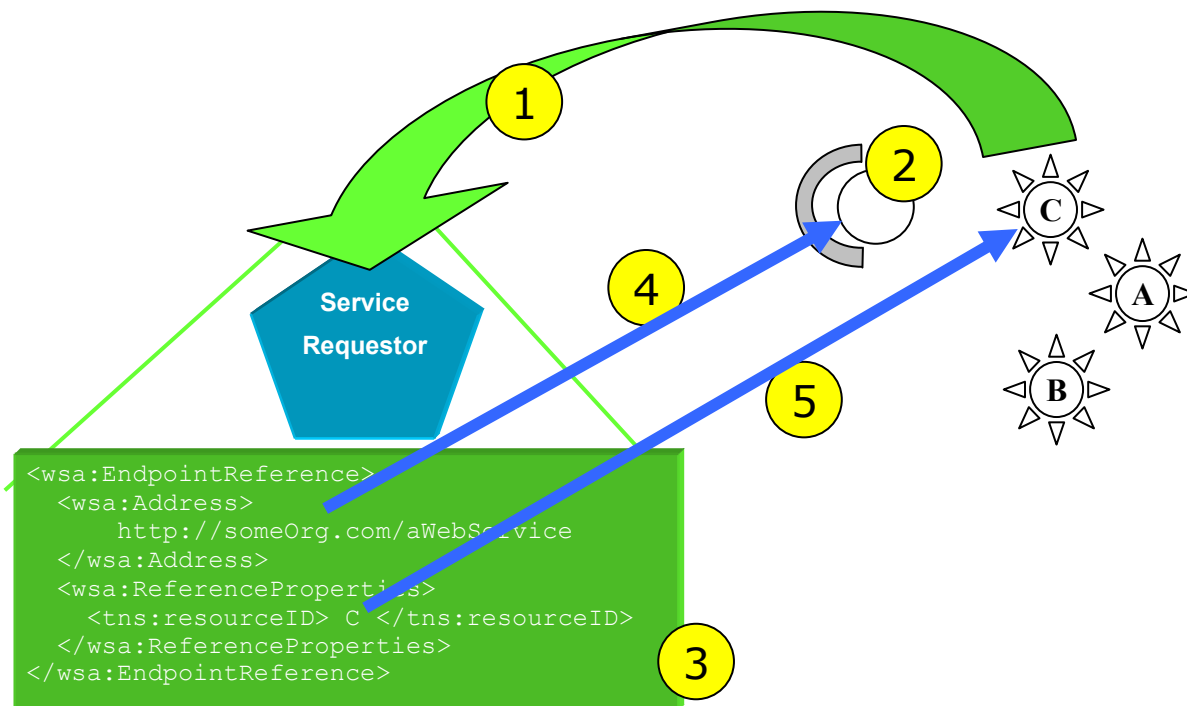


Figure 2 - An Endpoint Reference containing a WS-Resource context

An endpoint reference (labeled 1 in Figure 2) is returned to the requestor in response to some request sent to the Web service (2). Let us assume that the processing of the request resulted in the creation of the WS-Resource "C." We say that the Web service represents an *explicit WS-Resource factory*. The response message from a WS-Resource factory contains the endpoint reference of a Web service capable of acting upon the newly created WS-Resource. This endpoint reference contains information that expresses the implied resource pattern relationship between the Web service and the newly created WS-Resource.

The endpoint reference (3) contains two important components:

- The `wsa:Address` component (4) refers to the network transport-specific address of the Web service (often a URL in the case of HTTP-based transports). This is the same address that would appear within a port element in a WSDL description of the Web service.
- The `wsa:ReferenceProperties` component contains an XML serialization of the WS-Resource identity, as understood by the Web service addressed by the endpoint reference. The WS-Resource identity represents the WS-Resource to be used in the execution of the request message (5). The set of reference properties used to hold the WS-Resource identity within the endpoint reference is referred to as the *WS-Resource context*. An endpoint reference

containing a WS-Resource context is a *WS-Resource-qualified endpoint reference*.

The XML serialization of the WS-Resource context uses a service-specific XML element to represent the WS-Resource context information that is opaque to the service requestor. The service requestor's applications should not examine or attempt to interpret the contents of the WS-Resource context. The WS-Resource context is meaningful only to the Web service, and is used by the Web service in an implementation-specific way to identify the WS-Resource needed for the execution of the request message.

The WS-Resource context must identify a unique WS-Resource that is to be used in the execution of the request message. There is no requirement that the identity of the WS-Resource be universally unique, but it must be possible for the Web service to use the WS-Resource identity to identify the intended WS-Resource unambiguously. In other words, the scope of the WS-Resource identity contained in the WS-Resource context must be unique within the scope of the Web service and may be unique beyond the scope of the Web service. In addition, multiple identifiers within the scope of a Web service may refer to the same WS-Resource.

From the point of view of the service requestor, the endpoint reference represents a *pointer* to the Web service that has been further constrained to execute its message exchanges within the context of a specific WS-Resource. The service requestor must understand that the use of the endpoint reference represents a WS-Resource-based, contextual use of the Web service. In other words, the service requestor must recognize that the endpoint reference is a WS-Resource-qualified endpoint reference. The use of a WS-Resource-qualified endpoint reference is illustrated in Figure 3.

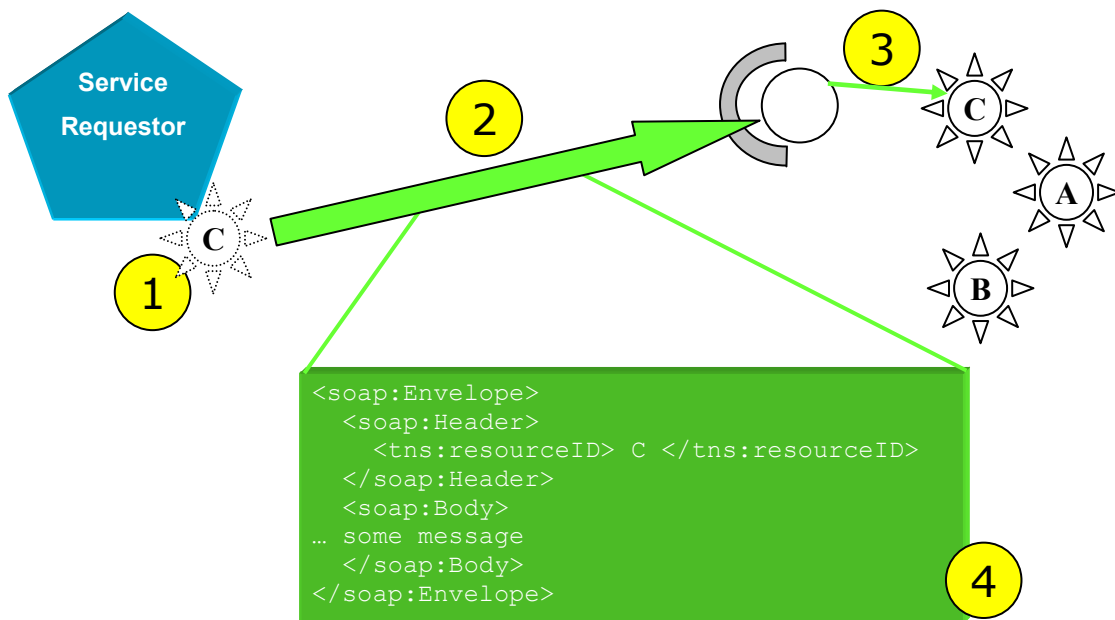


Figure 3 - Using a WS-Resource-qualified endpoint reference

The service requestor's applications would use the endpoint reference (labeled 1 in Figure 3) to send messages (2) to the identified Web service (3). The request messages directed to the service must contain the WS-Resource context obtained from the reference properties component of the WS-Resource's endpoint reference.

Note that the ReferenceProperties of a WS-Addressing message are processed in a binding-specific way. The WS-Addressing specification mandates that the ReferenceProperties element of the endpoint reference must appear as part of any message sent to the Web service identified by the endpoint reference. Each type of WSDL binding must declare how child elements of the ReferenceProperties element must appear in messages using that binding. For example, WS-Addressing specifies that ReferenceProperty elements must appear as SOAP header elements in the message. In Figure 3, the component labeled (4) illustrates the use of a SOAP header to propagate the WS-Resource context, in this case representing the identity of the WS-Resource named "C." The Web service (3) then extracts the WS-Resource context from the SOAP message and uses it to locate the WS-Resource needed for the execution of the request message.

3.4 WS-Resource Relationship Cardinality

We need to further refine the relationship between Web services and WS-Resources. In particular, we need to describe the cardinality of the relationship between the WS-Resource and the Web service at both the type and instance levels.

A Web service can execute message exchanges against zero or more WS-Resources of a given type. In a typical situation, a single Web service at a particular endpoint is associated with several individual WS-Resources. In some circumstances, the number of WS-Resources acted upon by a single Web service could be extremely large, as for example in the case of a Web service interface to a file system that models each file as a distinct WS-Resource.

At the type level, a WSDL 1.1 portType, defining the interface to a Web service, can be associated with at most one type of WS-Resource. The standard means for forming this association is described below. Any Web service that implements this portType is by definition a Web service associated with that type of resource.

One type of WS-Resource can be associated with many portTypes. This one-to-many relationship at the type level allows an individual resource to be associated with multiple Web services, each of which implements a different interface.

At the instance level, a WS-Resource can be associated with one or more Web services. The one-to-many relationship between a WS-Resource and a Web service can be exploited to allow multiple network protocol or network endpoints to process messages for the WS-Resource, or to allow different Web services interfaces to categorize and subset messages that act upon the WS-Resource.

3.5 WS-Resource Encapsulation

The benefits of data encapsulation are well known. Strict encapsulation guarantees that encapsulated data can only be accessed through well defined operations. These operations provide a control point, implementing data-related policy enforcement leading to increased data consistency, control, and integrity. Data encapsulation facilitates the use of data without the user having to understand the details of the

data implementation, thereby reducing the external dependencies on the implementation and providing increased design flexibility.

We introduce the implied resource pattern to facilitate Web service encapsulation of stateful resources. All access to the state of a WS-Resource can be accomplished with message exchanges implemented by the associated Web services. An additional form of encapsulation is used to express the association of a WS-Resource with a Web service. The WS-Resource context is managed by the Web service itself, not by the service requestor. The use of the WS-Resource context contained within the endpoint reference eliminates the need for the service requestor to have specific knowledge of the identity and location of the WS-Resource encapsulated by the Web service.

4 WS-Resource Lifecycle and Identity

The lifetime of a WS-Resource is defined as the period between its creation and its destruction. The actual mechanisms by which a specific WS-Resource is created and destroyed are implementation-specific. However, we do address the following three aspects of the WS-Resource lifecycle in the three subsections that follow:

- 1) WS-Resource creation through the use of a WS-Resource factory,
- 2) the assignment and use of WS-Resource identity, and
- 3) the destruction of a WS-Resource.

4.1 WS-Resource Creation

A WS-Resource may be created by some out-of-band mechanism, or alternatively (as we discuss here) through the use of a WS-Resource factory. A *WS-Resource factory* is any Web service capable of bringing a WS-Resource into existence and assigning the new WS-Resource an identity. The response message of a WS-Resource factory operation typically contains a WS-Resource-qualified endpoint reference containing a WS-Resource context that refers to the new WS-Resource, though a factory may convey the reference to the new WS-Resource through other means such as placing the WS-Resource-qualified endpoint reference into a registry for later retrieval.

Note that there may be many types of Web services (e.g., resource registries) that return WS-Resource-qualified endpoint references in their response messages. However, unless the Web service message exchange resulted in the actual creation of the WS-Resource referred to in the returned WS-Resource-qualified endpoint reference, the message exchange is not considered a WS-Resource factory operation.

Note also that what we refer to here as a WS-Resource factory is a use pattern for Web services, not a single standard operation. This use pattern may be encoded in a variety of different Web service operations that may, for example, create one or many WS-Resources.

4.2 WS-Resource Identity

We describe and contrast the role and use of WS-Resource identity from two perspectives:

1. the Web service with which a WS-Resource is associated, and
2. a service requestor to whom an endpoint reference to a WS-Resource is provided.

Recall that, as stated in Section 3.3, each WS-Resource has at least one form of identity that identifies that unique WS-Resource within the context of the Web service that provides access to that WS-Resource. This identity is not the same as the identity of that Web service, but the Web service can construct an address for an associated WS-Resource by including the WS-Resource's local identity information into the reference properties portion of a WS-Addressing endpoint reference. That endpoint reference is then said to be *WS-Resource qualified*. A WS-Resource-qualified endpoint reference can then be made available to other entities in a distributed system, which can subsequently use that endpoint reference to direct requests to the WS-Resource.

For a Web service with which a WS-Resource is associated, the WS-Resource identity information contained as context in a request message is meaningful. The Web service implementation understands the content of the implementation-dependent WS-Resource context, and can use that information to identify the WS-Resource to be used in the message execution.

In contrast, a service requestor that obtains access to a WS-Resource-qualified endpoint reference should not examine or attempt to interpret the contents of the WS-Resource context. Even an attempt by the service requestor to compare the contents of two WS-Resource contexts is considered an invalid use of the WS-Resource context. From the perspective of the service requestor, the content of the WS-Resource context is opaque.

So how would a service requestor reason about the identity of a WS-Resource? The short answer is that the semantic meaning of the WS-Resource identity, and the means by which it is defined and exposed to a service requestor, is Web service implementation dependent. At the current time, there are no adopted Web service specifications that provide for the definition of stateful resource identity. Nor is there any definition of the means by which the identity of a stateful resource is obtained by a service requestor.

Whether or not the identity of a WS-Resource is exposed to a service requestor is a property of a particular Web service design. However, we believe many Web services will provide the ability to retrieve the identity of the WS-Resource. The identity should be a portable, namespace-scoped value. Portability is important as it allows one application to pass the identity to another. Namespace scoping is important as it allows for disambiguation of multiple identities that may originate from different sources.

We envision that a common approach for exposing the identity of the WS-Resource will be to treat the identity as one or more resource state properties expressed in the WS-Resource properties document. This approach would allow a service requestor to direct a query against the document, targeting the properties understood to represent the identity of the WS-Resource. If the identity is exposed as one or more WS-Resource properties, the Web service should ensure read-only access to those properties. Typically, it would be invalid to allow a service requestor to change the identity of a WS-Resource.

As another option, the Web service may implement application-specific message exchanges intended to provide access to the identity of the WS-Resource. We anticipate that many applications will recognize the need to introduce message exchanges related to WS-Resource identity. Some such exchanges may provide for retrieving identity, and some may provide WS-Resource comparison and equality checks.

4.3 WS-Resource Destruction

A requestor that sends a message request to a WS-Resource factory that causes the creation of a new WS-Resource will typically only be interested in that new WS-Resource for some finite period. After that time, it should be possible to destroy the WS-Resource so that its associated system resources can be reclaimed.

The definition of specific interfaces used to support the destruction of WS-Resources is beyond the scope of this paper. However, we can describe general requirements.

A service requestor that wishes to cause the destruction of a WS-Resource uses the appropriate WS-Resource-qualified endpoint reference to send a destroy request message to the Web service identified by the endpoint reference. The WS-Resource context within the endpoint reference is used to identify the WS-Resource to be destroyed. The receipt of the response to the destroy request message represents a point of synchronism between the service requestor and the Web service receiving the destroy request message. Upon receipt of the response message, any further message exchanges with the service using a WS-Resource context representing the destroyed WS-Resource must result in a fault message indicating that the WS-Resource was unknown, absent any other fault conditions that may take precedence.

We can also define message exchanges for establishing and renewing leases on WS-Resources, so as to provide for implicit destruction in situations where a client cannot or will not destroy a WS-Resource explicitly.

5 WS-Resource Properties

We now discuss the means by which the type and values of a WS-Resource's state can be viewed and modified by service requestors through a Web services interface. The key ideas are as follows.

- The WS-Resource state is expressed as an XML *resource property document* defined using XML schema.
- Service requestors may determine a WS-Resource's type by retrieving the XML schema definition via standard means.
- Service requestors may use Web services message exchanges to read, modify, and query the XML document representing the WS-Resource's state.

We use the term *resource property* to refer to an individual component of a WS-Resource's state. We call the XML document describing the type of a WS-Resource a *WS-Resource properties* document. Each resource property is represented as an XML element within the WS-Resource properties document.

5.1 WS-Resource Properties Document

The WS-Resource properties document acts as a view on, or projection of, the actual state of the WS-Resource. The document serves to define the structure upon which service-requestor-initiated query and update messages can be directed. Any operation that manipulates a resource property via the WS-Resource properties document must be reflected in the actual implementation of the WS-Resource's state.

The WS-Resource properties document is expressed using XML Schema. Specifically, the WS-Resource properties document is expressed as an XML global element declaration (GED) in some XML namespace. For example, consider the WS-Resource "C" mentioned in previous sections. If the state of "C" comprises three components, named p1, p2, and p3, then its resource properties document, named "ExampleResourceProperties," might be defined as follows.

```
<xs:schema
  targetNamespace="http://example.com/ResourcePropertiesExample"
  xmlns:tns="http://example.com/ResourcePropertiesExample"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  ...
  ... >

  <xs:element name="p1" type= ... />
  <xs:element name="p2" type= .../>
  <xs:element name="p3" type= ... />

  <xs:element name="ExampleResourceProperties">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="tns:p1" />
        <xs:element ref="tns:p2" />
        <xs:element ref="tns:p3" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  ...
</xs:schema>
```

Service requestors may obtain and examine this XML schema definition of the WS-Resource properties document, which represents the type of WS-Resource "C," by various means, including message exchanges defined by WS-MetaDataExchange [WS-MetaDataExchange].

But how did the service requestor know that the GED named "ExampleResourceProperties" defines the WS-Resource properties document associated with the Web service? The WS-Resource properties document declaration for the Web service occurs in the WSDL definition of the Web service interface. The WS-Resource properties document declaration is associated with the WSDL portType definition via the use of a standard attribute, resourceProperties, as in the following example.

```
<wsdl:definitions
  targetNamespace="http://example.com/ResourcePropertiesExample"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:wsrp=
```

```

    "http://www.ibm.com/xmlns/stdwip/web-services/ws-resourceProperties"
    xmlns:tns="http://example.com/ResourcePropertiesExample"
...>
...
<wsdl:types>
  <xs:schema>
    <xs:import
      namespace="http://example.com/ResourcePropertiesExample"
      schemaLocation="..." />
    </xs:schema>
  </wsdl:types>
...
<wsdl:portType name="SomePortTypeName"
  wsrp:resourceProperties="tns:ExampleResourceProperties" >
  <operation name="...
...
</wsdl:portType>
...
</wsdl:definitions>

```

This association effectively defines the WS-Resource type of all WS-Resources supported in the implied resource pattern by any Web service that implements this portType.

5.2 WS-Resource Property Composition

Web services allow us to construct a new interface from several existing interfaces via a process of composition. In WSDL 1.1, this composition must be achieved by a copy-and-paste of the operations defined in the constituent portTypes used in the composition. For example, the operations defined in an example portType “foo” can be combined with the operations defined in various standards and specifications to yield a final, complete set of message exchanges to be implemented by a Web service.

In addition to operation composition, the designer may also aggregate the WS-Resource properties defined in the WS-Resource properties documents of the various constituent portTypes to yield the final, complete WS-Resource property document declared with the final composed portType. This WS-Resource properties document composition may be accomplished by adding additional XML element declarations, using the `xs:ref` attribute, as demonstrated in the following example.

```

<xs:element name="ExampleResourceProperties">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="tns:p1" />
      <xs:element ref="tns:p2" />
      <xs:element ref="tns:p3" />

      <xs:element ref="xxxx:SomeAdditionalResourceProperty"
        xmlns:xxxx= ... />

    </xs:sequence>
  </xs:complexType>
</xs:element>

```

This WS-Resource properties document was constructed by combining the resource property elements of the WS-Resource properties document for WS-Resource "C" with a resource property element (SomeAdditionalResourceProperty) defined in some other namespace.

5.3 Accessing WS-Resource Property Values

The state of a WS-Resource, i.e., the values of resource properties exposed in the WS-Resource's resource properties document, can be read, modified, and queried by using standard Web services messages. We outline these messages that might be used for this purpose here; details are beyond the scope of this paper.

The base functionality is to retrieve the value of a single resource property using a simple Web services request/response message exchange. The request message identifies the WS-Resource using a WS-Resource-qualified endpoint reference as described previously and identifies the resource property by the qualified name of its GED. A slightly more sophisticated variant of this retrieval function can allow the retrieval of the value of multiple resource properties with a single request/response message exchange. The Web service responds with a message containing the values of the requested WS-Resource properties.

It is also possible to use a standard message exchange to execute an arbitrary XPath expression against the resource properties document. Various query expression types may be used, for example, to support resource discovery based on the current values of a WS-Resource's state.

We also envisage an operation that would allow the values of resource properties to be inserted, updated, and deleted through the view provided by WS-Resource's resource property document.

6 WS-Resource and ACID Properties

The acronym ACID denotes four important properties that must generally apply to stateful resources used within the context of a transactional unit of work within a traditional, two-phase commit-enabled transaction system.

- **Atomicity** requires that the updates to stateful resources used within the context of a transactional unit of work be made in an *all or nothing* fashion.
- **Consistency** refers to the ability of a transaction to leave resources in a consistent state, even in the event of failure.
- **Isolation** ensures that partial updates to stateful resources used within the transaction are not visible outside of the transaction until the end of the transactional unit of work. Isolation is implemented by means of concurrency control, or transactional locking, as it is sometimes referred.
- **Durability** provides for the permanence of stateful resource updates made under the transactional unit of work.

The ability to associate a transactional recovery policy to the execution of a Web service message exchange is described in the Web Services Atomic Transaction specification [WS-AtomicTransaction]. In the presence of a transactional unit of work, a Web service capable of participating in the transactional protocol must abide by the rules of two-phase-commit transaction management. However, in the absence

of a transaction management policy, the Web service is under no obligation to recover the state of the WS-Resource in the event of a failure.

The WS-Resource definition is not prescriptive with respect to policy that governs concurrent read or write access to a WS-Resource through a Web service. The definition of specific policy governing concurrent updates, whether or not separate message executions targeting the same WS-Resource may be interleaved, and whether partially completed WS-Resource updates within a given message execution may be observed by other concurrent requests is beyond the scope of the WS-Resource definition. If WS-Resource isolation is needed, we suggest the use of a transaction [WS-AtomicTransaction] to provide a context within which isolation of WS-Resource updates can be provided. In the absence of a transactional unit of work, the level of WS-Resource update atomicity, recovery, isolation, and durability provided by a Web service is implementation dependent.

We believe that the ability to declare and attach isolation-level policy to the definition of a Web service message exchange, whether or not a transactional unit of work is present, represents a general requirement not met by the current Web service architecture. In the future, isolation-level policy declarations may be introduced as a formal part of the WS-Resource definition.

7 WS-Resource Security

The ability to associate security related policy with a Web service is described in the WS- security specifications [WS-Security]. In the presence of a valid security context associated with a message exchange, a Web service capable of participating in the expressed security protocols must implement and enforce the security policies implied by the security context. In the absence of such security policy, the Web service is under no obligation to secure the execution of the message exchange nor the state of the WS-Resource designated by the WS-Resource context associated with the message request.

The WS-Resource definition is not prescriptive with respect to policy that governs access permission to a WS-Resource through a Web service. The definition of specific security policy governing access to the WS-Resource is beyond the scope of the WS-Resource definition. If WS-Resource access control is required, we suggest the use of the functions defined in the WS-Security specifications to provide a security context for the WS-Resource. In the absence of a valid security context and associated access control policies, the extent to which the Web service provides security of the WS-Resource is implementation dependent.

8 Conclusions

We have presented the WS-Resource approach to standardizing the representation of, and access to, stateful resources in a distributed environment. This approach defines the patterns by which state is represented and manipulated, so that a Web service can describe the stateful resources (WS-Resources) to which it provides access, and a service requestor can discover the type of that WS-Resource and use standardized operations to read, update, and query values of its state, and to manage its lifecycle.

The WS-Resource approach facilitates the construction and use of interoperable services, by making it possible for different service providers and service consumers

to describe, access, and manage their stateful resources in standard ways. Equally importantly, it introduces support for stateful resources without compromising the ability to implement Web services as stateless message processors.

9 Acknowledgements

This paper has been developed as a result of joint work with many individuals and teams. The authors wish to acknowledge contributions from many people, including Nick Butler, Christine Draper, Sonny Fulkerson, Rob High, Jim Knutson, Tom Maguire, Susan Malaika, Jeff Nick, Chris Sharp, and Jay Unger. We also acknowledge those with whom we have discussed issues addressed in this paper, including Malcolm Atkinson, Carl Kesselman, and Savas Parastatidis.

10 References

[OGSI-Refactor]

Foster, I., Frey, J., Graham, S., Tuecke, S., From Open Grid Services Infrastructure to Web Services Resource Framework: Refactoring and Evolution.

[OGSI-Spec]

Open Grid Services Infrastructure (OGSI) V1.0
<http://forge.gridforum.org/projects/ggf-editor/document/draft-ogsi-service-1/en/1>

[Parastatidis]

Parastatidis, S., Webber, J., Watson, P., Rischbeck, T., A Grid Application Framework based on Web Services Specifications and Practices, Technical Report, North East Regional e-Science Centre, University of Newcastle.

[Physiology]

Foster, I., Kesselman, C., Nick, J., Tuecke, S., The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, Globus Project, 2002. Available at <http://www.globus.org/research/papers/ogsa.pdf>

[SOAP]

The fundamental message enveloping mechanism in Web services.
<http://www.w3.org/TR/SOAP>.

[Tao]

Burbeck, S. The Tao of e-business Services.
<http://www.ibm.com/developerworks/webservices/library/ws-tao/>. October, 2000.

[Vogels]

Vogels, W. Web Services are not Distributed Objects: Common Misconceptions about the Fundamentals of Web Service Technology. *IEEE Internet Computing*, 7 (6). 2003.

[Web Services]

Ferguson, D., Lovering, B., Shewchuk, J., Storey, T. *Secure, Reliable Transacted Web Services* <http://www-106.ibm.com/developerworks/webservices/library/ws-securtrans/>

[WS-Addressing]

WS-Addressing, an XML serialization and standard SOAP binding for representing network wide "pointers" to services.

<http://www.ibm.com/developerworks/webservices/library/ws-add/>

[WS-Arch]

The W3C Web Services Architecture working group, public draft, August 2003.

<http://www.w3.org/TR/2003/WD-ws-arch-20030808/>

[WS-AtomicTransaction]

<http://www.ibm.com/developerworks/webservices/library/ws-atomtran/>

[WS-MetaDataExchange]

WS-MetadataExchange is a set of Web service mechanisms to exchange policies, WSDL, schema and other metadata between two or more parties. This specification is part of the Web services roadmap for WS-Federation.

[WS-Security]

The roadmap to the various security related Web services standards. See

<http://www.oasis-open.org>

[WSDL 1.1]

The Web Services Description Language, version 1.1. W3C Note:

<http://www.w3.org/TR/wsdl>.

[WSDL 2.0]

The Web Services Description Language, version 2.0. W3C Note:

<http://www.w3.org/TR/wsdl20/>.

[WSRF]

The Web Services Resource Framework.