Web Services Manageability – Concepts

(WS-Manageability)

Version 1.0
September 10, 2003

Authors

Mark Potts, Talking Blocks (mark.potts@talkingblocks.com)
Igor Sedukhin, Computer Associates (igor.seduhkin@ca.com)
Heather Kreger, IBM (kreger@us.ibm.com)

Contributors

Ellen Stokes, IBM (stokese@us.ibm.com)

THE SPECIFICATION OR THE PERFORMANCE OR IMPLEMENTATION OF THE CONTENTS THEREOF.

You may remove these disclaimers from your modified versions of the Specification provided that you effectively disclaim all warranties and liabilities on behalf of all copyright holders in the copies of any such modified versions you distribute.

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

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

Status of this Document

This document is provided as a submission to the OASIS Web Services Distributed Management Technical Committee and to the Web service community in general.

Abstract

As Web services become pervasive and critical to business operations, the task of managing Web services and implementations of the Web services architecture will be imperative to the success of business operations. Web services manageability is defined as a set of capabilities for discovering the existence, availability, health, performance, and usage, as well as the control and configuration of a Web service within the Web services architecture. This implies that Web services can be managed using Web services technologies. The importance of a standardized management model for Web services and the promise of Web services as a management integration technology has driven industry leaders in Web services and management technologies and applications to form a technical committee in OASIS called the Web Services Distributed Management (WSDM) Technical Committee.

The WSDM TC is chartered to develop the specifications for defining how to use Web services to manage any IT resource – the "WSDM Management Using Web Services" specification. They are also charted to define the manageability model for a Web service as an IT resource and how to access that model using Web services – the "WSDM Management OF Web Services" specification. WS-Manageability defines the later, the manageability model for a Web service and how to access that model using Web services. WS-Manageability does not provide a "Management Using Web Services" specification. But it does provide insight, illustration, and input into how "Management Using Web Services" should be developed to support Web services and Grid services communities through a concrete use case: Managing Web services.

WS-Manageability

WS-Manageability is composed of three documents, a concepts document, a normative specification, and renderings listing. These documents are summarized below:

The WS-Manageability – Concepts document outlines the scope and definitions for the specification. It provides an overview of Web services architecture and implications for management of that architecture. The Concepts document further defines the role of the manager in the Web services architecture and provides practical information on manageability implementation patterns and discovery considerations.

The WS-Manageability - Specification begins by introducing the general concepts of a manageability model in terms of manageability topics, (identification, configuration, state, metrics, and relationships) and the aspects (properties, operations and events) used to define them. These abstract concepts apply to understanding and describing the manageability information and behavior of any IT resource, not just Web services. We use these concepts to organize our approach to Web services manageability.

The manageability model for Web services endpoint is defined as concrete models in UML using the topics and aspects concepts, without implying any particular implementation or locus of implementation. Appropriate manageability interfaces are defined based on the UML manageability models. While some parts of this model may be useful for modeling the manageability of any IT resource, this specification is focused exclusively on the requirements of Web service endpoints and does not propose a complete generic resource manageability model. The WSDM "Management Using Web Services" specification may incorporate these more generic models.

The WS-Manageability – Representation document provides the interface definitions based on the model as WSDL 1.1 and GWSDL renderings. These definitions are meant to show how the topics and aspects concepts along with concrete models can influence the development of consistent Web services interfaces for accessing the manageability information of Web services. The interfaces illustrate how the manageability model for Web services can be divided into aspects of topics that apply to all manageable resources and aspects of topics that apply only to the manageability of Web service endpoints. The interfaces that may apply to all manageable resources may be incorporated into the "WSDM Management Using Web Services" specification. The details of these WSDL interfaces specific to managing a Web service endpoint may change to remain consistent with the "WSDM Management Using Web Services" specification when it is available.

This specification depends on Web services as a distributed platform. There are several common functions which are required for management using Web services that are not specific to management and not yet available as standardized Web service technologies. Interim solutions have been proposed in this specification which will be replaced by suitable standards when they become available. It is not expected that the WSDM TC will continue specification of or standardization of the interim solutions.

This specification defines the manageability model for Web services, and then uses Web services as a 'worked example' of how manageability interfaces in general, as well as Web services manageability interfaces, may be rendered based on one approach to management using Web services practices.

Notational Conventions

The key words “MUST,” “MUST NOT,” “REQUIRED,” “SHALL,” “SHALL NOT,” “SHOULD,” “SHOULD NOT,” “RECOMMENDED,” “MAY,” and “OPTIONAL” are to be interpreted as described in RFC-2119 [RFC 2119]. This is not yet done consistently. This specification uses namespace prefixes throughout; they are listed in Table 1. Note that the choice of any namespace prefix is arbitrary at this time and not semantically significant.

Prefix	Namespace
soap	http://schemas.xmlsoap.org/wsdl/soap/
crm	http://www.gridforum.org/namespaces/2003/17/crm
wsdl	http://www.w3.org/2002/07/wsdl
gwsdl	http://www.gridform.org/namespaces/2003/gridWSDLExtensions
sd	http://www.gridforum.org/namespaces/2003/serviceData
ogsi	http://www.gridforum.org/namespaces/2003/OGSI
http	http://www.w3.org/2002/06/wsdl/http
xsd	http://www.w3.org/2001/XMLSchema
xsi	http://www.w3.org/2001/XMLSchema-instance
cbe	http://www.ibm.com/AC/commonbaseevent1_1

Table 1: *Some standard prefixes and namespaces commonly used in this document*.

Unresolved issues with the submission are interspersed in appropriate locations through this specification, are highlighted in yellow.

149
150 ## Table of Contents

1. Introduction

The emergence of Service-Oriented Architectures (SOA) [1] and industry commitment to Web services and Grid computing [2] requires new management capabilities in order to truly achieve the visions and goals of these initiatives. Because, both Web services and Grid environments, leverage the loosely coupled nature of services, deployed across heterogeneous platforms and ownership domains and can dynamically discover and interact with each other during solution design or execution, the manageability of such environments is more critical and challenging to the business they support. Web services can also cross administrative domains leading to multiple aspects of control requiring a greater emphasis on declarative and real-time management through the use of management policies and service level agreements.

This submission details a manageability model and architecture that form the basis for the Management of Web Services. It directly addresses the requirements for manageable Web service architecture implementations identified through the work of the W3C Web service architecture group (WSAG) and it's Management Task Force (MTF). This submission also takes into account and leverages the work completed and underway in other management domains, including the DMTF's Common Information Model (CIM) and the Global Grid Forum's Open Grid Services Infrastructure specifications and Common Management Model (CMM) proposals.

This document is structured as follows: **Section 2** introduces the Manageability Requirements defined by the W3C and the Management Task Force, the scope of this work and potential future work. **Section 3** gives an overview of the Web service architecture and the role of management within that architecture. **Section 4** defines the Management Architecture within which we define manageability of Web services **Section 5** introduces the principles of description and discovery of manageable Web services.

The concepts are used in the "Web Services Manageability – Specification" which defines and details the Web services Manageability Model. The specification provides the representation of the manageability model in WSDL 1.1, and Grid WSDL**.** It explains how this document's material pertains to the work completed in this area for the Grid, specifically the Common Management Model (CMM).

***Note**: Even though SOAP is used throughout the document for clarity and examples, Management capabilities can be exposed through other protocols including any protocol capable of supporting the XML infoset.*

***Note**: The techniques used to express the Web services manageability model in schema, WSDL and GWSDL descriptions are techniques chosen based on their availability at the time of this draft. These techniques may need to evolve based on new specifications, best practices and techniques for expressing manageability information in WSDL 1.1, WSDL 1.2, and GWSDL.*

2. Requirements, Definitions and Scope

2.1. Manageability of Web services

As Web services become pervasive and critical to business operations, the task of managing Web services and implementations of the Web services architecture will be imperative to the success of business operations. Management of Web services in this case is defined as a set of capabilities for discovering the existence, availability, performance health, usage, control and configuration of resources within the Web services architecture. Like other Web Services, manageable Web services architecture implementations need to be secure, including support for authentication, authorization, integrity, and confidentiality.

The W3C Web Services Architecture Group has identified a set of requirements specific to the management of Web services architecture and Web services. For convenience, the Manageability Goals from the Requirements document [3] are restated here;

D-AC018 The Web Services Architecture must enable the management and provisioning of Web Services.

- AC018.1 Ensure that implementations of the Web Services Architecture are manageable.
  - AR018.1.1 Define a base set of standard metrics for architectural components and their interactions accompanied by guidelines for measurement.
  - AR018.1.2 Define a base set of standard management operations for Web Services Architecture implementations. Management operations includes, but is not limited to, support for configuration control and lifecycle control.
  - AR018.1.3 Define a base set of management events to be issued by the Web Services Architecture implementation.
  - AR018.1.4 Define a standard methodology for accessing management capabilities from the Web Services Architecture implementation.
- AC018.2 Ensure that implementations of the Web Service instances are manageable.
  - AR018.2.1 Define how a web service should expose web service specific metrics, configuration, operations, and events.
  - AR018.2.2 Support the discovery of web service management capabilities.
  - AR018.2.3 Define a standard methodology for accessing management capabilities of a Web Service through the Web Services Architecture implementation.
- AC018.3 Ensure that at least the following types of management aspects are supported: Resource Accounting, Usage Auditing and Tracking, Performance Monitoring, Availability, Configuration, Control, Security Auditing and Administration, and Service Level Agreements.

2.2. Scope of Work

This submission is initially limited in scope to the definition of a sufficient set of management capabilities for the architectural element Web services endpoint (Web service definition, endpoint definition). We do expect that the scope of this

submission will be expanded to define the manageability of the other elements of the Web service architecture.

As a result of these requirements for Manageable Web services the W3C introduced the role of the *Manager* (see Web Service Architecture and the Management Role) which uses the manageability information and capabilities provided by manageable resources. The definition of the *Manager* role, beyond the information model offered to it and how it interacts with the existing roles of the Web services architecture, is outside the scope of the W3C Web Services Architecture draft and therefore also outside the scope of this submission.

The work also has given considerations to the Grid initiatives that also have similar management requirements for Grid services (themselves Web services). Management of Web services is not purely a technical or physical perspective on management but a service perspective and directly ties into the business objectives of the provider and concerns of the consumer.

2.3. Definitions

These definitions are derived from the W3C definitions surrounding the management architecture.

- **Web service endpoint** is an association between a Binding and a network address, specified by a URI that may be used to communicate with an instance of a Service.
- **Manageable** implies that a Web service can be managed by a management system.
- **Managed** implies that an entity is actively being managed by a management system.
- **Management capabilities** include identification, configuration, metrics, status, operations, and events offered by manageable entities for management purposes**.**
- **Manageability** implies the existence of a sufficient set of management capabilities such that an entity is manageable.
- **Management** is the utilization of the management capabilities by the management system.
- **Manageability interface** is that interface through which manageability capabilities are offered.
- **Management endpoint** is the Web services endpoint offering the manageability interface for management purposes.

We define one additional term.

- **Manageable resources** include the subset of components specific to the Web service architecture (Web services, Web service endpoint Web Service Environment, Clients, Client Environments and Web service Descriptions)

and roles defined in a Web service architecture (Discovery Agency, Requestor, Provider, Manager).

3. Basic Web Service Architecture

3.1. Web Services Architecture

W3C defines a Web Service as a software system which functions can be accessed via standard internet protocols by exchanging structural data (XML) messages.

Although architectural concepts, constraints and relationships are defined in the W3C Web Services Architecture, this document outlines some of the basic principles of the architecture in a more concrete manner to facilitate understanding Web services manageability. We have used the WS-I Basic Profile [4] for Web services, to help explain Web services using XML, WSDL, SOAP, and HTTP technologies. The concepts here still apply when other interaction technologies besides SOAP and HTTP are used.

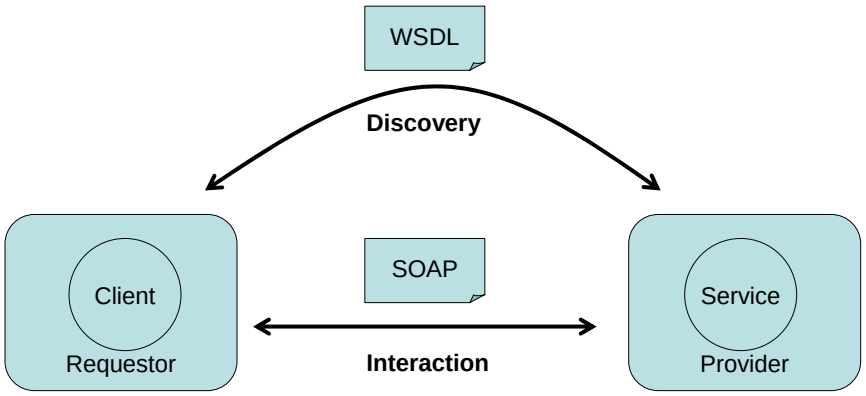


Figure 1: Web Services Architecture

Figure 1, represents an abstract architecture with elements such as roles, activities and artifacts. *Roles* are named sets of activities. *Activities* are named sets of procedures (steps, invocations, or interactions). *Artifacts* are named sets of data (information). Every named element has its associated meaning described in detail below.

When actually implemented, an element of a software system may take on one or more roles and therefore be responsible for activities associated with both of those roles.

In most practical cases, a Web service (also referred to as a *service* in this document) is owned by a *provider*. A provider is a role that could represent a legal entity e.g. business or a person owning or willing to provide/offer access to certain functionality via software. A provider has to describe its service in a machine processable manner, that is, in a Web Services Description Language (WSDL) document (artifact). A provider is also responsible for establishing and maintaining an endpoint for a service. An endpoint is an addressable location where data messages could be sent to request execution of some function from the service. The location address, transport

details, etc. are usually part of the service description, e.g. a WSDL document, but can also be defined as a references that is resolved at run-time.

A *requestor* is a role that accesses and uses the functionality of the service offered by the provider. The requestor first discovers the service and understands its requirements and functions by finding and inspecting its WSDL document. A requestor is then able to realize its intent by sending messages (artifacts) carrying structured data (XML) representing the request to the *service endpoint*. Similar reply messages may then be delivered in response.

For interoperability, an endpoint must support a SOAP (WS-Interoperability) binding for its WSDL interface. The endpoint may also offer additional bindings.

The understanding of the provider's intent to offer certain functionality as a service is encoded in the WSDL document. The requestor's intent to use that functionality is encoded in the messages sent to the service endpoint.

The service and the client in Figure 1 are abstract representations of the entity realizing the functionality and of the entity intending to use the functionality. An example of a client could be a Windows application used by a human. An example of a service could be a piece of software returning a stock quote by accessing information in a database.

3.2. Implementation of the Web Services Architecture

The abstract architecture may not present enough details for those actually planning or designing (architecting) Web services-enabled software applications in any organization. For this reason, this document briefly outlines a deployable implementation of the Web Services Architecture and the resulting Services Network (Figure 3).

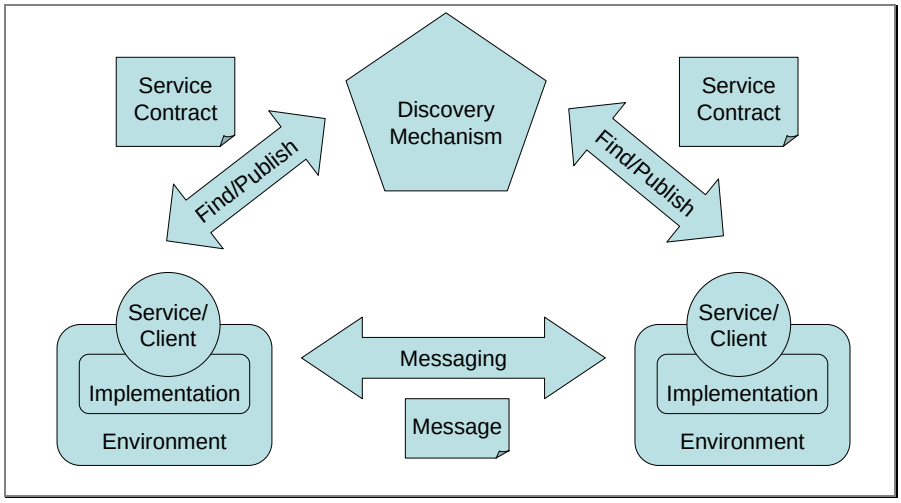


Figure 2: An Implementation of the Web Services Architecture

Note: *The depiction above shows the simplest scenario for a Web services architecture but could include and does not preclude support of the concepts of intermediaries and proxies, playing both roles of client and service, between ultimate requester and service provider.*

This implementation (Figure 2) builds on the abstract Web Services Architecture and closely resembles other known enterprise software architectures.

When a Web service application is conceived, it has an implementation that is deployed to a suitable environment. An *implementation* is the code that executes, and it is controlled by the *environment* to which it is deployed. Environments can be viewed as purely software (e.g. a platform such as .NET, J2EE) or hardware (e.g. a server) or both. It is likely that many implementations will be deployed to disparate environments but all interact regardless of the vagrancies of the platforms or implementations.

The implementation, often in conjunction with the environment, exposes its functionality as one or more services and, in turn, can consume other services as part of the execution of its functions. For example, an order fulfillment service may use a shipping service to obtain a package tracking number.

Therefore, one implementation could realize abstract service and client roles simultaneously. The service and the client are both abstractions representing exposed and consumed functionality respectively. It is a responsibility of the provider to describe exposed functionality and access details in one or more *Service Contracts*. Service contract is, in many cases, just a WSDL document, but may additionally contain other machine processable markup such as security requirements and policies, e.g. WS-Policy.

The provider is also responsible for publishing service contracts (all pertinent information, technical and business, required for usage of the service) via a chosen *Discovery Mechanisms*. It could use centralized corporate UDDI registry, WSIL files, public search engines, or a peer-to-peer discovery mechanism such as the one found in Open Grid Services Architecture. The presence and use of a discovery mechanism is a key to any sensible implementation of the Web services architecture.

The ability of implementations to dynamically discover and use functionality as Services is one of the great advantages of Web services. Service contracts that define access to abstract functionality could be established independently of its actual implementation or the runtime environment. Properly designed and built Web services applications are loosely-coupled and can be flexibly changed. This enables the logical parts of applications to be dynamically replaced or re-implemented to accommodate new business requirements and changing circumstances.

Service implementations that interact form a logical network of services layered over the infrastructure (e.g. software environments, servers, etc.) and the physical connectivity network. This logical network is also commonly called a Services Network (see Figure 3). As a Services Network grows, its existence and performance

becomes highly relevant to the main line of business, just like other critical IT resources.

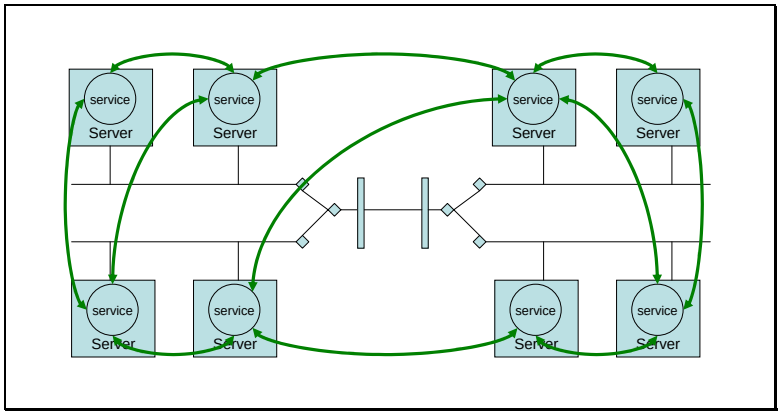


Figure 3: Web Services Network

3.3. Anatomy of Web Services Implementations

Within the Web services architecture the focal elements are the *service* and the *client*. The abstract idea of a service, once conceived, has to be defined, implemented, deployed, and offered (not strictly in that order) such that it can be utilized by clients. A requestor has to recognize the need for a service, discover it, and implement the client to use the service.

Figure 4 depicts the organizing principles and relationships between the elements of the Web Services Architecture that provide for the materialization of a service concept into an invokable service. The functional definition of a service or, in other words, an interface of a service is what determines the service type. A particular instance of a service type is described as a Web service endpoint and includes the information required to invoke the service. Definition of a service endpoint includes a service access point that defines the network location at which the service can be invoked. In terms of the WSDL, the logical service type is described in the interface, the service endpoint is an endpoint or port, and the service access point is the address (location).

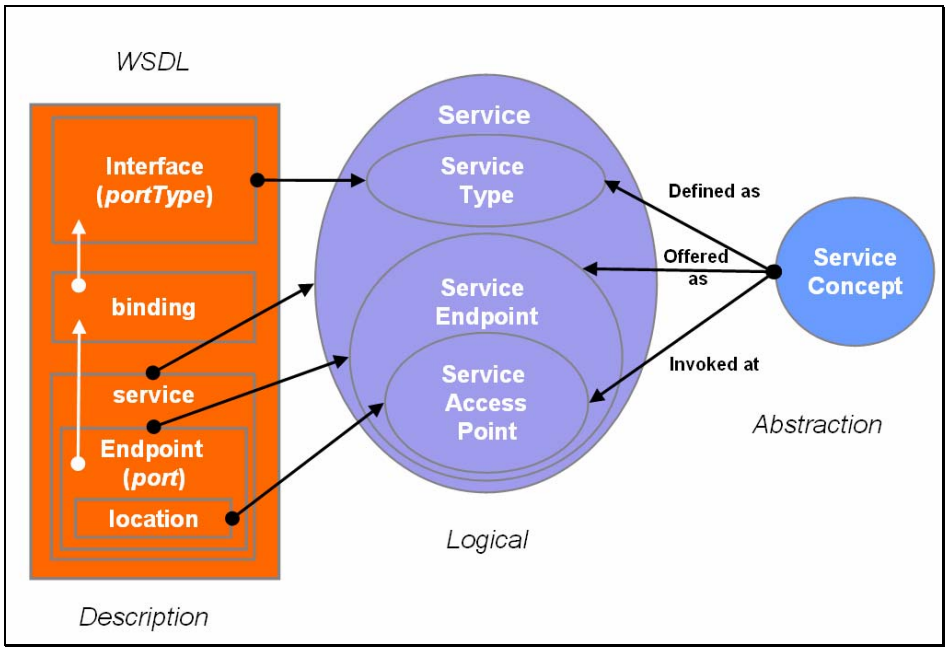


Figure 4: Abstract anatomy of a Service Definition

The process for defining, implementing, offering, and potentially publishing a service can be seen in more detail below (Figure 5). The initial business concept of a service materializes in the logical and physical architecture as shown. Some elements are logical in terms of the service being offered, and some are physical in terms of the implementation. These elements may well be the responsibility of different parties.

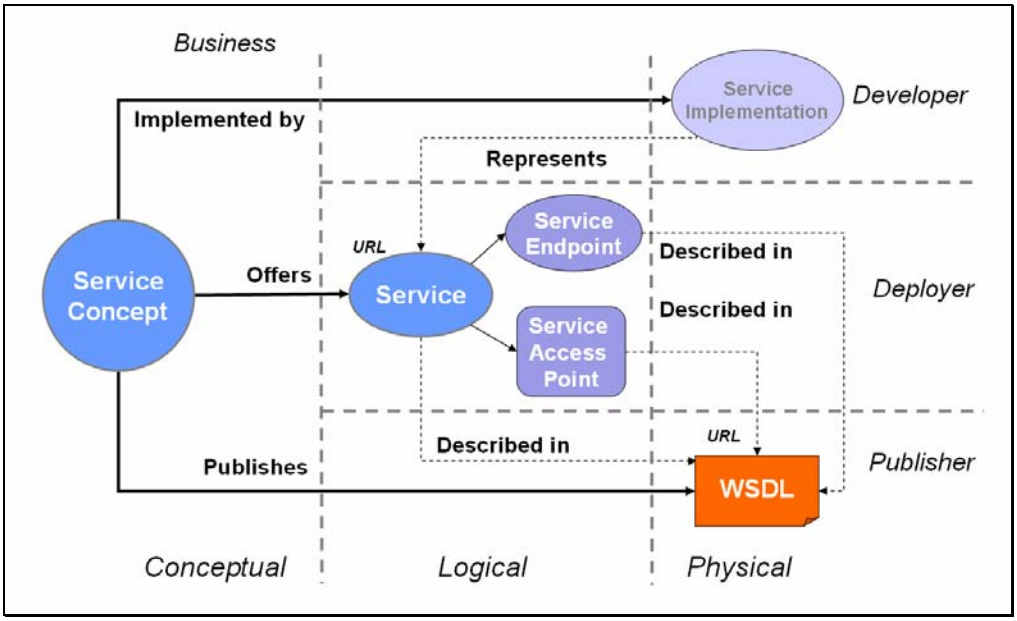


Figure 5: Anatomy of a Service Lifecycle

The same perspective can be used to look at the anatomy of a client. Figure 6 shows how a business need for a service materializes by discovering WSDL, implementing a client (requesting application), and using the client to interact with the service.

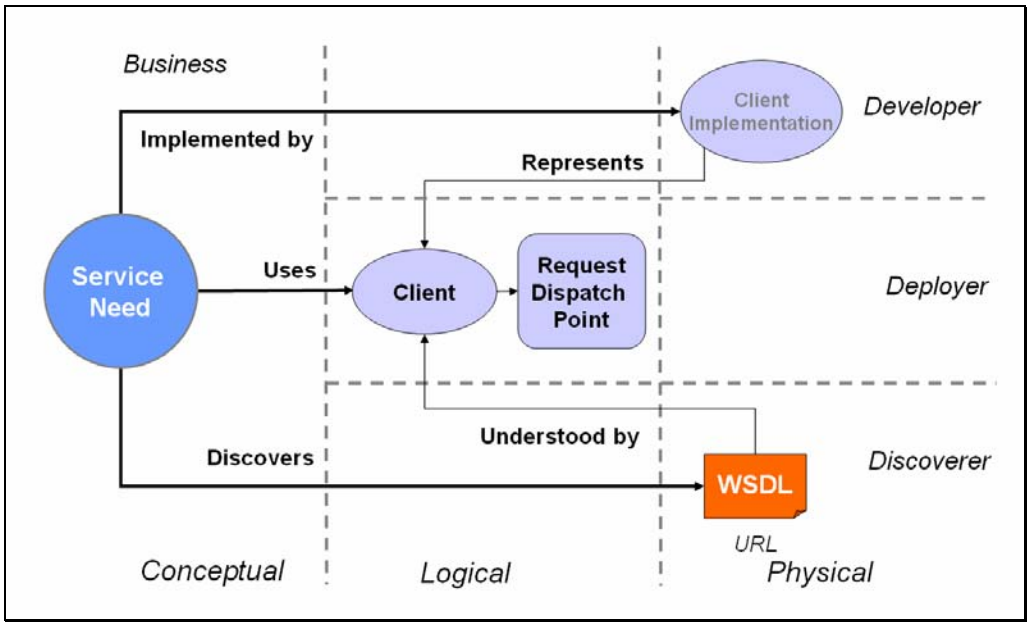


Figure 6: Anatomy of a Client Lifecycle

The anatomy of the service and client can be viewed even more concretely by looking at the implementation and deployment of both elements. Figure 7 depicts the anatomy of a service within the provider.

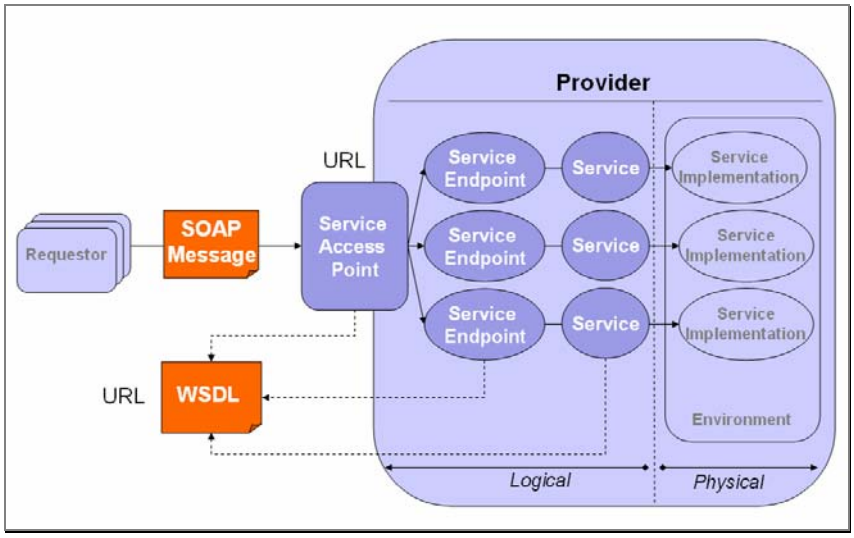


Figure 7: Provider perspective on services

Each *service* has an implementation (*service implementation*). It is the code or logic that implements the functionality offered by the service. It is exposed as a Web service through a service endpoint or potentially multiple service endpoints. Each endpoint is essentially a listener for protocol specific messages at the given URL (via a *service access point*). The URL may provide access to one endpoint or many endpoints.

The *service* itself is the logical element that represents the offered functionality, which is implemented by the service implementation.

The *service endpoint* is the representation of the access mechanisms to the service. A service endpoint may be supported by several service implementations for scalability, differentiated service, or SLA enforcement.

A *service access point* is the logical element that represents the URL at which endpoints receive messages and to which requestors send messages. A URL may be shared by multiple endpoints. In such sharing cases, message dispatch happens based on the content of the messages and their context. For example, the header of the message can identify the exact recipient of the routed message.

The *environment* hosts or executes service implementations and provides for service access points. It provides support for a stack of protocols and infrastructure involved in offering a Web service. For example, it may include HTTP over TCP/IP listeners, SOAP processor, XML marshalling/de-marshalling, etc. The Environment can be perceived as a software platform, a hardware server, or an entire data center.

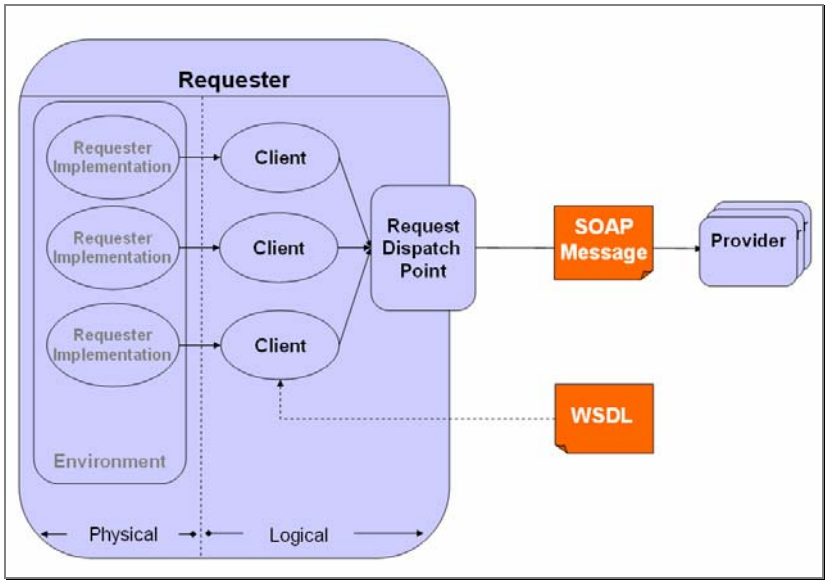


Figure 8: Requester perspective on Clients

In Figure 8, the anatomy of a *client* is portrayed within the requestor role of the Web services architecture. It also shows the *requester implementation* and its *environment*.

A *requestor implementation* utilizes the formal description of a Web service which it reads from the WSDL document.

The client represents the capabilities natively to the requester implementation such that the Web service can be invoked. For example, a requester implementation may be a Windows GUI application and the client is the part that encapsulates the particular knowledge about a given service, such as how to marshal parameters into XML representation and how to pack a SOAP message properly to the service.

The environment hosts and executes requester implementations and acts as the *dispatch point*. The dispatch point sends messages assembled by the client to one or more providers of the service. It then receives the response from the service. The client further processes it into a form that the requester implementation expects (e.g. a set of Java objects).

3.4. Web Service Relationships

Intrinsic relationships exist between elements of the architecture (e.g. services, their implementations and the environments they are deployed to), and between Web services themselves. The latter represents the business/functional semantics of the services rather than the way services are realized in the architecture.

Web services interact with each other to offer new composite Web services, and the interactions form the logical networks of services.

The relationships, important to managing Web services, can be summarized into four distinct categories: 1) relationships between Web service interfaces, 2) relationships between Web service endpoints, 3) relationships between Web service endpoints and other architectural elements specified in the Web services architecture, and 4) relationships between Web service endpoints and other manageable resources outside of the WSA.

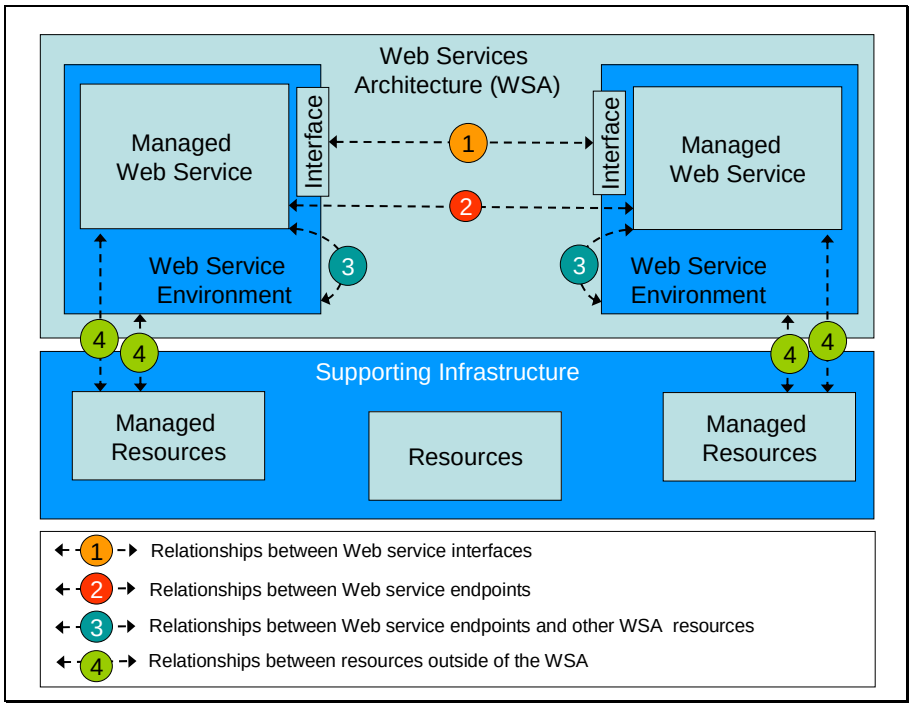
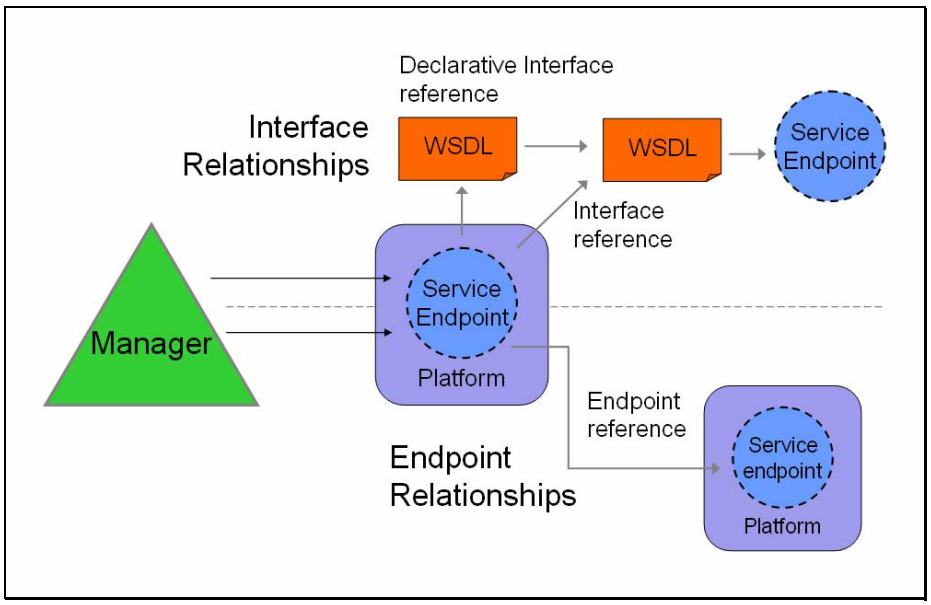


Figure 9: Web service Relationship Categories

The relationships defined for a Web service in this document fall into the first two groups – the relationships between Web service interfaces and Web service endpoints. The continued work surrounding the management of the Web services architecture and the underlying infrastructure will no doubt lead to additional relationships currently outside the scope of this document.

Web service relationships are declared in two ways - at an interface level or at a service endpoint level. Declaring relationships at the interface level allows for visibility into the relationships even where the actual Web service endpoint is resolved dynamically at runtime. Interface level relationships can be defined declaratively in the WSDL document describing the Web service, or they can be made available by the Web service endpoint itself via manageability interfaces. Service endpoint relationships are only made available by the Web service endpoint via manageability interfaces.



4. The Role of Management in the Web Services Architecture

To help represent manageability of the Web Services Architecture, a *manager* role was introduced (Figure 11) by the W3C Web Services Architecture Working Group. A manager is responsible for management of the manageable resources in the architecture (see definitions).

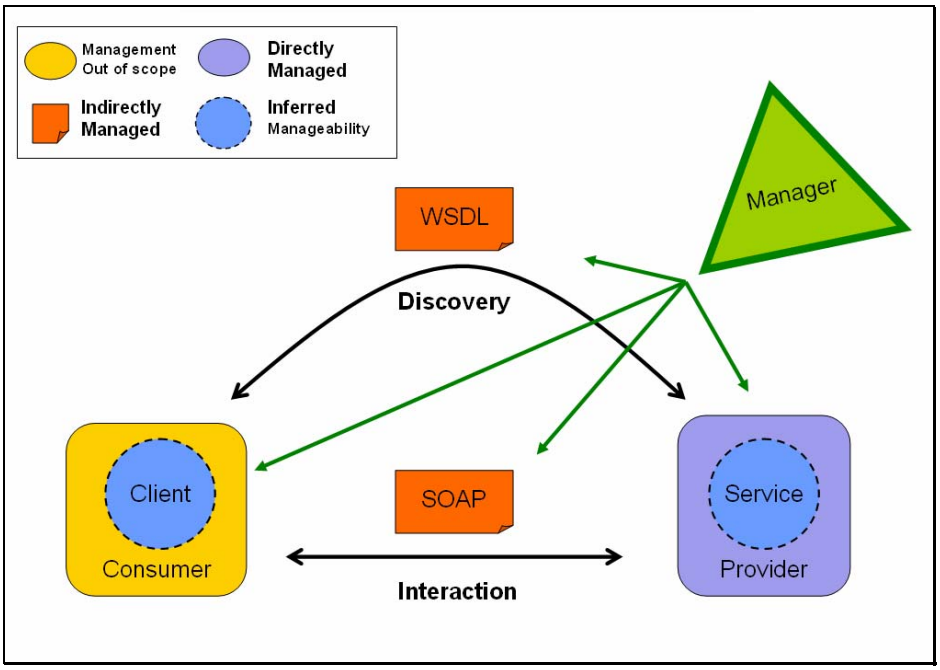


Figure 11: The Manager Role

Various resources of the architecture may have different *manageability capabilities*, and the manager is essentially the "user" of those capabilities (see Figure 12). A manageability capability could be something as simple as enabling/disabling certain function of an element. For example, it could stop acceptance of all requests by a Web service. Manageability capabilities are realized through a *manageability information model* that represents the manageability of resources and manageability information concerning those resources (state, configuration, relationships etc). The model can be accessed by managers which may retrieve information and exert control. For example, semantics of the manageability of a certain resource can be expressed in a UML model, and access to those specific manageability semantics may be described in WSDL interfaces and XML Schema data types. Managers adhering to both standardized access and semantics may manage those resources in an interoperable manner.

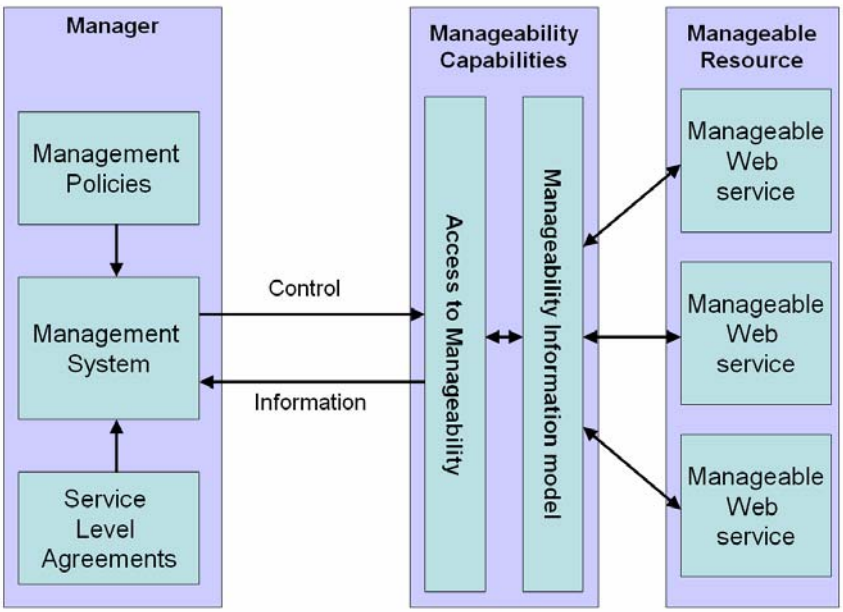


Figure 12: Web service Manageability Capabilities

Some elements of the architecture are *directly managed*. This means that a manager can exercise manageability capabilities offered by that element directly. For example, a manageability interface may be exposed by the element itself. In the Web services architecture, a service endpoint is an element that can be directly managed.

The manager is also aware of other elements of the architecture that are indirectly managed**.** *Indirect management* means the manager can request or subscribe to the information about those elements but doesn't directly access the element for the information. For example, SOAP messages can be recognized, counted, and related to the managed Web service, yet a manager cannot invoke an operation on a SOAP message.

There is also information important to manageability beyond that provided directly or indirectly by elements of the architecture. This manageability information is *inferred* through analysis of the information provided. For example, a SOAP message header can contain a digital certificate of the client's identity. A manager would have to extract that information and translate into the client's identification that could be used for management purposes later on (e.g. counting SOAP messages sent to a particular Web service by a particular client).

4.1. Anatomy of Web Service Manageability

The anatomy of a Web service discussed in section 3.3 can be categorized according to the type of manageability available. Service endpoints (following through to services) are directly manageable by any actual implementation that assumes the manager role (e.g. a management system). The SOAP messages and WSDL documents are indirectly manageable. Manageability of client and service concepts is inferred.

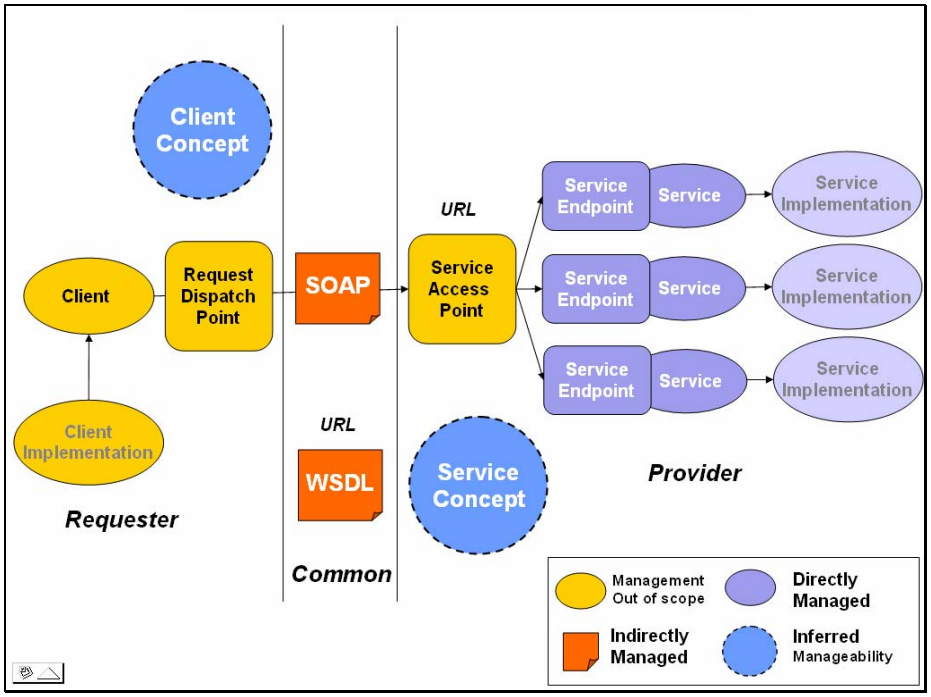
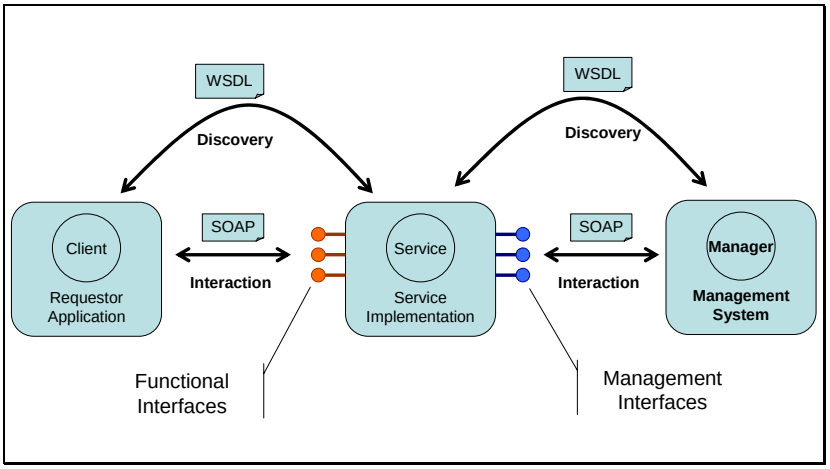


Figure 12: Anatomy of the Web Service Manageability

4.2. Web Services Management Architecture

A Web service is manageable if its manageability capabilities are exposed via standard manageability interfaces. The manageability interfaces must be accessible in a way compliant with the Web Services Architecture. In other words, manageability interfaces are exactly similar to the functional interfaces exposed by a Web service, only that they bear management related semantics and are used by the manager and not by the client. Figure 13 illustrates manageability interfaces exposed by a manageable Web service and accessible by the management system that assumes the role of the manager. A WSDL document exists that describes the manageability interfaces and the endpoints to which messages with management related payloads could be sent.

***Note**: An application might assume the role of the client and the role of the manager simultaneously. In this case, the application might exercise business functions of the Web service and manage it at the same time.*



606
607 **Figure 13:** Manageable Web Services
608
609

610 ## *4.3. Management Patterns*

611

612 ### 4.3.1. Consolidated Interfaces

613 In the first management pattern (Figure 14), one WSDL document describes one
614 service with an endpoint whose implementation supports both functional and
615 manageability interfaces. Messages carrying functional requests and management
616 requests can be sent to the same endpoint, and the implementation will do the proper
617 task depending on the content of the message. The endpoint will provide access
618 control and authorization to the operations in the manageability and business
619 interfaces.
620
621 This pattern is applicable when a Web service provider intends to offer business
622 functionality and its manageability within a single description of the service. This is
623 the simplest method for implementation and use of the business functionality and its
624 manageability. It is similar to security policies and enforcement mechanisms being
625 described along with the service they apply to. The drawback of this pattern is the "all
626 or nothing" nature it brings to visibility. Whoever discovers and uses the business
627 functionality also discovers the manageability capabilities associated with the service.
628 It does not mean that manageability capabilities can automatically be used by the
629 same clients that use business functionality; access control policies may only allow
630 certain known clients to use the manageability interface.
631

632
633 **Figure 14:** Consolidated Interfaces Management Pattern

4.3.2. Associated Interfaces

In the second management pattern (Figure 15), the manageability is implemented and offered completely separately from the service itself. For example, it could be a management agent implanted into the hosting environment, or it could be that the environment itself provides manageability capabilities. In this case, two WSDL documents may exist: one that describes the functional Web service endpoint and another one that describes manageability interfaces exposed via an endpoint (manageability endpoint). A referencing mechanism associates the two endpoint descriptions and makes it possible to manage the Web service endpoint. Possible referencing mechanisms are outlined in the Describing and Discovering Manageability section later in this document.

This pattern is applicable when a Web service provider needs to separate visibility of the service from its manageability. It may be complex to implement, deploy, and use for the managers, but developers of the service implementation and client applications are not impacted directly with the manageability description and publication. It becomes fully the responsibility of the deployment and management activities. If a client needs to be aware of the manageability, such as it may need to query if service is up, then the client needs to be developed such that it is aware of both interfaces.

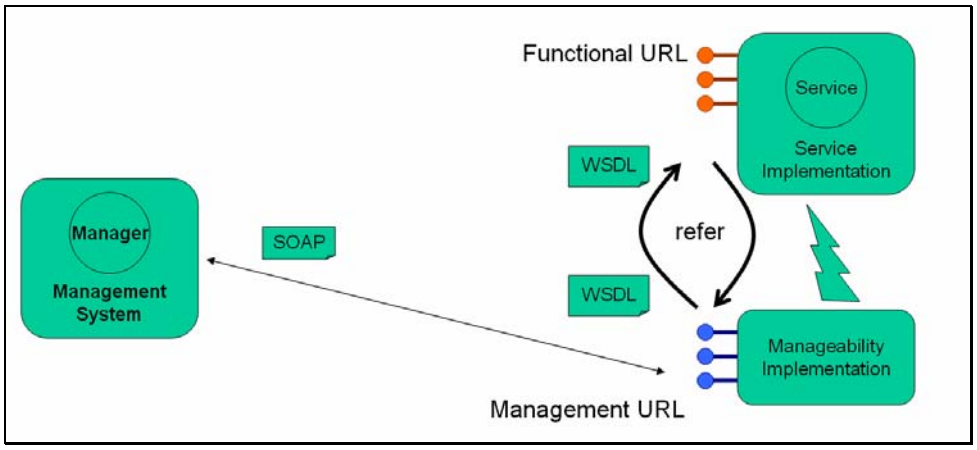


Figure 15: Associated Interfaces Management Pattern

4.4. Managers and Manageability Interfaces

4.4.1. Implementing the Manageability Interface

It is important that the manageability defined in this document be implementable in an easy, flexible, and interoperable manner. Many implementation and deployment techniques must be supported, and manageability must be achievable across many disparate solutions.

Figure 16 outlines three major methods of making a Web service manageable. First, a Web service implementation itself can offer manageability interfaces. This would be an *explicitly manageable* Web service implementation. Second, an *intermediary* can add the manageability interface to a Web service it represents. An intermediary in this case is used as a generic term for a service proxy or a service agent that is capable of interjecting itself between a service and a client. And, finally, an *environment* can offer manageability interfaces for the Web services it hosts. The latter two methods *delegate* provision of the manageability of a Web service to that other element (the environment or the intermediary).

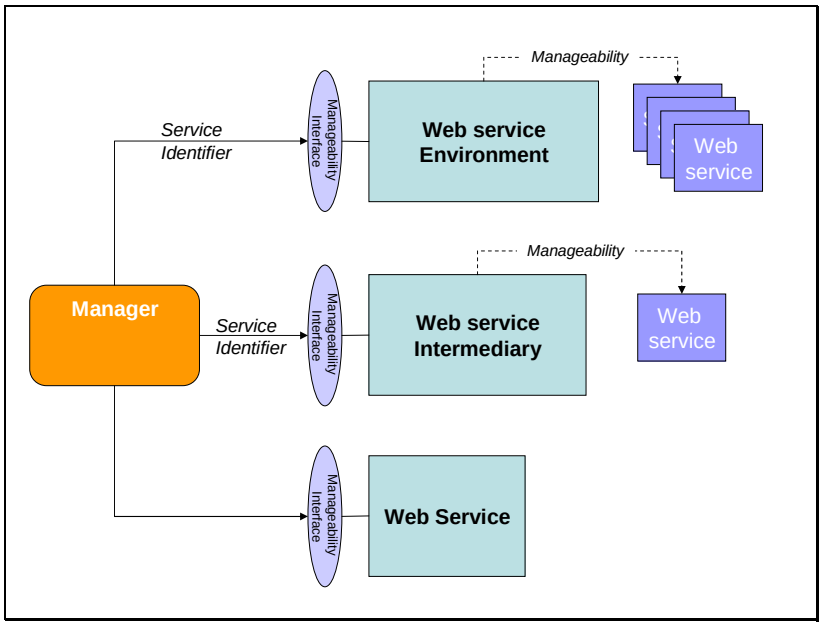


Figure 16: Delegated Manageability within the Web services Architecture

4.4.2. Multiple Managers Considerations

In large-scale enterprises where manageability capabilities and interfaces are standardized, individual services can be managed by multiple managers. Two common multiple manager scenarios are (1) separation of responsibilities between managers (i.e. one does performance monitoring and another one takes care of deployment changes) and (2) installations with management software from various vendors. The manageability interface and the manageability model must enable operational consistency. The provider of the manageability must ensure consistency

of simultaneously executed operations, and managers must ensure there are not executing conflicting operations.

As outlined earlier (Scope of Work), this document and the related Web Services Manageability Specification do not prescribe precisely how to implement manageability of a Web service or how to implement a manager. Only the agreed information model, access mechanism, and architecture are described for the purposes of the interoperability between manageable Web services and managers. Although manageability information model itself does not preclude multiple managers, actual implementation of the manageability of a Web service or a manager may or may not be capable of properly supporting multiple simultaneous managers.

Implementers and product architects supporting multiple manager environments should consider the following recommendations:

- Proper *access control* to the manageability capabilities of a Web service must be in place. The authentication and policy awareness/enforcement mechanisms must be compliant with Web Services Architecture. For example, a requirement for a WS-Security with X.509 certificates acquired from an identity authority could be stated by the security policy included in the description of the manageability interface (e.g. a WS-Policy statement).

- An implementation of the manageability capabilities of a Web service and the implementation of the manager must be aware of common and manager specific information in the manageability model, For example, if a Web service endpoint is not accepting any new requests, its state is 'DOWN' for all managers (common data). However, a set of event conditions about transitions or properties which an individual manager is interested in are specific to that manager.

- An implementation of the manageability capabilities of a Web service should provide a mechanism to establish *affinity* with between the service and the manager. For example, a session/context mechanism such as WS-SecureConversation [7] could be used. A Manager may be required to establish a context that expires over time. Alternatively, a context mechanism described in WS-Coordination [6] may be used or any other preferred standard method.

- Managers should be capable of making themselves *aware of the changing situation* around the manageable Web service. For example, if a particular state transition, configuration property or performance metric is important to a manager, it has to subscribe to events with respect to that information. Since other managers may cause state changes, the manager cannot assume that it will be the origin of all state changes for the manageable Web service.

- To avoid conflicting operation of the manageability of a Web service, managers must *coordinate and negotiate usage* amongst themselves. Coordination strategies should a) happen among peers – federation based on negotiation, b) happen in a delegated manner – a trusted third party coordinates usage, or c) happen by direct

735 command – a hierarchy of managers. Standards must be used to coordinate and
736 negotiate usage of the manageability. For example, federated coordination may be
737 supported by WS-Agreement [5] and delegated coordination by WS-Coordination
738 [6].
739

5. Description and Discovery of Manageability

A manageable Web Service (as described in section 4) has to provide machine processable description of the manageability capabilities and the manageability must be discoverable by the managers. Both the description and discovery must be conformant to the Web Services Architecture.

5.1. Describing the Manageability

The Web Services Management Architecture requires a Web service to have a description of its manageability capabilities in a WSDL document. The WSDL 1.2 definition of the principles of describing services, interfaces and endpoints is available at http://www.w3.org/TR/wsdl12/#intro_ws. Similar information about WSDL 1.1 is available http://www.w3.org/TR/wsdl#_introduction.

The manageability interfaces in this work are described in WSDL 1.1 (or in a WSDL with Grid extensions, i.e. GWSDL (see WS-Manageability Specification). These interfaces have to be made available for (or associated with) a given functional Web Service endpoint. According to the earlier requirement and to the WSDL itself, the manageability interface must also be offered via a Web Service endpoint.

The principles of describing the manageability of a Web Service and its endpoints are defined following a set of manageability patterns suggested in section 4.3.

5.1.1. Consolidated Interfaces

WSDL 1.2 draft proposes an interface composition mechanism directly in the language. A WSDL 1.2 fragment that represents consolidation of interfaces may look as follows.

```
<definitions name="AConsolidationOfManageabilityExampleInWSDL12"
  targetNamespace="http://www.oasis-open.org/wsdm/example/3"
  xmlns:tns="http://www.oasis-open.org/wsdm/example/3"
  xmlns:wsdm="urn:wsdm:webservice:endpoint:manageability" … >

<import namespace="urn:wsdm:webservice:endpoint:manageability"
  location="http://www.oasis-
open.org/wsdm/ManageableWebServiceEndpoint.wsdl">
[. . .]
<interface name="myBusinessFunction"> … </interface>
[. . .]
<interface name="myServiceInterface"
  extends="tns:myBusinessFunction wsdm:ManageableEndpoint"/>
…
<service name="myService" interface="tns:myServiceInterface">
  <endpoint name="myEndpoint" binding="tns:SecureSOAP">
    <soap:address>http://mycorp.com/myService</soap:address>
  </endpoint>
</service>
</definitions>
```

WSDL 1.1 does not have interface composition mechanism as part of the language. Therefore, consolidation of interfaces is only possible by composing (appending) operations of the manageability interface directly into the functional interface. The appended manageability operations must then be bound to endpoints in the same way that other functional operations were bound. The following WSDL 1.1 fragment represents consolidation of interfaces.

```
<definitions name="AConsolidationOfManageabilityExampleInWSDL11"
  targetNamespace="http://www.oasis-open.org/wsdm/example/4"
  xmlns:tns="http://www.oasis-open.org/wsdm/example/4"
  xmlns:wsdm="urn:wsdm:webservice:endpoint:metrics:data" … >

<import namespace=" urn:wsdm:webservice:endpoint:metrics:data"
  location="http://www.oasis-
open.org/wsdm/ManageableWebServiceEndpointMetricsData.wsdl">
[. . .]
<portType name="myServiceInterface"> …
  <operation name="myBusinessOperation"> … </operation>
  <operation name="getRequestCounters"> … </operation> …
</portType>
[. . .]
<binding name="mySecureSOAP" portType="tns:myServiceInterface"> …
  <operation name="myBusinessOperation"> … </operation>
  <operation name="getRequestCounters"> … </operation> …
</binding>
[. . .]
<service name="myService">
  <port name="myEndpoint" binding="tns:mySecureSOAP">
    <soap:address>http://mycorp.com/myService</soap:address>
  </port>
</service>
</definitions>
```

***Note** : In the case of consolidated interfaces there is no need to identify at runtime which endpoint manageability applies to. It is the same endpoint that exposes both interfaces. The functional interface and management interface are associated by being defined in the same service description. They can be defined in the same interface by using inheritance from business and manageability interfaces or by implenting the operations directly in the endpoint's interface.*

5.1.2. Associated Interfaces

There could be various mechanisms to associate the interfaces. Following are the mechanisms that are most likely to be used.

- Associating at "design time" by embedding a reference in the WSDL document.
- Associating at "runtime" by an interface method that returns a reference.
- Associating by external "discovery" of manageable elements.

Associating at runtime and by discovery is outlined in the section 5.2. The method of association at design time using WSDL extensions is outlined in section 1.2 of the Web Services Manageability – Relationships document.

5.2. Discovery of Manageability

In order for a manager to make use of the manageability of a Web service endpoint, it needs to:

- Understand that a Web Service endpoint is manageable
- Understand what manageability capabilities are supported by a Web service endpoint.

The process of discovery of the manageability is required to be compliant with the Web Services Architecture.

5.2.1. Discovery of Manageable Web Services

Manageable Web Service endpoints can be discovered by managers in the following ways.

- Use any standard Web Services discovery mechanism (e.g. search engine, UDDI registry, peer discovery such as WS-Inspection, etc.). Then locate manageability endpoint (consolidated or associated).
- Discover manageability endpoint via management discovery mechanism (proprietary or Web Services Architecture compliant, e.g. a UDDI registry). Then request identification of the Web service endpoint it applies to.
- Follow relationships of a manageable Web service endpoint to locate other related elements. Some of the related elements may be other Web service endpoints which also have manageability endpoints.

Basically, either way, after discovering a Web service endpoint or manageability endpoint, the manager has to understand which one is which. Following is the brief outline of how to locate the manageability endpoint having discovered a Web service endpoint and vice versa.

From functional Web Service endpoint to its manageability endpoint

When a manager discovers a Web service endpoint it must have the WSDL document that describes it. To determine if the given Web service endpoint is manageable, the manager must interrogate the WSDL document first. It must understand whether consolidated interfaces or associated interfaces patterns are used.

To determine if consolidated interfaces pattern was used, the manager simply finds if the endpoint offers standard manageability operations. It finds those operations by the input/output message definition QName. For example, a sufficient indication that an endpoint offers standard metrics data is an operation which the input message is **{urn:wsdm:webservice:endpoint:metrics:data,getRequestCountersIn}**. This approach works equally well for WSDL 1.1, GWSDL and 1.2.

In case of WSDL 1.2 there is a simpler way to determine if an endpoint is manageable in a consolidated manner. The manager could simply look up the “composition” line of an interface which the endpoint offers. The “composition” line is formed by taking wsdl:service element which contains the given endpoint description as wsdl:endpoint element and then recursively looking up wsdl:extends attribute of every interface that the service implements (wsdl:implements on

wsdl:service). For example, the manager may be looking for the **{urn:wsdm:webservice:endpoint:manageability,ManageableEndpoint}** interface QName or by inheritance from known manageability interfaces to determine if the endpoint is manageable.

To determine if associated interfaces pattern was used**,** a manager could use the WSDL extensibility mechanism described in section WS-Manageability specification document. A few XML tags from **urn:wsrl:webservice:endpoint:relationship** namespace appearing in the WSDL document would point the manager to the manageability endpoint.

The associated manageability endpoint could be established using the discovery mechanism such as UDDI registry. For example, a registry could contain an entry for a service (business service) and then two implementation entries (tModels). One tModel could contain a reference to a WSDL that describes the functional Web service endpoint, and another could contain a reference to a WSDL that describes the associated manageability Web service endpoint. The registry could control the visibility and only allow appropriate parties to see the second tModel pointing to the manageability endpoint description.

From manageability endpoint to the functional Web Service endpoint

A manager may know about a manageability endpoint. It may have established such an endpoint using a management-oriented discovery or by following relationships of manageable elements, etc. The manager, then, may need to find a description of the Web service endpoint which the manageability is provided for. For example it may need to know what functional operations are available, or it may want to send a synthetic transaction request to the functional endpoint.

Manageability interface declares access to the identification information of the endpoint it provides manageability for. A manager may use this information to locate the description of the functional Web service endpoint. Access to the identification must always be provided by the manageability endpoint that is used in the associated interfaces pattern.

Note that in case of consolidated interfaces pattern, manageability endpoint is the same as the functional endpoint and, therefore, may choose not to provide access to the identification information. Determining this is trivial for the manager, though. A manager must assume that if a manageable endpoint does not provide access to the identification information, it is a case of the consolidated interfaces pattern. A manager must then use a standard Web services discovery approach to determine the functional capabilities of the endpoint if that becomes necessary.

5.3. Discovery of Manageability Capabilities

A manager may need to know what capabilities a given manageability endpoint offers. There are two ways to determine that: (1) by interrogating the endpoint description, and (2) by asking the endpoint itself. Certain capabilities can be

discovered only from the description, and certain others can only be determined by asking (sending request at runtime).

For example, to determine if a manageability endpoint offers request counters data, a manager would simply lookup the description document if there is an operation with the input message **{urn:wsdm:webservice:endpoint:metrics:data,getRequestCountersIn}** QName. This way most other principal capabilities described in the composable, standard manageability interfaces can be determined.

An example of when a manager has to ask the manageability endpoint could be when determining the availability of a particular metric. In this case, a manager must send a request to retrieve the list of available metric QNames. It then has to check the returned list if a metric it is interested in is actually provided by the given manageability endpoint.

6. Summary

This document has outlined the fundamental concepts of Web services architectures, how the architecture is realized, and how the realization influences the manageability of Web services. Patterns for supporting manageability interfaces and discovery are outlined here to provide guidance during the development of manageability specifications. The WS-Manageability – Specification builds on these concepts to define the manageability model for Web services in an implementation agnostic manner. The WS-Manageability – Representations document renders the manageability model into interfaces expressed in WSDL 1.1 and GWSDL.

7. References

[1] **SOA – Service Oriented Architecture**: http://www.w3.org/TR/2003/WD-ws-arch-20030808/#service_oriented_architecture

[2] **Grid**: http://www.globus.org/research/papers.html#Overview%20Papers -

[3] **Web Services Architecture:** http://www.w3.org/TR/wsa-reqs

[4] **WS-I Basic Profile:** http://www.ws-i.org/Profiles/Basic/2003-08/BasicProfile-1.0a.htm :

[5] **WS-Agreement**: http://www.globus.org/research/papers/OGSI_Agreement_2003_06_12.pdf

[6] **WS-Coordination**: http://www-106.ibm.com/developerworks/library/ws-coor/ :

[7] **WS-SecureConversation**: http://www-106.ibm.com/developerworks/library/ws-secon/