# Web Services Architecture

## W3C Working Draft 8 August 2003

This version:
    http://www.w3.org/TR/2003/WD-ws-arch-20030808/
Latest version:
    http://www.w3.org/TR/ws-arch/
Previous version:
    http://www.w3.org/TR/2003/WD-ws-arch-20030514/
Editors:
    David Booth, W3C Fellow / Hewlett-Packard
    Hugo Haas, W3C
    Francis McCabe, Fujitsu Labs of America
    Eric Newcomer, Iona
    Michael Champion, Software AG (until March 2003)
    Chris Ferris, IBM (until March 2003)
    David Orchard, BEA Systems (until March 2003)

This document is also available in these non-normative formats: PostScript version and PDF version.



---

## Abstract

This document defines the Web Services Architecture. The architecture identifies the functional components, defines the relationships among those components, and establishes a set of constraints upon each to effect the desired properties of the overall architecture.

## Status of this Document

*This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.*

This is the third public Working Draft of the Web Services Architecure specification. It has been produced by the W3C Web Services Architecture Working Group, which is part of the W3C Web Services Activity. Since the last publication, the concepts and relationships have been organized into five architectural models (see **2.3 The Architectural Models** [p.18] ).

A list of open issues against this document is maintained by the Working Group.

Comments on this document should be sent to www-wsa-comments@w3.org ( public archives ). It is inappropriate to send discussion emails to this address.

Discussion of this document takes place on the public www-ws-arch@w3.org mailing list ( public archives ) per the email communication rules in the Web Services Architecture Working Group Charter.

Patent disclosures relevant to this specification may be found on the Working Group’s patent disclosure page.

Publication as a Working Draft does not imply endorsement by the W3C Membership. This is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to cite this document as other than ’work in progress.’

## Table of Contents

1 Introduction [p.5]
- 1.1 Purpose of the Web Service Architecture [p.5]
- 1.2 Intended Audience [p.6]
- 1.3 Document Organization [p.6]
- 1.4 Notational Conventions [p.6]
- 1.5 What is a Web service? [p.7]
  - 1.5.1 Agents and Services [p.7]
  - 1.5.2 Requester and Provider [p.7]
  - 1.5.3 Service Description [p.7]
  - 1.5.4 Semantics [p.8]
  - 1.5.5 The Role of Humans [p.8]
- 1.6 Architectural Style [p.9]
  - 1.6.1 Interoperability Architecture [p.9]
  - 1.6.2 Service Oriented Architecture [p.10]
  - 1.6.3 SOA and REST archictures [p.11]
- 1.7 Web Service Technologies [p.13]

2 Core Concepts and Relationships [p.16]
- 2.1 Introduction [p.16]
- 2.2 How to read this architecture [p.16]
  - 2.2.1 Concepts [p.16]
  - 2.2.2 Relationships [p.16]
  - 2.2.3 Feature [p.17]
  - 2.2.4 Model [p.17]
  - 2.2.5 Conformance [p.18]
- 2.3 The Architectural Models [p.18]
  - 2.3.1 Message Oriented Model [p.20]
    - 2.3.1.1 Correlation [p.20]
    - 2.3.1.2 Intermediary [p.21]
    - 2.3.1.3 Message [p.22]
    - 2.3.1.4 Message envelope [p.24]

2.3.1.5 Message Exchange Pattern (MEP) [p.24]
2.3.1.6 Message Header [p.27]
2.3.1.7 Message description language [p.28]
2.3.1.8 Message Identifier [p.28]
2.3.1.9 Message Path [p.29]
2.3.1.10 Message recipient [p.29]
2.3.1.11 Message sender [p.30]
2.3.1.12 Message Transport [p.30]
2.3.1.13 Reliable messaging [p.31]
2.3.2 The Service Oriented Model [p.32]
2.3.2.1 Action [p.32]
2.3.2.2 Agent [p.33]
2.3.2.3 Choreography [p.34]
2.3.2.4 Choreography Description Language [p.34]
2.3.2.5 Service [p.35]
2.3.2.6 Service description [p.37]
2.3.2.7 Service end point [p.38]
2.3.2.8 Service interface [p.39]
2.3.2.9 Service operation [p.39]
2.3.2.10 Service provider [p.39]
2.3.2.11 Service requester [p.40]
2.3.2.12 Service semantics [p.41]
2.3.2.13 Service Task [p.42]
2.3.3 The Resource Oriented Model [p.42]
2.3.3.1 DELETE [p.43]
2.3.3.2 Discovery [p.44]
2.3.3.3 Discovery Service [p.44]
2.3.3.4 GET [p.45]
2.3.3.5 Identifier [p.46]
2.3.3.6 PUT [p.46]
2.3.3.7 Representation [p.47]
2.3.3.8 Resource [p.47]
2.3.4 The Policy Model [p.48]
2.3.4.1 Audit guard [p.49]
2.3.4.2 Authentication [p.50]
2.3.4.3 Domain [p.51]
2.3.4.4 Obligation [p.51]
2.3.4.5 Permission [p.52]
2.3.4.6 Permission guard [p.53]
2.3.4.7 Person or organization [p.53]
2.3.4.8 Policy [p.54]
2.3.4.9 Policy guard [p.55]
2.3.5 The Management Model [p.56]
2.3.5.1 Deployed element [p.57]
2.3.5.2 Life cycle [p.58]
2.3.5.3 Management capability [p.59]
2.3.5.4 Management configuration [p.59]

2.3.5.5 Management event [p.60]
2.3.5.6 Manager [p.61]
2.3.5.7 Manageable Element [p.61]
2.3.5.8 Manageability Interface [p.62]
2.3.5.9 Management metric [p.63]
2.4 Relationships [p.63]
2.4.1 The is a relationship [p.63]
2.4.1.1 Summary [p.63]
2.4.1.2 Relationships to other elements [p.63]
2.4.1.3 Description [p.64]
2.4.2 The describes relationship [p.64]
2.4.2.1 Summary [p.64]
2.4.2.2 Relationships to other elements [p.64]
2.4.2.3 Description [p.64]
2.4.3 The is expressed in relationship [p.65]
2.4.3.1 Summary [p.65]
2.4.3.2 Relationships to other elements [p.65]
2.4.3.3 Description [p.65]
2.4.4 The has a relationship [p.65]
2.4.4.1 Summary [p.65]
2.4.4.2 Relationships to other elements [p.65]
2.4.4.3 Description [p.65]
2.4.5 The realized relationship [p.66]
2.4.5.1 Summary [p.66]
2.4.5.2 Relationships to other elements [p.66]
2.4.5.3 Description [p.66]
3 Stakeholder's perspectives [p.66]
3.1 Web integration [p.66]
3.2 Information and service [p.67]
3.3 Web service agents [p.67]
3.4 Web Service Discovery [p.68]
3.4.1 Scenarios Not Requiring Discovery [p.68]
3.4.2 Scenarios Requiring Discovery [p.69]
3.4.2.1 Human Discovery [p.69]
3.4.2.2 Autonomous Selection [p.71]
3.4.2.3 Triangle Diagram [p.73]
3.5 Web service semantics [p.74]
3.6 Web services security [p.76]
3.6.1 Threats to security and privacy [p.77]
3.6.2 Policies [p.78]
3.6.2.1 Policies and security [p.79]
3.6.2.2 Policies and privacy [p.79]
3.6.3 Policies beyond security [p.80]
3.7 Scalability and extensibility [p.80]
3.8 Modularity [p.80]
3.9 Extensibility [p.81]
3.10 Peer to peer interaction [p.81]

3.11 Long running transactions [p.82]
3.12 Conversations [p.82]
3.13 Message reliability [p.82]
3.14 Web service manageability [p.84]
3.15 Web services technologies [p.85]
3.15.1 XML and Web services [p.86]
3.15.2 SOAP [p.87]
3.15.3 WSDL [p.87]

## Appendices

A Acknowledgments [p.88] (Non-Normative)
B References [p.89] (Non-Normative)
C Web Services Architecture Change Log [p.90] (Non-Normative)

---

# 1 Introduction

| **Editorial note** | |
|---|---|
| dbooth editing this section. The previous "Contract with the Reader" has been merged into this section. | |

## 1.1 Purpose of the Web Service Architecture

Web services provide a standard means of interoperating between different software applications, running on a variety of platforms and/or frameworks. This document (WSA) is intended to provide a common definition of a Web service, and define its place within a larger Web services framework to guide the community.

The WSA provides a model and a context for understanding Web services and the relationships between the various specifications and technologies that comprise the WSA. The WSA promotes interoperability through the definition of compatible protocols. The architecture does not attempt to specify how Web services are implemented, and imposes no restriction on how services might be combined. The WSA describes both the minimal characteristics that are common to all Web services, and a number of characteristics that are needed by many, but not all, Web services.

The WSA integrates different conceptions of Web services under a common "reference architecture". There isn't always a simple one to one correspondence between the architecture of the Web and the architecture of existing SOAP-based Web services, but there is a substantial overlap.

We offer a framework for the future evolution of Web services standards that will promote a healthy mix of interoperability and innovation. That framework must accommodate the edge cases of pure SOAP-RPC at one side and HTTP manipulation of business document resources at the other side, but focus on the area in the middle where the different architectural styles are both taken into consideration.

## 1.2 Intended Audience

This document is intended for a large and diverse audience. Expected readers include users and creators of individual Web services, Web service specification authors, and others.

## 1.3 Document Organization

This document provides introductory overview material followed by normative material.

The body of the architecture is presented as a set of core concepts and key relationships between them. A core concept is usually a noun, but does not have to be, and the relationships between conncepts are usually predicates, i.e., verbs. Both noun-style and verb-style concepts are present in the architecture, the latter playing a prominent role in the relationships between concepts.

A primary goal of the concepts section [p.16] is to provide a basis for measuring *conformance* to the architecture. For example, the resource [p.47] concept states that resources have identifiers (in fact they have URIs). Using this assertion as a basis, we can measure conformance to the architecture of a particular resource by looking for its identifier. If, in a given instance of this architecture, a resource has no identifier, then it is not a valid instance of the architecture.

While the concepts and relationships [p.16] represent an enumeration of the architecture, the stakeholders' viewpoints [p.66] approaches from a different perspective: how the architecture meets the goals and requirements. In this section we elucidate the more global properties of the architecture and demonstrate how the concepts [p.18] actually achieve important objectives.

A primary goal of the Stakeholder's Perspectives [p.66] section is to relate the actual architecture with the requirements of the architecture, especially as outlined in the Web services requirements [WSA Reqs] [p.89] document.

For example, in the **3.14 Web service manageability** [p.84] section we show how the management of Web services is modeled within the architecture. The aim here is to demonstrate that Web services are manageable and which key concepts and features of the architecture achieve that goal. In this case, manageability is realized by showing a link between the concept of a physically deployed resource [p.57] and the abstract concept it realizes (such as a Web service). Management of such deployed resources then leads to management of Web services themselves.

The key stakeholder's viewpoints supported in this document reflect the major goals of the architecture itself: interopability, extensibility, security, Web integration, implementation and manageability.

Where appropriate, the WSA also identifies candidate technologies that have been determined to meet the functionality requirements defined within the architecture.

## 1.4 Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC 2119] [p.89] .

# 1.5 What is a Web service?

There are many things that might be called "Web services" in the world at large. However, for the purpose of this Working Group and this architecture, and without prejudice toward other definitions, we will use the following definition:

[Definition: 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.]

## 1.5.1 Agents and Services

A Web service [p.35] is viewed as an abstract notion that must be implemented by a concrete agent [p.33] . (See Figure 1.) The agent is the concrete entity (a piece of software) that sends and receives messages [p.22] , while the service is the abstract set of functionality that is provided. To illustrate this distinction, you might implement a particular Web service using one agent one day (perhaps written in one programming language), and a different agent the next day (perhaps written in a different programming language). Although the agent may have changed, the Web service remains the same.

## 1.5.2 Requester and Provider

The purpose of a Web service is to provide some functionality on behalf of its owner -- a person or organization [p.53] , such as a business or an individual. The *provider entity* is the person or organization [p.53] that provides an appropriate agent to implement a particular service. (See Figure 1: Basic Architectural Roles.)

A *requester entity* is a person or organization [p.53] that wishes to make use of a provider entity's Web service. It will use a *requester agent* to exchange messages with the provider entity's *provider agent*. In order for this message exchange to be successful, the requester entity and the provider entity must first agree on both the semantics and the mechanics of the message exchange.

## 1.5.3 Service Description

The mechanics of the message exchange are documented in a Web service description [p.37] (WSD). (See Figure 1.) The WSD is a machine-processable specification of the Web service's interface. It defines the message formats, datatypes, transport protocols, and transport serialization formats that should be used between the requester agent and the provider agent. It also specifies one or more network locations ("endpoints") at which a provider agent can be invoked, and may provide some information about the message exchange pattern that is expected.

### 1.5.4 Semantics

The semantics [p.41] ("Sem" in Figure 1) of the message exchange represents the "contract" between the requester entity and the provider entity regarding the purpose and consequences of the interaction. It also includes any additional details on the mechanics of the message exchange that are not specified in the service description. Although this contract represents the overall agreement between the requester entity and the provider entity on how and why their respective agents will interact, it is not necessarily written or explicitly negotiated. It may be explicit or implicit, oral or written, machine processable or human oriented.

While the service description represents a contract governing the mechanics of interacting with a particular service, the semantics represents a contract governing the meaning and purpose of that interaction.

### 1.5.5 The Role of Humans

Although one of the main purposes of Web services is to automate processes that might otherwise be performed manually, humans still play a role in their architecture and use, notably in two ways:

1. Humans need to agree on the semantics and the service description. Since a human (or organization) ultimately is the legal owner of any Web service, people must either implicitly or explicitly agree on the semantics and the service description that will govern the interaction.

   Often this agreement will be accomplished by the provider entitity publicizing and offering both the semantics and the service description as take-it-or-leave-it "contracts" that the requester entity must accept unmodified as conditions of use. However, nothing in this architecture prevents them from reaching agreement by other means. For example, in some situations, the service description (excepting the network address of the service) may be defined by an industry organization, and shared by many requester and provider entities. In other situations, it may originate from the requester entity (even if it is written from provider entity's perspective).

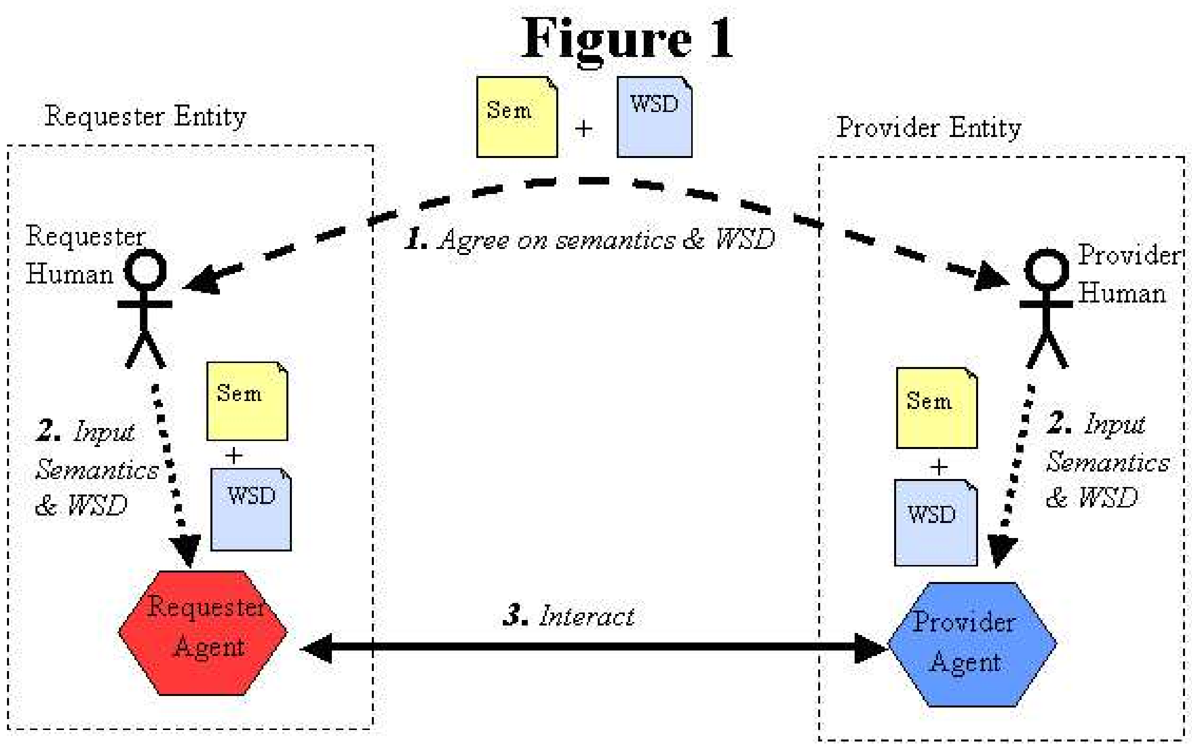
2. Humans create the requester and provider agents (either directly or indirectly). Ultimately, humans must ensure that these agents implement the terms of the agreed-upon service description and semantics. There are many ways this can be achieved, and this architecture does not specify or care what means are used. For example:

   - an agent could be hard coded to implement a particular, fixed service description and semantics;
   - an agent could be coded in a more general way, and the desired service description and/or semantics could be input dynamically; or
   - an agent could be created first, and the service description and/or semantics could be produced from the agent code.

   Regardless of the approach used, from an information perspective both the semantics and the service description must be somehow be input to, or embodied in, both the requester agent and the provider agent before the two agents can interact.

Figure 1: Basic Architectural Roles.

| **Editorial note** | |
|---|---|
| dbooth: Not sure how we should label the figures. Also, Figure 1 may need to be resized. | |




## 1.6 Architectural Style

### 1.6.1 Interoperability Architecture

The Web services architecture is an *interoperability* architecture it identifies those global elements of the global Web services network that are required in order to ensure interoperability between Web services. It is not intended to be an architecture for individual Web services; the structure and implementation of these is inherently private and is left to the discretion of the developers of these. However, in order to ensure interoperability, certain concepts, relationships and constraints are important; and this architecture identifies those.

The major goals of the architecture are outlined in the Web Services Architecture Requirements document [WSA Reqs] [p.89] . These goals are to promote

- interoperability between Web services,

- integration with the World Wide Web,
- reliability of Web services,
- security of Web services,
- scalability and extensibility of Web services, and
- manageability of Web services.

The role of this architecture is to provide a global perspective on the networked service architecture. Other specifications, such as [SOAP 1.2 Part 1] [p.89] and [WSDL 1.2 Part 1] [p.90] give detailed recommendations for specific requirements. This architecture is intended to show how these, and other related, technologies fit together to deliver the benefits of Web services.

Some non-goals of the architecture include:

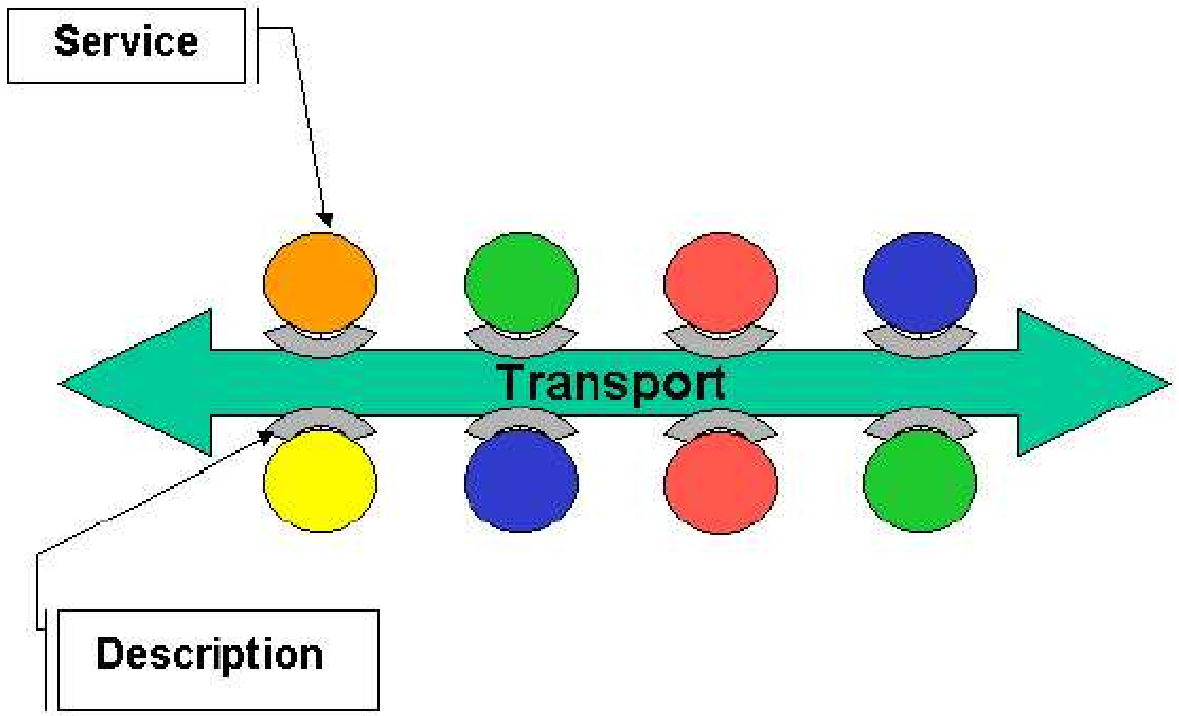
- to prescribe a specific programming model or programming technology
- to constrain the internal architecture and implementation of specific Web services
- to demonstrate how Web services are constructed
- to be specific about how messages or other descriptions are formatted
- to determine specific technologies for messaging, discovery, choreography etc.

### 1.6.2 Service Oriented Architecture

The Web architecture and the Web Services Architecture (WSA) are instances of a Service Oriented Architecture (SOA). To understand how they relate to each other and to closely related technologies such as CORBA, it may be useful to look up yet another level and note that SOA is in turn a type of distributed system. A distributed system, consists of discrete software agents that must work together to implement some intended functionality. Furthermore, the agents in a distibuted system do not operate in the same processing environment, so they must communicate by hardware/software protocol stacks that are intrinsically less reliable than direct code invocation and shared memory. This has important architectural implications because distributed systems require that developers (of infrastructure and applications) consider the unpredictable latency of remote access, and take into account issues of concurrency and the possibility of partial failure. [Samuel C. Kendall, Jim Waldo, Ann Wollrath and Geoff Wyant, "A Note On Distributed Computing"].

An SOA is a specific type of distributed system in which the agents are "services". For the purposes of this document, 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 (e.g., the purchase order processing capability of an enterprise resource planning system might be exposed as a discrete service), the users of that server need be concerned only with the interface description of the service. Furthermore, most definitions of SOA stress that "services" have a network-addressable interface and communicate via standard protocols and data formats.

Figure 2, Generic Service Oriented Architecture Diagram




The description of a service in a SOA is essentially a description of the messages that are exchanged. This architecture adds the constraint of stateless connections, that is where the all the data for a given request must be in the request.

| **Editorial note** | |
|---|---|
| Put in a good word about the Semantic web and semantics in general here | |

In essence, the key components of a Service Oriented Architecture are the messages that are exchanged, agents that act as service requesters and service providers, and shared transport mechanisms that allow the flow of messages. In addition, in public SOAs, we include the public descriptions of these components: descriptions of the messages, descriptions of the services and so on. These descriptions may be machine processable, in which case they become potential messages themselves: for use in service discovery systems and in service management systems.

### 1.6.3 SOA and REST archictures

The World Wide Web is a SOA that operates as a networked information system that imposes some additional constraints: Agents identify objects in the system, called "resources," with Uniform Resource Identifiers (URIs). Agents represent, describe, and communicate resource state via "representations" of the resource in a variety of widely-understood data formats (e.g. XML, HTML, CSS, JPEG, PNG). Agents exchange representations via protocols that use URIs to identify and directly or indirectly address the agents and resources. [Web Arch] [p.89]

An even more constrained architectural style for reliable Web applications known as "Representation State Transfer" or REST has been proposed by Roy Fielding and has inspired both the TAG's Architecture document and many who see it as a model for how to build Web services [Fielding] [p.89] . The REST Web is the subset of the WWW in which agents are constrained to, amongst other things, expose and use services via uniform interface semantics, manipulate resources only by the exchange of "representations", and thus use "hypermedia as the engine of application state."

The scope of "Web services" as that term is used by this Working Group is somewhat different. It encompasses not only the Web and REST Web services whose purpose is to create, retrieve, update, and delete information resources but extends the scope to consider services that perform an arbitrarily complex set of operations on resources that may not be "on the Web." Although the distrinctions here are murky and controversial, a "Web service" invocation may lead to services being performed by people, physical objects being moved around (e.g. books delivered).

We can identify two major classes of "Web services":

- REST-compliant or "direct resource manipulation" services in which in which the primary purpose of the service is to manipulate XML representations of Web resources using the a minimal, uniform set of operations operations,

- "distributed object" or "Web-mediated operation" services in which the primary purpose of the service is to perform an arbitrarily complex set of operations on resources that may not be "on the Web", and the XML messages contain the data needed to invoke those operations.

In other words, "direct" services are implemented by Web servers that manipulate data directly, and "mediated" services are external code resources that are invoked via messages to Web servers.

| **Editorial note** | |
|---|---|
| Lots of open terminology issues here, such as what we call these two types of services, and whether the "Web service" is the interface to the external code or the external code itself. | |

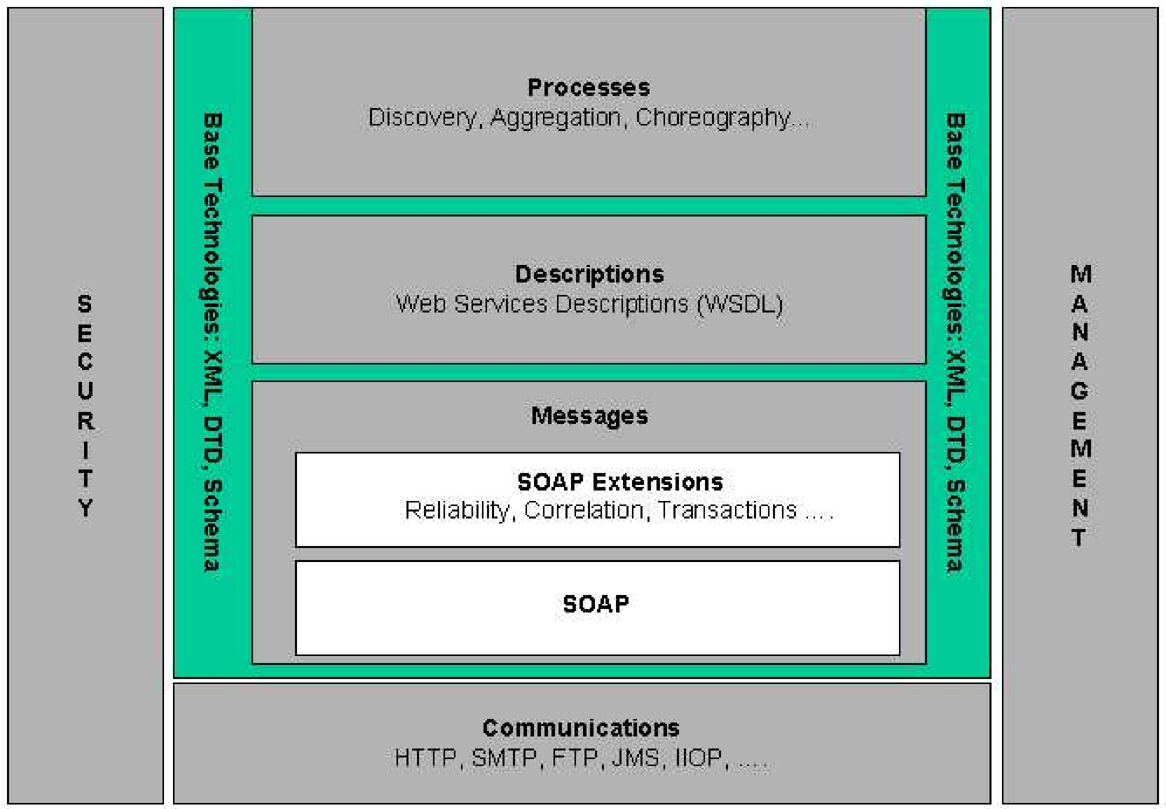
Both classes of "Web services" use URIs to identify resources and use Web protocols and XML data formats for messaging. Where they fundamentally differ is that "distributed object" (editors' note: or "mediated services") use application specific vocabularies as the engine of application state, rather than hypermedia. Also, they achieve some of their benefits in a somewhat different way. The emphasis on messages, rather than on the actions that are caused by messages, means that SOAs have good "visibility": trusted third parties may inspect the flow of messages and have a good assurance as to the services being invoked and the roles of the various parties. This, in turn, means that intermediaries, such as firewalls, are in a better situation for performing their functions. A firewall can look at the message traffic, and at the structure of the message, and make predictable and reasonable decisions about security.

In REST-compliant SOAs, the visibility comes from the uniform interface semantics, essentially those of the HTTP protocol: an intermediary can inspect the URI of the resource being manipulated, the TCP/IP address of the requester, and the interface operation requested (e.g. GET, PUT, DELETE) and determine whether the requested operation should be performed. The TCP/IP and HTTP protocols have a widely supported set of conventions (e.g. known ports) to support intermediaries, and firewalls, proxies, caches, etc. are almost universal today. In non-REST [Ed. note: or "distributed object" or "mediated" ] but XML-based services, the visibility comes from the fact that XML is the universal meta-format for the data. Intermediaries can be programmed or configured to use the specifics of the SOAP XML format, standardized SOAP headers (e.g. for encryption, digital signature exchange, access control, etc.), or even generic XPath expressions to make routing, filtering, and cacheing decisions. XML-aware firewall and other "edge appliance" products are just coming to market as of this writing.

## 1.7 Web Service Technologies

Web service architecture involves many layered and interrelated technologies. There are many ways to visualize these technologies, just as there are many ways to build and use Web services. Figure 3 provides one illustration of some of these technology families.

Figure 3: Stack Diagram.




Marketing documents from Web services vendors often contain a three-part diagram to show how the different Web services "standards" relate to one another: WSDL describes the format SOAP messages, and UDDI serves as a discovery service for the WSDL descriptions. The problem with such diagrams is that they don't convey the multiple dimensions of the Web services standards "space" and can't easily be extended to handle new standards, e.g. for security, management, choreography, and so on. In order to show the big picture of the Web services architecture as we envision it, the picture needs to be somewhat more complex.

First and foremost, XML is the "backplane" of the WSA. One can imagine Web services that don't depend on the SOAP envelope framework or processing model, or that don't employ WSDL to describe the interaction between a service and its consumers, but XML is much more fundamental. It provides the extensibility and vendor, platform, and language neutrality that is the key to loosely-coupled, standards-based interoperability that are the essence of the Web services value proposition. Furthermore, XML helps blur the distinction between "payload" data and "protocol" data, allowing easier mapping and bridging across different communications protocols, which is necessary in many enterprise IT infrastructures that are built on industrial-strength but proprietary components. Thus, the "base technology" of the WSA consists of some key XML specifications, including XML 1.x, XML Schema Description Language and the XML Base specification. Note that we do *not* rely on all XML technologies; for example, we do not rely on XML DTDs, in the architecture.

This leads to the next key concept in the WSA: services are invoked and provide results via messages that must be exchanged over some communications medium. The WSA encompasses a wide, almost infinite variety of communications mechanisms: HTTP (the dominant protocol of "the Web"), other Internet protocols such as SMTP and FTP, generic interface APIs such as JMS, earlier distributed object protocols such as IIOP, and so on. In principle, Web services invocation and result messages could be passed around by "sneakernet", RFC 1149-compliant carrier pigeons, or mechanisms that have not yet been invented. WSA says almost nothing about this communication layer other than it exists -- it does not specificy that it be at any particular level of the OSI reference architecture protocol stack, and allows Web services messages to be "tunnelled" over protocols designed for another purpose.

WSA does have quite a bit to say about the messages themselves, if not about the mechanism by which they are communicated. SOAP is the key messaging technology in the WSA: while very simple information transfer services can be implemented without SOAP, secure, reliable, multi-part, multi-party and/or multi-network applications are much easier to build if there is a standard way of packaging the messaging information in a protocol neutral way. This also allows the messaging infrastructure (which may be specialized hardware, SOAP intermediaries, or code libraries called by the ultimate recipient of a SOAP message) to provide authentication, encryption, access control, transaction processing, routing, delivery confirmation, etc. services. SOAP’s envelope (and attachment) structure and header / processing model have proven to be a very robust and powerful framework within which to do this.

Interoperability across heterogenous systems requires a mechanism to allow the precise structure and data types of the messages to be commonly understood by Web services producers and consumers. WSDL is an obvious choice today as the means by which the precise description of Web services messages can be exchanged.

| **Editorial note** | |
|---|---|
| Obviously we have open issues with respect to whether description mechanisms such as shared code "qualify" here. | |

In the future, more sophisticated description languages that handle more of the *semantic* content of the messages are likely to become technologically viable, and such languages (perhaps based on RDF and OWL) will fit well in the WSA framework.

Beyond the description of individual messages such as WSDL provides, the WSA envisions a variety process descriptions: the process of discovering service descriptions that meet specified criteria, the process of describing multi-part and stateful sequences of messages, the aggregation of processes into higher-level processes, and so on. This area is much much clearly defined than other parts of the WSA, but there is much work going on and the WSA incorporates them at an abstract level.

In addition to specific messaging and description technologies, the architecture also provides for security and management. These are complex areas that touch on many of the different levels and technologies deployed in the service of Web services.

# 2 Core Concepts and Relationships

## 2.1 Introduction

The formal core of the architecture is this enumeration of the core concepts and relationships that are central to Web services' interoperability.

## 2.2 How to read this architecture

The architecture is described in terms of a few simple elements: concepts, relationships, features and models. Concepts are often noun-like in that they identify things or properties that we expect to see in realizations of the architecture, similarly relationships are normally linguistically verbs.

As with any large-scale effort, it is often necessary to structure the architecture itself. We do this with two larger-scale "meta-concepts" — feature [p.17] and model [p.17] . Features are elements of the architecture which may be realized using concepts and relationships defined within the architecture itself. A model is a coherent portion of the architecture that focuses on a particular theme or aspect of the architecture.

### 2.2.1 Concepts

A concept is expected to have some correspondence with any realizations of the architecture. For example, the message [p.22] concept identifies a class of object (not to be confused with Objects and Classes as are found in Object Oriented Programming languages) that we expect to be able to identify in any Web services context. The precise form of a message may be different in different realizations — the message [p.22] concept tells us what to look for in a given concrete system rather than prescribing its precise form.

Not all concepts will have a realization in terms of data objects or structures occurring in computers or communications devices; for example the person or organization [p.53] refers to people and human organizations. Other concepts are more abstract still; for example, message reliability [p.82] denotes a property of the message transport service — a property that cannot be touched but none-the-less is important to Web services.

Each concept is presented in a stylized regular way: consisting of a short definition, an enumeration of the relationships with other concepts, and a slightly longer explanatory description. For example, the concept for agent [p.33] includes as relating concepts the fact that an agent is a [p.63] computational resource, has an identifier [p.65] and an owner. The description part of the agent [p.33] explains in more detail why agents are important to the archicture.

### 2.2.2 Relationships

Relationships denote a relationship between concepts. Syntactically, relationships are verbs; or more accurately, predicates.

A statement of a relationship typically takes the form: concept predicate concept. For example, in agent [p.33] , we state that:

An agent is [p.63]

a computational resource

This statement makes an assertion, in this case about the nature of agents. Many such statements are descriptive, others are definitive:

A message has [p.65]

a message sender [p.30]

Such a statement makes an assertion about valid instances of the architecture: we expect to be able to identify the message sender in any realization of the architecture. Conversely, any system for which we cannot identify the sender of a message is not conformant to the architecture.

### 2.2.3 Feature

A feature is a kind of concept that has a larger granularity — it may refer to a particular architectural requirement or to a network of concepts and relationships that defines a property with some internal structure. A key aspect of features is that they may have realizations [p.66] , possibly within the architecture itself. A realization of a feature is simply a way — expressed in terms of other concepts and relationships — of implementing the feature. More accurately, a realization is a way of ensuring that the feature is satisfied within the architecture itself.

For example, message correlation is a feature of the architecture. The requirement is to be able to associate a message with a particular context. Message correlation may be realized in one of several ways:

- message identifier in message
- message occurrence in a stream of messages

By identifying the message correlation concept as a feature, we show that message correlation is simultaneously an important concept in the architecture, and that there is some guidance on how message correlation may be achieved.

### 2.2.4 Model

A model is a coherent subset of the architecture that typically revolves around a particular aspect of the overall architecture. Models represent a complete explication of their focus; although there may be dependencies on other aspects of the architecture: these dependencies are well defined.

Each model is described separately below, in terms of the concepts and relationships inherent to the model. The ordering of the concepts in each model section is alphabetical; this should not be understood to imply any relative importance. For a more focused viewpoint the reader is directed to the Stakeholder's perspectives [p.66] section which examines the architecture from the perspective of key stakeholders of the architecture.

The reason for choosing an alphabetical ordering is that, inevitably, there is a large amount of cross-referencing between the concepts. As a result, it is very difficult, if not misleading, to choose a non-alphabetic ordering that reflects some sense of priority between the concepts. Furthermore, the ‘optimal ordering’ depends very much on the point of view of the reader. Hence, we devote the Stakeholders perspectives [p.66] section to a number of prioriterized readings of the architecture.

### 2.2.5 Conformance

Unlike language specifications, or protocol specifications, conformance to an architecture is necessarily a somewhat imprecise art. However, the presence of a concept in this enumeration is a strong hint that, in any realization of the architecture, there should be a corresponding feature in the implementation. Furthermore, if a relationship is identified here, then there should be corresponding relationships in any realized architecture. The absence of such a concrete feature may not prevent interoperability; but it will certainly make such interoperability more difficult.

A primary function of the Architecture’s enumeration in terms of models, concepts and relationships is to give guidance about conformance to the architecture. For example, the architecture notes that a message [p.22] has [p.65] a message sender [p.30] ; any realization of this architecture that does not permit a message to be associated with its sender is not in conformance with the architecture.

Unless otherwise noted, all predicate statements relating concepts are normative: i.e., they should be interpreted as MUST be satisfied. Otherwise, the keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC2119.

## 2.3 The Architectural Models

This architecture has five models:

- The Message Oriented Model [p.20] which focuses on messages, message structure, message transport and so on — without particular reference as to the reasons for the messages, nor to their significance.

- The Service Oriented Model [p.32] which focuses on aspects of service [p.35] , action and so on. While clearly, in any distributed system, services cannot be adequately realized without some means of messaging, the converse is not the case: messages do not need to relate to services.

- The Resource Oriented Model [p.42] focuses on resources [p.47] . The ROM is layered over the SOM, and yet its focus is not really service but the nature of resources and the service actions associated with them. It is quite possible to conceive of a resource model that does not involve service, and conversely it is possible to have services without a strong notion of resource — hence the separation.

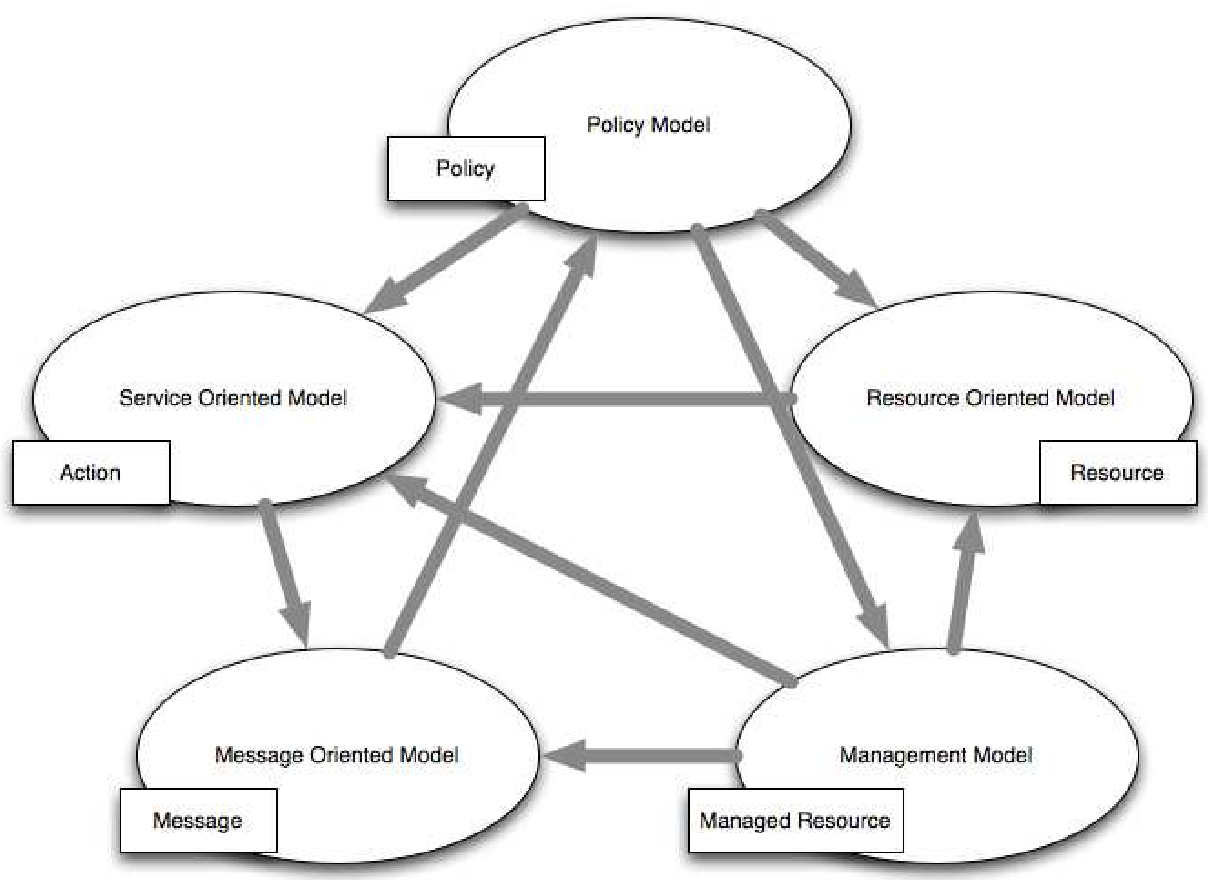
  Of course, in the context of Web services, we do expect the resource model to be quite important.

- The Policy Model [p.48] focuses on policies associated with the architecture. Put simply, a policy is just a constraint on the behavior of agents and services. Policies may be enacted to represent security concerns, quality of service concerns, management concerns and even application concerns. The policy model focuses on the core concepts needed to relate all such policies to Web services.

- The Management Model [p.56] focuses on the management of Web services. It is a model that explicates the relationship between deployed elements, other elements of the architecture, and in particular what kind of management fits well with the use of Web services for managing Web services.

  The Management model uses many of the other features and concepts of the architecture: including the concept of resource, description, service and so on. This is to be expected as deployed resources (for a Web service to be used by a requestor it must first be deployed) are also resources. The management view of a deployed resource is typically a meta-view of the resource.

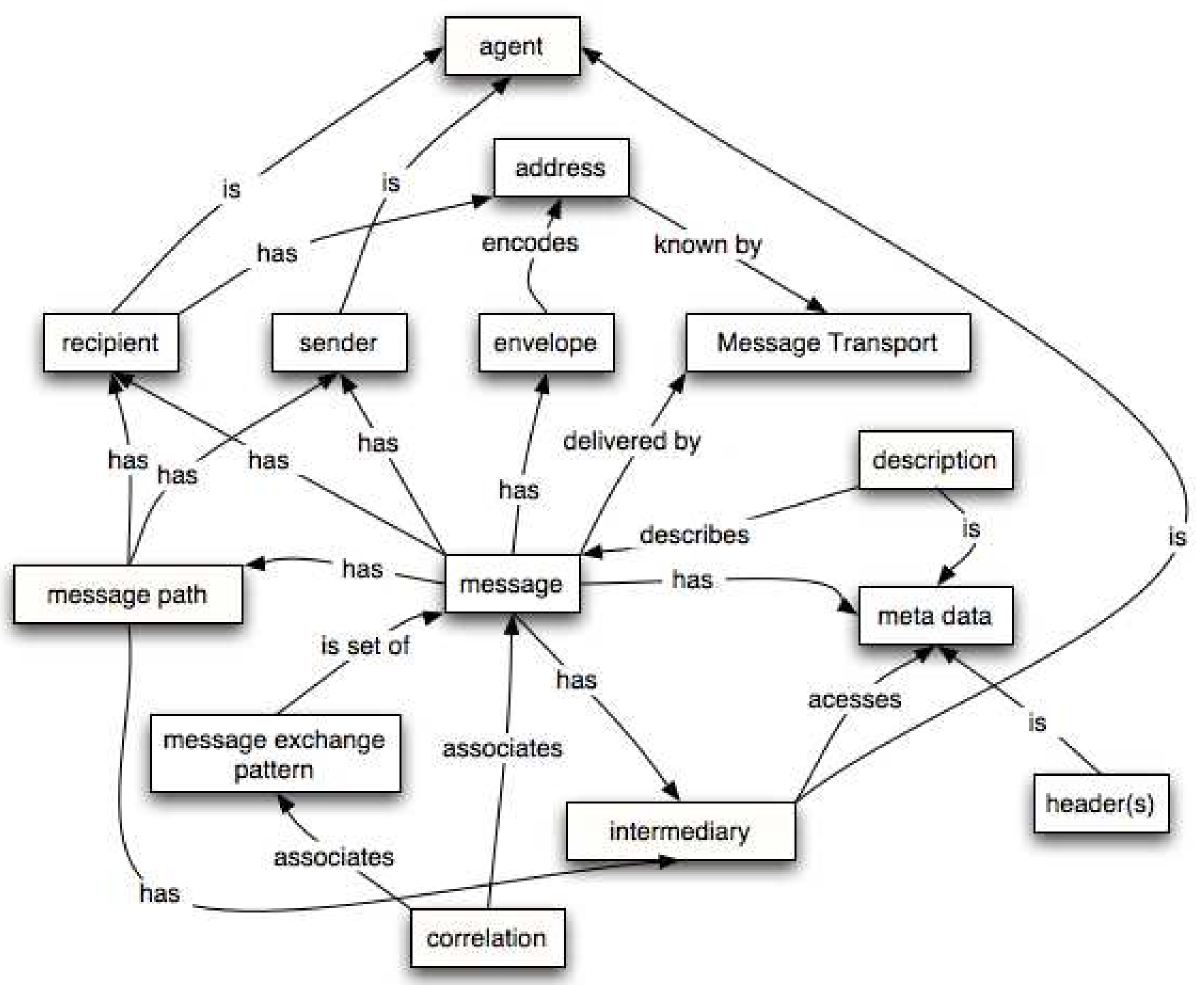
The figure below illustrates the key models in the architecture, and their dependencies on each other. Each model in the figure is labeled with what may be viewed as the key concept of that model:



### 2.3.1 Message Oriented Model

The Message Oriented Model focuses on those aspects of the architecture that relate to messages [p.22] and the processing of them. Specifically, in this model, we are not concerned with any semantic significance of the content of a message or its relationship to other messages. However, the MOM does focus on the structure of messages, on the relationship between message senders, receivers and other message processors (such as intermediaries [p.21] ).

The MOM is illustrated in the figure:




#### 2.3.1.1 Correlation

#### 2.3.1.1.1 Definition

Correlation [n.] is the association of a message with a context. Message correlation ensures that the agent requesting a service execution can match the reply with the request, especially when multiple replies may be possible.

A message identifier is a piece of information that is used to establish a relationship between one or more messages. A conversation identifier is used to establish a relationship between one or more parties that may exchange multiple messages.

#### 2.3.1.1.2 Relationships to other elements

Correlation is [p.63]

a feature [p.17]

Correlation is [p.63]

a means of associating a message within a specific conversational context.

Message correlation may be realized [p.66]

by including message identifiers to enable messages [p.22] to be identified.

#### 2.3.1.1.3 Explanation

Correlation, specifically message correlation, refers to the ability to identify a given message as being *the* message for a particular purpose. In a conversation, it is important to be able to determine that an actual message that has been received is the expected message. Often this is implicit when conversations are relayed over stream-oriented message transports; but not all transports allow correlation to be established so implicitly.

For situations where correlation must be handled explicitly, one technique is to associate a message identifier with messages. The message identifier is an identifier that allows a received message to be correlated with the originating request. The sender may also add an identifier for a service, not necessarily the originating sender, who will be the recipient of the message (see asynchronous messaging).

Correlation may be realized by the underlying protocol; it may be realized by an intermediary between the protocol and the application, ie specification of a conversation ID header; or it may be realized by the application. Application uses the identifier directly and it must be passed through and available to the application. The need for the application to do that depends on how much infrastructure intermediary software is present or not present.

### 2.3.1.2 Intermediary

#### 2.3.1.2.1 Definition

An intermediary is an agent [p.33] that is both a message recipient [p.29] and a message sender [p.30] . An intermediary may process some aspect of the message [p.22] , and acts to forward the message to the next message recipient towards an ultimate message receiver along the message path [p.29] .

#### 2.3.1.2.2 Relationships to other elements

an intermediary is [p.63]

a agent [p.33]

an intermediary may have partial access

to messages [p.22] it processes

an intermediary forwards

the messages [p.22] along the message path [p.29]

#### 2.3.1.2.3 Explanation

Intermediaries process messages and then forward them along the message path. An intermediary is not the ultimate message recipient of a message.

There are two types of intermediaries:

- Forwarding intermediaries: the processing done by these agents was explicitely requested in the message by the message sender.
- Active intermediaries: they are agents that undertake additional processing of the message in ways not described or requested by the message sender. This additional processing may be done without the message sender or receiver’s knowledge, intent or consent. The potential set of services provided by an active intermediary includes, but is not limited to: security services, annotation services, and content manipulation services.

### 2.3.1.3 Message

#### 2.3.1.3.1 Definition

A message is the basic unit of data sent from one software agent to another in the context of Web services.

#### 2.3.1.3.2 Relationships to other elements

a message is [p.63]

a unit of data sent from one agent [p.33] to another a message may be [p.63] part of

a message exchange pattern [p.24]

a message may be described [p.64] using

a message description language [p.28]

a message has [p.65]

a message sender [p.30]

a message has [p.65]

one or more message recipients [p.29]

a message may have [p.65]

a message identifier [p.28]

a message may have [p.65]

a message content

a message may have [p.65]

zero or more message headers [p.27]

a message may have [p.65]

a message envelope [p.24]

#### 2.3.1.3.3 Explanation

A message represents the data structure passed from its sender to its recipients. The structure of message is defined in service descriptions by a message description language.

A message is defined as a construct that can include zero or more headers, an envelope, data within the envelope and data external to the envelope. The header part of a message can include information pertinent to extended Web services functionality, such as security, transaction context, orchestration information, message routing information, or management information. The data part of a message contains the message content or URIs to the actual data resource.

A message can be as simple as an HTTP GET request, in which the HTTP headers are the headers and the parameters encoded in the URL are the content. Note that extended Web services functionality in this architecture is not supported in HTTP headers.

A message can also simply be a plain XML. However, such messages do not support extended Web services functionality defined in this architecture.

A message can be a SOAP XML, in which the SOAP headers are the headers. Extended Web services functionality are supported in SOAP headers.

### 2.3.1.4 Message envelope

#### 2.3.1.4.1 Definition

A message envelope is that meta-data associated with a message that permits the message to be delivered, intact.

#### 2.3.1.4.2 Relationships to other elements

a message envelope is [p.63]

> meta-data associated with a message [p.22]

a message envelope contains [p.65]

> address information about the intended recipients [p.29] of its associated message [p.22]

a message envelope contains [p.65]

> the body of the message.

#### 2.3.1.4.3 Explanation

The message envelope is that information needed to actually deliver messages. It must at least contain sufficient address information so that the message transport [p.30] can deliver the message. Typically this information is part of the service *binding* information found in a WSDL document.

Other meta data that may be present in an envelope includes security information to allow the message to be authenticated and quality of service information.

A correctly design message transport mechanism should be able to deliver a message based purely on the information in the envelope. For example, an encrypted message that fully protects the identities of the sender, recipient as well as the message content, may still be delivered using only the address information (and presumably the encrypted data stream itself).

### 2.3.1.5 Message Exchange Pattern (MEP)

#### 2.3.1.5.1 Definition

A Message Exchanage Pattern (MEP) is a template, devoid of application semantics, that describes a generic pattern for the exchange of messages between agents.

#### 2.3.1.5.2 Relationships to other elements

a message exchange pattern is [p.63]

a template describing a generic pattern for the exchange of messages [p.22] between agents [p.33] .

a message exchange pattern is [p.63]

a feature [p.17] of the architecture

a message exchange pattern should have [p.63]

a unique identifier [p.46]

a message exchange pattern is [p.63]

the life cycle of a message [p.22] exchange

a message exchange pattern describes [p.64]

the temporal and causal relationships, if any, of multiple messages [p.22] exchanged in conformance with the pattern.

a message exchange pattern describes [p.64]

the normal and abnormal termination of any message exchange conforming to the pattern.

a message exchange pattern may realize [p.66]

message correlation [p.20]

a message exchange pattern may describe [p.64]

a service [p.35] invocation

#### 2.3.1.5.3 Explanation

Distributed applications in a Web services architecture communicate via message exchanges. A Message Exchange Pattern (MEP) is a template, devoid of application semantics, that establishes describes a generic a pattern for the exchange of (one-way) messages between agents. These message exchanges are logically factored into patterns that may compose at different levels. These patterns can be described by state machines that indicate define the flow of the messages, including the handling of faults that may arise, and the correlation of messsages.

In order to promote interoperability, it is useful to define common MEPs that are broadly adopted and unambiguously identified. When a MEP is described for the purpose of interoperability, it should be associated with a URI that will identify that MEP.

| **Editorial note** | |
|---|---|
| The following sentence is contentious in the paragraph below: "The exchanges may be synchronous or asynchronous. An asynchronous exchange involves some form of rendezvous to associate the message and its responses, typically due to separate invocations of the underlying transport or to long response time intervals." | |

At the SOAP messaging level, an MEP refers to an exchange of messages in various invoking-response patterns. Each message at this level may travel across multiple transports en route to its destination. A message and its response(s) are correlated, either implicitly in the underlying protocol (e.g., request-response in HTTP) or by other correlation techniques implemented at the binding level. The exchanges may be synchronous or asynchronous. An asynchronous exchange involves some form of rendezvous to associate the message and its responses, typically due to separate invocations of the underlying transport or to long response time intervals.

| **Editorial note** | |
|---|---|
| The following paragraph still has a draft status. | |

MEPs are abstract and must be mapped to a protocol. Some protocols may implicitly support certain MEPs, e.g., HTTP supports request-response. In other cases there is a need for additional glue to map MEPs onto a protocol.

Web service description languages at the level of WSDL view MEPs from the perspective of a particular service actor. A simple request-reponse MEP, for example, appears as an incoming message which invokes an operation and an associated outgoing message with a reply. Extremely simple applications based on single message exchanges may be adequately characterized at the operation level. More complex applications require multiple, related message exchanges.

| **Editorial note** | |
|---|---|
| The following paragraph is contingent on WS Choreography WG input. | |

Choreography describes patterns where the units of communication are themselves instances of MEPs and adds application semantics (choreography = MEPs + application semantics). Especially at this higher level of abstraction, the communicating actors are seen as peers which play various roles in more complex applications. These choreographic patterns form the communication structure of the application.

#### 2.3.1.5.4 Example

| **Editorial note: FGM** | |
|---|---|
| Not sure that the following text belongs here | |

Consider the pattern:

1. agent A uses an instance of an MEP (possibly request-response) to communicate initially with B.
2. agent B then uses a separate, but related instance of an MEP to communicate with C.
3. agent A uses another instance of an MEP to communicate with C but gets a reply only after C has processed (2).

The example makes it clear that the overall pattern cannot be described in terms of the inputs and outputs of any single interaction. The pattern involves constraints and relationships among the messages in the various MEP instances. It also illuminates the fact that exchange (1) is in in-out MEP from the perspective of actor B, and mirrored by an out-in MEP from the perspective of actor A. Finally, an actual application instantiates this communication pattern and completes the picture by adding computation at A, B and C to carry out application-specific operations.

It is instructive to consider the kinds of fault reporting that occur in such a layering. Consider a fault at the transport protocol level. This transport level may itself be able to manage certain faults (e.g., re-tries), but it may also simply report the fault to the binding level. Similarly the binding level may manage the fault (e.g., by re-initiating the underlying protocol) or may report a SOAP fault. The choreography and application layers may be intertwined or separated depending on how they are architected. There is also no rigid distinction between the choreography and binding layers; binding-level MEPs are essentially simple choreographies. Conceptually, the choreographic level can enforce constraints on message order, maintain state consistency, communicate choreographic faults to the application, etc. in ways that transcend particular bindings and transports.

### 2.3.1.6 Message Header

#### 2.3.1.6.1 Definition

A message header is the part of the message that is available to any potential intermediaries and contains information about the message, such as its structure and the identity of the service provider.

#### 2.3.1.6.2 Relationships to other elements

a message header is [p.63] part of

a message [p.22]

a message header may contain [p.65]

message [p.22] routing information

a message header may contain [p.65]

message [p.22] security information

a message header may contain [p.65]

message [p.22] orchestration information

a message header may contain [p.65]

message [p.22] transaction context

#### 2.3.1.6.3 Explanation

The header part of a message can include information pertinent to extended Web services functionality, such as security, transaction context, orchestration information, or message routing information.

### 2.3.1.7 Message description language

#### 2.3.1.7.1 Definition

A message description language allows the structure of messages to be described.

#### 2.3.1.7.2 Relationships to other elements

a message description language describes [p.64]

the structure of a message [p.22]

#### 2.3.1.7.3 Explanation

A message description language allows the formal structure of messages to be described, including the types of elements of the message, how recipients and senders are identified and which headers are associated with messages.

### 2.3.1.8 Message Identifier

#### 2.3.1.8.1 Definition

A message identifier is an identifier [p.46] that uniquely identifies a message.

#### 2.3.1.8.2 Relationships to other elements

a message identifier is [p.63]

a unique identifier [p.46]

a message identifier identifies

a message [p.22]

#### 2.3.1.8.3 Explanation

A message may have an identifier [p.46] . Message identifiers allow messages to be correlated [p.20] within an extended transaction; for example, an event message may reference the original subscription request message.

Message identifiers also support message reliability [p.82] and management and accountability [p.84] of services: by providing the means to uniquely identify messages in an audit trail.

### 2.3.1.9 Message Path

#### 2.3.1.9.1 Definition

A message path is the sequence of agents [p.33] that process a message [p.22] ; starting with the originating sender [p.30] of the message and terminating with the intended recipient [p.29] of the message.

#### 2.3.1.9.2 Relationships to other elements

a message path is [p.63]

a sequence of agents [p.33]

a message path contains

zero or more message intermediaries [p.21] ,

#### 2.3.1.9.3 Explanation

A message path is the sequence of agents that process a message. A message has a unique originator [p.30] and a recipient [p.29] . However, messages may also be processed by a number of intermediaries [p.21] that manage and constrain the message along its path.

### 2.3.1.10 Message recipient

#### 2.3.1.10.1 Definition

A message recipient is an agent [p.33] that receives a message [p.22] .

#### 2.3.1.10.2 Relationships to other elements

a message recipient is [p.63]

a agent [p.33]

a message sender may be [p.63]

an intermediary [p.21]

#### 2.3.1.10.3 Explanation

The message recipient is an agent [p.33] of possibly multiple agents that the message sender [p.30] transmits the message [p.22] to. The message recipient may be identified by its agent identifier [p.46] in a message envelope [p.24] ; however, the agent identifier of the message recipient is not required to be supplied in the case of broadcast-style interactions.

In general, a message may be intended for more than one recipient. Furthermore, in some cases, the sending agent may not have direct knowledge of the identity of the message recipient (for example, in multicast or broadcast situations).

Messages may also be passed through intermediaries [p.21] that process aspects of the message; typically by examining the message headers [p.27] . The message recipient may or may not be aware of processing by such intermediaries.

### 2.3.1.11 Message sender

#### 2.3.1.11.1 Definition

A message sender is the agent that transmits a message [p.22] .

#### 2.3.1.11.2 Relationships to other elements

a message sender is [p.63]

an agent [p.33]

a message sender may be [p.63]

an intermediary [p.21]

#### 2.3.1.11.3 Explanation

A message sender is an agent [p.33] that transmits a message [p.22] to an agent in a message path [p.29] . The message sender may be identified by its agent's identifier [p.46] in a message envelope [p.24] ; however, the agent identifier of the message sender may not be available in the case of anonymous interactions.

Messages may also be passed through intermediaries that process aspects of the message; typically by examining the message headers [p.27] . The sending agent may or may not be aware of such intermediaries [p.21] .

### 2.3.1.12 Message Transport

#### 2.3.1.12.1 Definition

A Message Transport is a mechanism that may be used by agents to deliver messages.

#### 2.3.1.12.2 Relationships to other elements

a message transport is [p.63]

a mechanism that delivers messages [p.22]

#### 2.3.1.12.3 Explanation

The message transport is the actual mechanism used to deliver messages. Examples of message transport include HTTP over TCP, SOAP transport, message oriented middleware, RMI and so on.

The primary responsibility of a message transport is to deliver messages intact. Other responsiblities may include timeliness, privacy, reliability and so on.

For a message transport to function, the sending agent must place the address of the initial recipient in the message envelope. In the case of a message path [p.29] involving intermediaries [p.21] , then the initial recipient is the first intermediary.

### 2.3.1.13 Reliable messaging

#### 2.3.1.13.1 Definition

Reliable messaging is a feature that represents a key infrastructure-level notion of reliability.

#### 2.3.1.13.2 Relationships to other elements

reliable messaging is [p.63]

a feature [p.17]

reliable messaging may be realized [p.66] by

a combination of message acknowledgement and correlation [p.20] .

#### 2.3.1.13.3 Explanation

Reliable messaging is an important contributory factor to the overall reliability of Web services. The goal of reliable messaging is to both reduce the the error frequency for interactions and, where errors occur, to provide a greater amount of information about either successful or unsuccessful attempts at message delivery.

Reliable messaging may be realized with a combination of message receipt acknowlegment and correlation. In the event that a message has not been properly received, the sender may attempt a resend, or some other compensating action. Note that in a distributed system, it theoretically not possible to guarantee correct notification of delivery; however, in practice, simple techniques can greatly increase the overall confidence in the message delivery.

Message correlation may be used by the receivers of messages to ensure that messages are only acted on once - with duplicate messages being ignored or treated as errors.

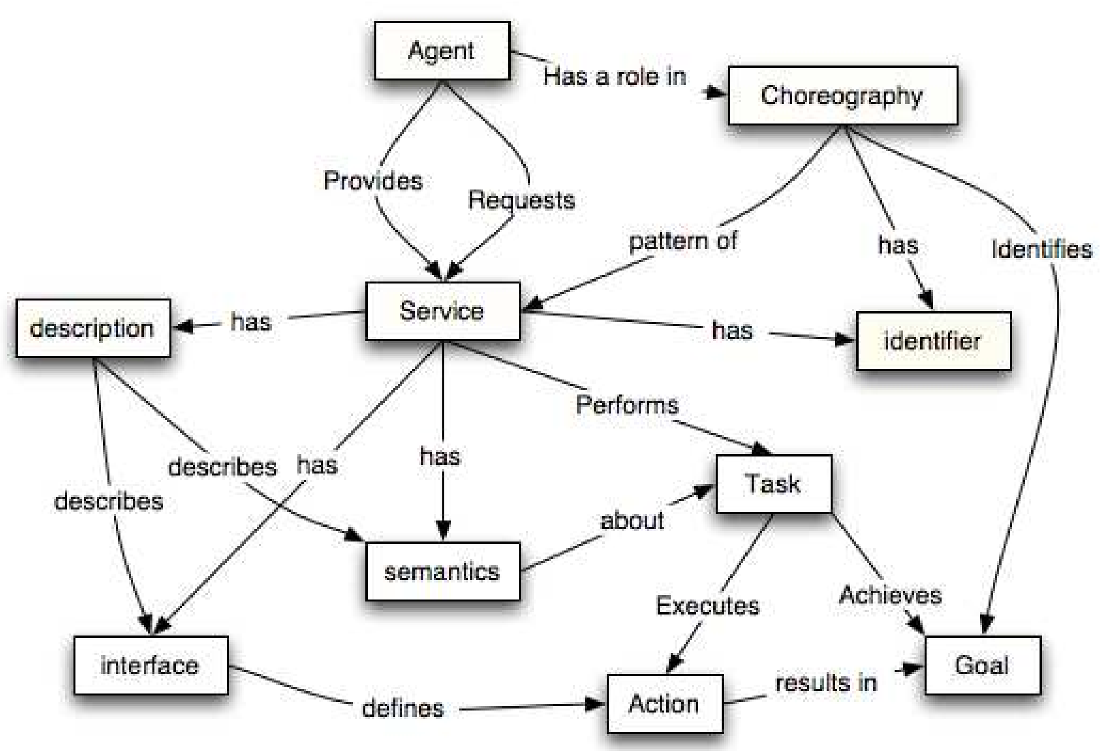
### 2.3.2 The Service Oriented Model

The Service Oriented Model focuses on those aspects of the architecture that relate to Service [p.35] and action [p.32] .

The primary purpose of the SOM is to explicate the relationships between an agent [p.33] , the services [p.35] it offers and requests.

While it is clearly the case that an agent cannot offer or request a service without being able to send and receive messages, the SOM does not mention messages or message transport. The SOM builds on the MOM; but its focus is on action [p.32] rather than message.

The concepts and relationships in the SOM are illustrated in the figure:




#### 2.3.2.1 Action

#### 2.3.2.1.1 Definition

An action, for the purposes of this architecture, is any action that may be performed by an agent [p.33] as a result of receiving a message [p.22] , or results in sending a message [p.22] or other observable state change.

#### 2.3.2.1.2 Explanation

At the core of the concept of service [p.35] is the notion of one party performing action(s) on behalf of another party. A philosophical definition of action involves the application of effort (i.e., application of force) in order to achieve an observable change of state. From the perspective of Web service providers and requesters, an action is typically expressed in terms of executing some fragment of a program.

In the WSA, the actions performed by service providers and requesters are largely out of scope, except in so far as they are the result of messages being sent or received. In effect, the programs that are executed by agents are not in scope of the architecture, however the resulting messages are.

### 2.3.2.2 Agent

#### 2.3.2.2.1 Definition

An agent [p.33] is a program acting on behalf of another person, entity, or process [Web Arch] [p.89] .

#### 2.3.2.2.2 Relationships to other elements

An agent is [p.63]

> a computational resource

An agent has [p.65]

> an identifier [p.46]

An agent has [p.65]

> an owner that is a person or organization [p.53]

An agent may provide [p.39]

> one or more services [p.35]

An agent may request [p.40]

> zero or more services [p.35]

#### 2.3.2.2.3 Explanation

Agents are programs that engage in some set of actions as representatives of other entities. On the Web, for example, agents can seek information. Agents implement services.

Agents are the computational representatives of a person or organization [p.53] . This architecture specifically eschews any attempt to govern the implementation of agents; it is only concerned with ensuring interopability between systems.

### 2.3.2.3 Choreography

#### 2.3.2.3.1 Definition

A choreography defines the sequence and conditions under which multiple cooperating independent Web services exchange information in order to achieve some useful function.

#### 2.3.2.3.2 Relationships to other elements

A choreography is [p.63]

> the pattern of possible interactions between a set of services [p.35]

A choreography may be expressed [p.65] in

> a choreography description language [p.34]

#### 2.3.2.3.3 Explanation

A choreography is model of the sequence of operations, states, and conditions which control how the interactions occur. Successfully following the pattern of interaction prescribed by a choreography should result in the completion of some useful function, for example: the placement of an order, information about its delivery and eventual payment, or putting the system into a well-defined error state.

A choreography is not to be confused with orchestration. An orchestration defines the sequence and conditions in which one Web service invokes other Web services in order to realize some useful function. I.e., an orchestration is the pattern of interactions that a Web service agent [p.33] must follow in order to achieve its goal.

### 2.3.2.4 Choreography Description Language

#### 2.3.2.4.1 Definition

A choreography description language is a notation for describing a choreography [p.34] .

#### 2.3.2.4.2 Relationships to other elements

A choreography description language describes [p.64] the pattern of allowable interactions between a set of services [p.35]

A choreography description language may describe [p.64]

the life cycle of a service invocation [p.35]

A choreography description language describes [p.64]

the conversations possible between service requesters [p.40] and service providers [p.39] .

#### 2.3.2.4.3 Explanation

A choreography description language is focussed on enabling the description of how to interact with services at a larger scale than the individual message exchange pattern. It permits the description of how Web services can be composed, how roles and associations in Web services can be established, and how the state, if any, of composed services is to be managed.

A choreography description language is a formal, machine-processable language for defining specific choreographies [p.34] . It permits the description of how Web services can be used to acheive goals, how roles and associations in Web services can be established, and how the state, if any, of composed services is to be managed.

### 2.3.2.5 Service

#### 2.3.2.5.1 Definition

A service is a set of actions that form a coherent whole from the point of view of service providers [p.39] and service requesters [p.40] .

#### 2.3.2.5.2 Relationships to other elements

a service performs

one or more tasks [p.42]

a service has [p.65]

a service description [p.37]

a service has [p.65]

one or more service providers [p.39]

a service has [p.65]

zero or more service requesters [p.40]

a service has [p.65]

an identifier [p.46]

a service has [p.65]

a service semantics [p.41]

a service has [p.65]

a service interface [p.39]

a service is realized [p.66]

by one or more agents acting as service providers [p.39]

a service is invoked by

exchanging messages [p.22]

a service has [p.65]

a service execution model

**2.3.2.5.3 Explanation**

A service is a collection of related tasks [p.42] that form a coherent whole, from the point of view of service providers and requesters. Critically, services have descriptions that may be formally expressed in one or more service description languages.

The concept of a service is distinct from the software agent that provides the service (and the software agent that requests it). A service refers to the actions performed by the agents rather than the agents themselves.

In the case of atomic or simple services, a service is provided through the actions of a single (possibly federated) software agent [p.33] . In the case of a composite service, the service is provided through the collaboration of a collection of agents. In this latter case, it may not be obvious which software agent is providing the service either to the requester of a service or even to the agents providing the service.

A service has an identifier [p.46] , which in this architecture is a URI. However, the service's identifier should not be construed as identifying any of the agents that perform the tasks offered by the service.

The service description defines the functionality of a service, the service semantics and the interface of the service, i.e how to interact with the service.

The semantics of a service is expressed in terms of the tasks [p.42] that are performed by the service.

In a Service Oriented Architecture [p.10] , the description of a service's interface is expressed in terms of the messages that may be exchanged between service providers and requesters.

**Issue (service_uri):**

**What is identified with a service identifier?**

**Source**:

The Web services architecture builds directly upon the Web architecture [Web Arch] [p.89] . As such, the Web services architecture directly leverages the definition of, and architectural constraints placed on, identifiers from the identifiers from the Architecture of the World Wide Web.

URIs MUST be used to identify all conceptual elements of the system (see Web Services Architecture Requirements: AR009.3).

However, it is not clear what is referenced by a service identifier; is it the service end-points, the service description, the agent?. None of these seems definitive.

**Resolution:**

None recorded.

### 2.3.2.6 Service description

#### 2.3.2.6.1 Definition

A service description is a set of documents that describe the interface to and semantics of a service [p.35] .

#### 2.3.2.6.2 Relationships to other elements

a service description is [p.63]

> a description of a service [p.35]

a service description contains

> a description of the service’s interface [p.39]

a service description may contain

> a description of the service’s semantics [p.35]

a service description contains

> a description of the messages [p.22] that are exchanged by the service [p.35]

a service description has [p.65]

> an identifier [p.46] which is a URI.

a service description is expressed [p.65] in

a service description language [p.37]

#### 2.3.2.6.3 Explanation

A service description contains the details of the interface and implementation of the service. This includes its data types, operations, binding information, and network location. It could also include categorization and other meta data to facilitate discovery and utilization by requesters. The complete description may be realized as a set of XML description documents.

There are many potential uses of service descriptions, they may be used to facilitate the construction and deployment of services, they may be used by people to locate appropriate services and they may be used by service requesters to automatically discover appropriate providers in those case where requesters are able to may suitable choices.

### 2.3.2.7 Service end point

#### 2.3.2.7.1 Definition

A service end point is a network location at which an implementation of a service interface is made available.

#### 2.3.2.7.2 Relationships to other elements

a service end point has [p.65]

an identifier [p.46]

a service end point realizes [p.66]

a service interface [p.39]

#### 2.3.2.7.3 Explanation

| **Editorial note: HH** | |
|---|---|
| Need a definition of binding / bound to | |

An end point is the realization of an interface. This realization can be done using different protocols: the end point is said to be bound to the interface that it realizes.

An interface can be realized by several end points. For example, the same interface could be exposed with SOAP over HTTP, SOAP over SMTP, etc.

### 2.3.2.8 Service interface

#### 2.3.2.8.1 Definition

A service interface is the abstract boundary that a service exposes. It is defined as a set of operations.

#### 2.3.2.8.2 Relationships to other elements

a service interface has [p.65]

one or more service operations [p.39]

#### 2.3.2.8.3 Explanation

A service interface defines the different sets of messages that a service sends and receives.

Like the definition of service operation [p.39] , this definition is abstract. It usually is made available at a service endpoint [p.38] .

### 2.3.2.9 Service operation

#### 2.3.2.9.1 Definition

A service operation is an abstract grouping of messages that a service sends and receives in order to perform a particular task.

#### 2.3.2.9.2 Relationships to other elements

a service operation is [p.63]

an abstract grouping of messages [p.22]

a service operation describes [p.64]

the invocation of a service task [p.42]

#### 2.3.2.9.3 Explanation

A service operation defines the set of messages that a service sends and receives for each task that it performs.

This definition is abstract: it is about the format and order of the messages exchanged, independently from the way it is done, i.e. the protocols used.

### 2.3.2.10 Service provider

#### 2.3.2.10.1 Definition

A Service Provider is an agent that is capable of and empowered to perform the actions associated with a service.

#### 2.3.2.10.2 Relationships to other elements

a service provider is [p.63]

a Web service [p.35] agent [p.33]

a service provider provides

one or more services [p.35]

a service provider performs, or causes to perform

the actions associated with a service [p.35]

#### 2.3.2.10.3 Explanation

The service provider is the agent, i.e., computational entity, that provides a service.

A given service may be offered by more than one agent, especially in the case of composite services, and a given service provider may offer more than one service.

### 2.3.2.11 Service requester

#### 2.3.2.11.1 Definition

A service requester is the entity [p.33] that is responsible for requesting a service from a service provider [p.39] .

#### 2.3.2.11.2 Relationships to other elements

a service requester is [p.63]

a Web service [p.35] agent [p.33]

a service requeser requests

one or more services [p.35]

#### 2.3.2.11.3 Explanation

The service requester is the entity that requires a certain function to be satisfied. From an architectural perspective, this is the agent [p.33] that is looking for and invoking or initiating an interaction with a service.

### 2.3.2.12 Service semantics

#### 2.3.2.12.1 Definition

The semantics of a service is the contract between the service provider [p.39] and the service requester [p.40] that expresses the effect of invoking the service. A service semantics may be formally described in a machine readable form, identified but not formally defined or informally defined via an 'out of band' agreement between the provider and the requester.

#### 2.3.2.12.2 Relationships to other elements

a service semantics is [p.63]

> the contract between the service provider [p.39] and the service requester [p.40] concerning the effects and requirements pertaining to the use of a service [p.35]

a service semantics is about [p.63]

> the service tasks [p.42] that constitute the service.

a service semantics may be expressed [p.65]

> in a service description language [p.37]

a service semantics may be identified

> in a service description [p.37]

a service semantics describes [p.64]

> the intended effects of using a service [p.35]

a service semantics is describes [p.64]

> the relationship between the service provider [p.39] and the service requester [p.40]

#### 2.3.2.12.3 Explanation

Knowing the type of a data structure is not enough to understand the intent and meaning behind its use. For example, methods to deposit and withdraw from an account typically have the same type signature, but with a different effect. Semantics in web services provide formal documentation of meaning. Contracts describing semantics may be used in other web service features such as choreography.

The semantics of a service is fundamentally about the tasks that are encapsulated in the service. A given service may encapsulate a number of different tasks, each task consists of a goal and a process or action for achieving the goal.

#### 2.3.2.13 Service Task

##### 2.3.2.13.1 Definition

A service task is a unit of activity associated with a service. It is denoted by a pair: a goal and an action; the goal denotes the intended effect of the task and the action denotes the process by which the goal is achieved.

##### 2.3.2.13.2 Relationships to other elements

a service task has [p.65]

> a goal that represents the intended effect of the task

a service task has [p.65]

> one or more actions [p.32] that should result in the goal being achieved.

##### 2.3.2.13.3 Explanation

A task is an abstraction that encapsulates the intended effect of invoking a service.

A task is associated with a goal — a predicate that should be satisfied on successful completion of the task

A task is associated with a procedure (or action) that is used to achieve the task.

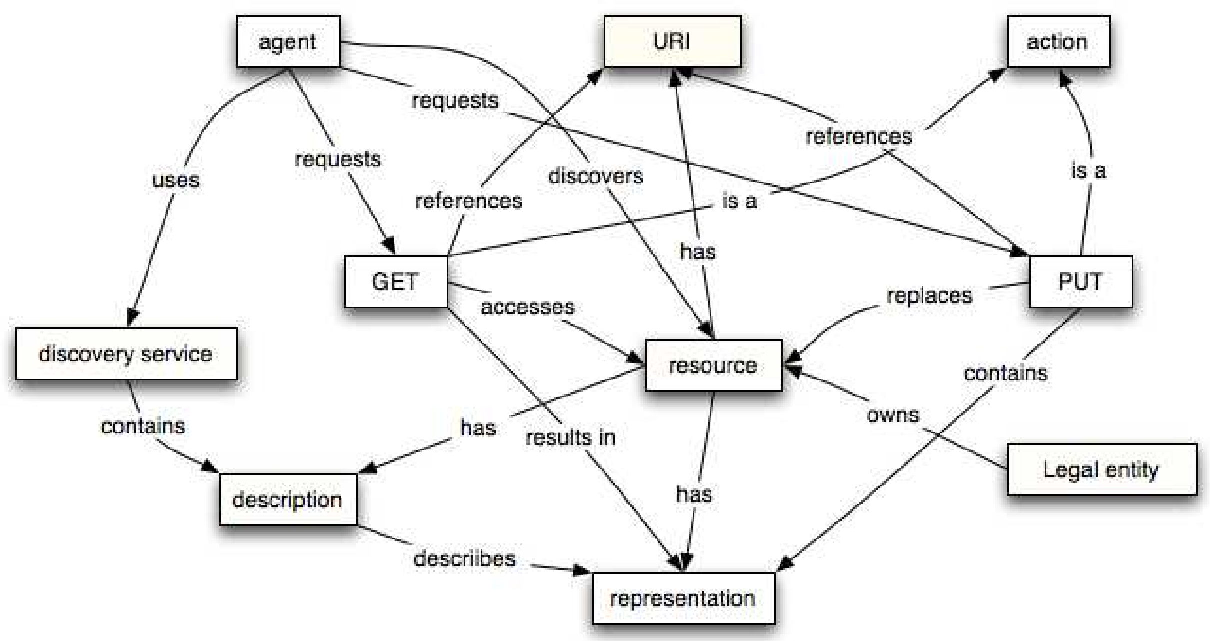
The actions associated with a task may be public, as in the exchange of messages between service providers and requesters; or it may be private, as in the calculation of a formula or in the update of a shared resource. In the case of a service oriented architecture [p.10] only the public aspects of a task are important, and these are expressed entirely in terms of the messages exchanged.

### 2.3.3 The Resource Oriented Model

The Resource Oriented Model focuses on those aspects of the architecture that relate to resources [p.47] , and the service model associated with manipulating resources. It builds on the Service Oriented Model [p.32] , primarily by developing the service model associated with resources and common actions associated with manipulating them.

A key role of this part of the architecture is to explicate the Web itself, and how it relates to Web services. The ROM does so by showing how resources are an independent concept, and yet how the manipulation of resources is an instance of a Service Model [p.32] : with particular kinds of services and identified actions on resources.

The concepts and relationships in the ROM are illustrated in the figure:




### 2.3.3.1 DELETE

#### 2.3.3.1.1 Definition

DELETE is a standard action associated with resources. It is used by an agent to request that a resource be removed and no longer accessible.

#### 2.3.3.1.2 Relationships to other elements

DELETE is [p.63]

> an action [p.32] defined relative to resources [p.47]

A service [p.35] implementing [p.66] DELETE removes

> the resource [p.47] such that its representation [p.47] is no longer available to any subsequent GET [p.45] action.

#### 2.3.3.1.3 Explanation

DELETE is a standard action associated with Web resources. Any service that realizes the Web model must realize the DELETE action — although security and other concerns may still prevent a requestor successfully using DELETE.

Of course, how a resource is DELETEd may vary; some systems may maintain permanent records of all resources; others may completely remove a DELETEd resource. The key change in state after a successful DELETE is that no subsequent GET on the DELETEd resource is successful (accessing an archived resource is conceptually distinct from accessing the resource itself)..

### 2.3.3.2 Discovery

#### 2.3.3.2.1 Definition

Discovery is the act of locating a machine-processable description of a Web service that may have been previously unknown and that meets certain functional criteria. [WS Glossary] [p.89]

#### 2.3.3.2.2 Relationships to other elements

Discovery is [p.63]

an act [p.32]

Discovery may be performed [p.32]

by an agent [p.33] , or by an end-user

Discovery may be realized [p.66]

using a discovery service [p.44]

Discovery may be realized [p.66]

are a direct interaction between the owners of the service requester [p.40] and service provider [p.39]

Discovery is the act [p.32]

of matching a set of functional and other criteria with a set of descriptions.

#### 2.3.3.2.3 Explanation

There are various means by which discovery can be performed. Various entities, e.g. human end users or agents may initiate discovery. Service requesters may find services and obtain binding information (in the service descriptions) during development for static binding, or during execution for dynamic binding. For statically bound service requesters, using service discovery is optional, as a service provider can send the description directly to service requesters. Likewise, service requesters can obtain a service description from other sources besides a service registry, such as a local file system, FTP site, URL, or WSIL document.

### 2.3.3.3 Discovery Service

#### 2.3.3.3.1 Definition

A discovery service is a service that performs discovery; of particular interest are discovery services that permit the discovery of Web services.

#### 2.3.3.3.2 Relationships to other elements

A discovery service is [p.63]

a service [p.35]

A discovery service is used to

publish service descriptions [p.37]

A discovery service is used to

search for service descriptions [p.37]

A discovery service may be accessed

automatically or under human guidance

#### 2.3.3.3.3 Explanation

A discovery service is used by agents and service owners to publish and search for service descriptions.

The discovery of a service takes place at different times in the overall life-cycle of a service. At one extreme, discovery is vestigial as a new service may be built directly in terms of an existing service; at the other extreme, a service requester dynamically searches for a service provider that may fulfil its requirements immediately prior to the actual service initiation.

### 2.3.3.4 GET

#### 2.3.3.4.1 Definition

GET is a standard action associated with resources. It is used by an agent to access a representation of a resource.

#### 2.3.3.4.2 Relationships to other elements

GET is [p.63]

an action defined relative to resources [p.47]

A service [p.35] implementing [p.66]

returns a representation [p.47] of the resource

#### 2.3.3.4.3 Explanation

GET is a standard action associated with Web resources. Any service that realizes the Web model must realize the GET action — although security and other concerns may still prevent a requestor successfully using GET.

### 2.3.3.5 Identifier

#### 2.3.3.5.1 Definition

An identifier is a preferably opaque string of bits that may be used to associate with a resource

#### 2.3.3.5.2 Relationships to other elements

an identifier is [p.63]

> an opaque string of bits

an identifier may be realized [p.66]

> a URI

an identifier identifies

> a resource that is relevant to the architecture

#### 2.3.3.5.3 Explanation

Identifiers are used to identify resources. In the architecture we use Uniform Resource Identifiers [RFC 2396] [p.89] to identify resources.

We have a strong preference that any concrete realization of identifiers does not exhibit any structure. The reason is that an identifier's primary role is to permit multiple references to a resource to be viewed as equivalent.

An opaque string means, in this context, that identifies do not have a structure that is discernible to an outside observer. However, identifiers MAY have internal structure that is relevant to the originator of the identifier; for example, an identifier in a challenge-response interchange may be opaque to the responder but not to the challenger.

### 2.3.3.6 PUT

#### 2.3.3.6.1 Definition

PUT is a standard action associated with resources. It is used by an agent to request that a resource be updated in accordance with a resource representation.

#### 2.3.3.6.2 Relationships to other elements

PUT is [p.63]

> an action defined relative to resources [p.47]

A service [p.35] implementing [p.66] PUT updates resources [p.47] to be consistent with a supplied representation [p.47]

#### 2.3.3.6.3 Explanation

PUT is a standard action associated with Web resources. Any service that realizes the Web model must realize the PUT action — although security and other concerns may still prevent a requestor successfully using PUT.

### 2.3.3.7 Representation

#### 2.3.3.7.1 Definition

A representation is a data object that denotes a resource state, and is the vehicle for conveying the meaning of a resource. A resource is an abstraction for which there is a conceptual mapping to a (possibly empty) set of representations.

#### 2.3.3.7.2 Relationships to other elements

a representation is [p.63]

a data object

a representation denotes

the state of a resource

#### 2.3.3.7.3 Explanation

Representations are data objects that denote the state of a resource. A resource has a unique identifier, whereas a representation is a data object that represents the resource itself. Note, that a representation of a resource need not be the same as the resource itself; for example the resource asociated with the booking state of a restaurant will have different representations depending on when the representation is retrieved.

Resources have identifiers but cannot be retrieved representations of resources may be retrieved but are typically not themselves resources.

### 2.3.3.8 Resource

#### 2.3.3.8.1 Definition

A resource is defined by [RFC 2396] [p.89] to be anything that has an identifier [p.46] .

#### 2.3.3.8.2 Relationships to other elements

a resource is [p.63]

an entity

a resource has [p.65]

an identifier [p.46]

a resource may have [p.65]

zero or more representations

#### 2.3.3.8.3 Explanation

Resources form the heart of the Web architecture itself the Web is a universe of resources that have URIs as identifiers, as defined in [RFC 2396] [p.89] .

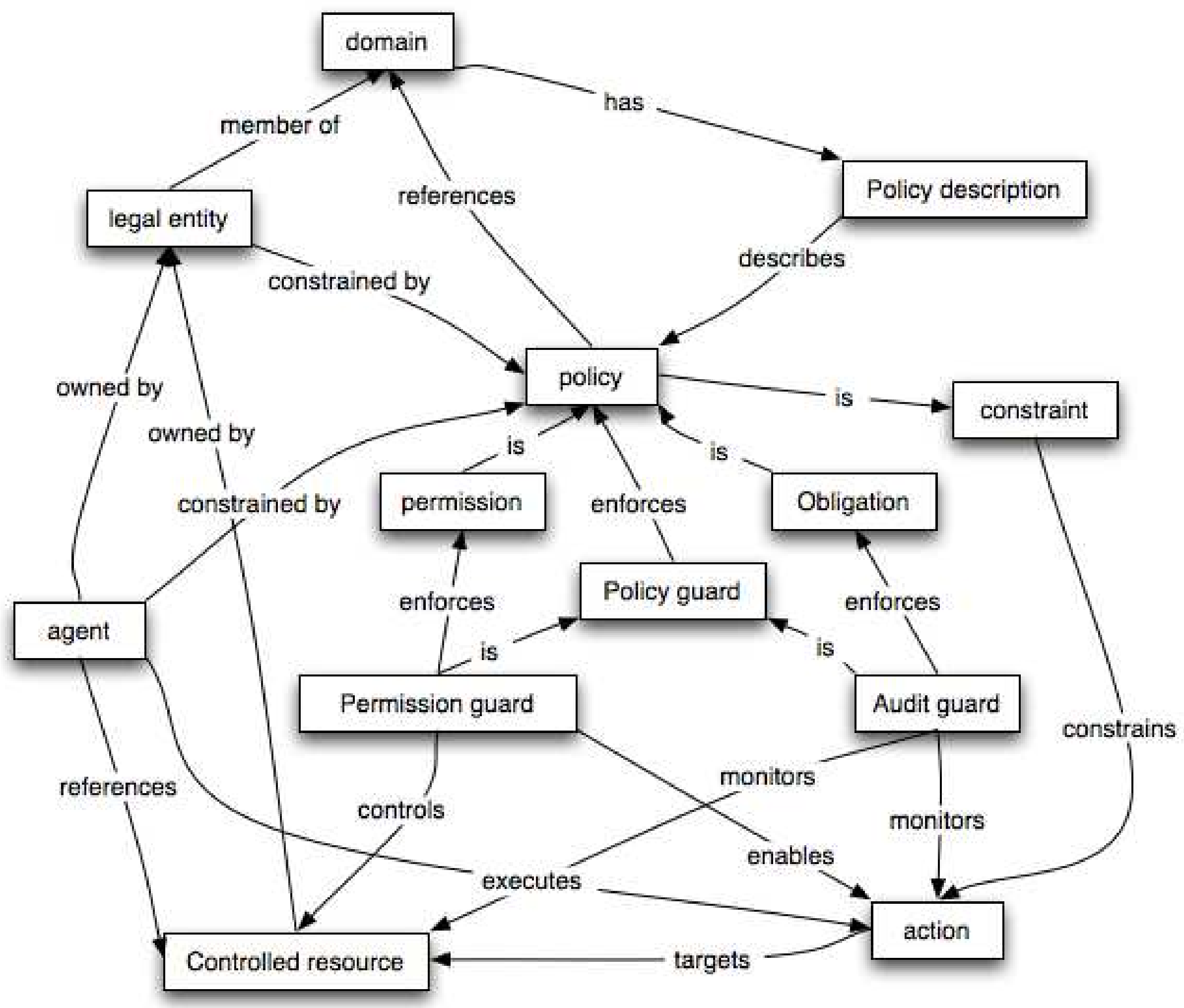
### 2.3.4 The Policy Model

The Policy Model focuses on those aspects of the architecture that relate to policies [p.54] and, by extension, security and quality of service.

Security is fundamentally about constraints; about constraints on the behavior on action and on accessing resources. Similarly, quality of service is also about constraints on service. In the PM, these constraints are modeled around the core concept of policy [p.54] ; and the relationships with other elements of the architecture. Thus the PM is a framework in which security can be realized.

However, there are many other kinds of constraints, and policies, that are relevant to Web services; including various application-level constraints.

The concepts and relationships in the PM are illustrated in the figure:




### 2.3.4.1 Audit guard

#### 2.3.4.1.1 Definition

An audit guard is a mechanism deployed on behalf of an owner that monitors actions and agents to verify the satisfaction of obligations.

#### 2.3.4.1.2 Relationships to other elements

a permission guard is a [p.63]

a policy guard [p.55]

an audit guard is a [p.63]

a mechanism that enforces obligation policies [p.51]

an audit guard may monitor

one or more resources. [p.47]

an audit guard may monitor

actions [p.32] relative to one or more services [p.35] .

an audit guard may determine

if an agent [p.33] 's obligations have been discharged.

#### 2.3.4.1.3 Explanation

An audit guard is an enforcement mechanism; used to monitor the discharge of obligation policies [p.54] . The role of the audit guard is to monitor that agents, resources and services are consistent with any associated obligations established by the service's owner or manager.

Typically, an audit guard sits monitors the state of a resource or a service, ensuring that the obligation is satisfied.

An audit guard acts by either determining if associated obligations are satisfied.

By their nature, it is not possible to proactively enforce obligations; hence, an obligation violation may result in some kind of retribution after the fact of the violation.

### 2.3.4.2 Authentication

#### 2.3.4.2.1 Definition

Authentication is the process of verifying that a potential partner in a conversation is capable of representing a person or organization [p.53]

#### 2.3.4.2.2 Relationships to other elements

Authentication is [p.63]

a feature [p.17] of the architecture

Authentication may be realized [p.66]

using a security authority

#### 2.3.4.2.3 Explanation

### 2.3.4.3 Domain

#### 2.3.4.3.1 Definition

A domain is a set of agents and/or resources that is subject to the constraints of one of more policies. [p.54]

#### 2.3.4.3.2 Relationships to other elements

A domain is [p.63]

> a collection of agents [p.33] and/or resources. [p.47]

A domain defines

> the scope of application of zero or more policies [p.54]

#### 2.3.4.3.3 Explanation

A domain defines the scope of applicability of policies [p.54] . A domain may be *emplicitly* represented, as in the domain of resources managed by a particular management system [p.61] ; or it may be *implicitly* represented, as in the domain of agents [p.33] that talk only to themselves.

### 2.3.4.4 Obligation

#### 2.3.4.4.1 Definition

An obligation is a kind of policy that relates to the required actions and states of an agent and/or resource.

#### 2.3.4.4.2 Relationships to other elements

an obligation is a [p.63]

> type of policy [p.54]

an obligation may require

> an agent [p.33] to perform one or more actions [p.32]

an obligation may require

> an agent or service to be in one or more allowable states

an obligation may be discharged

> by the performance of an action [p.32] or other event.

#### 2.3.4.4.3 Explanation

An obligation is one of the fundamental types of policies [p.54] . When an agent has an obligation to perform some action, then it is required to do so. When the action is performed, then the agent can be said to have satisfied its obligations.

Not all obligations relate to actions; for example, an agent providing a service may have an obligation to maintain a certain state of readiness (quality of service policies are often expressed in terms of obligations). Such an obligation is typically not discharged by any of the obligee’s actions; although an event (such as a certain time period expiring) may discharge the obligation.

Obligations, by their nature, cannot be proactively enforced. However, obligations are associated with enforcement mechanisms: audit guards [p.49] . These monitor controlled resources and agents and may result in some kind of *retribution*; retributions are not modeled by this architecture.

### 2.3.4.5 Permission

#### 2.3.4.5.1 Definition

A permission is a kind of policy that relates to the allowed actions and states of an agent and/or resource.

#### 2.3.4.5.2 Relationships to other elements

a permission is a [p.63]

  type of policy [p.54]

a permission may enable

  one or more actions [p.32]

a permission may enable

  one or more allowable states

#### 2.3.4.5.3 Explanation

A permission is one of the fundamental types of policies [p.54] . When an agent has permission to perform some action, to access some resource, or to achieve a certain state, then it is expected that any attempt to perform the action etc., will be successful. Conversely, if an agent [p.33] does *not* have the required permission, then the action should fail even if it would otherwise have succeeded.

Permissions are enforced by guards, in particular permission guards [p.53] , whose function is ensure that permission policies are honored.

### 2.3.4.6 Permission guard

#### 2.3.4.6.1 Definition

A permission guard is a mechanism deployed on behalf of an owner that enforces permission policies.

#### 2.3.4.6.2 Relationships to other elements

a permission guard is a [p.63]

  a policy guard [p.55]

a permission guard is a [p.63]

  a mechanism that enforces permissive policies [p.52]

a permission guard may control

  one or more resources. [p.47]

a permission guard enables

  actions relative to one or more services. [p.35]

#### 2.3.4.6.3 Explanation

A permission guard is an enforcement mechanism; used to enforce permissive policies [p.54] . The role of the permission guard is to ensure that any uses of a service or resource are consistent with the policies established by the service's owner or manager.

Typically, a permission guard sits between a resource or service and the requestor of the resource or service. In many situations, it is not necessary for a service to be aware of the permission guard. For example, one of the roles of an message intermediary [p.21] is to act as permission guards for the final intended recipient of messages.

A permission guard acts by either enabling a requested action or access, or by denying it. Thus, it is normally possible for permissive [p.52] policies to be proactively enforced.

### 2.3.4.7 Person or organization

| **Editorial note** | |
|---|---|
| The Working Group would like to request feedback on the use of "Person or Organization". A fairly strong minority in the Working Group prefers the concept "Person, entity or process" as in the definition of agent [p.33] . | |

#### 2.3.4.7.1 Definition

A person or organization may be the owner of agents that provide or request Web services.

#### 2.3.4.7.2 Relationships to other elements

a person or organization may be [p.63]

  the owner of an agent [p.33]

a person or organization has [p.65]

  a presence in the physical domain

a person or organization has [p.65]

  a physical address

a person or organization may agree to

  to a legally binding contract

a person or organization has [p.65]

  a name

#### 2.3.4.7.3 Explanation

The WSA concept of person or entity [p.53] is intended to refer to the people that are represented by agents and Web services.

From a architectural perspective, the key difference between an individual and an organization is that the former has no owner or shareholders, and that all actions are ultimately rooted in the actions of humans.

### 2.3.4.8 Policy

#### 2.3.4.8.1 Definition

A policy is a constraint on the behavior of agents.

#### 2.3.4.8.2 Relationships to other elements

a policy is a [p.63]

  constraint on the actions performed by agent [p.33]

a policy is a [p.63]

constraint on the allowable states achieved by agents [p.33]

a policy may have [p.65]

an identifier [p.46]

a policy has [p.65]

an owner that is a person or organization [p.53]

a policy may be described [p.64]

in a machine processable form

a policy may reference

resources, actions and agents

#### 2.3.4.8.3 Explanation

A policy is a machine processable description of some constraint on the behavior of agents as they perform actions, access resources.

There are many kinds of policies, some relate to accessing resources in particular ways, others relate more generally to the allowable actions an agent may perform: both as service providers and as service requestors.

Logically, we identify two types of policy: permissions [p.52] and obligations [p.51] .

Although most policies relate to actions of various kinds, it is not exclusively so. For example, there may be a policy that an agent must be in a certain state (or conversely may not be in a particular state) in relation to the services it is requesting or providing.

Closely associated with policies are the mechanisms for establishing policies and for enforcing them. This architecture does not model the former.

Policies have applications for defining security properties, quality of service properties, management properties and even application properties.

### 2.3.4.9 Policy guard

#### 2.3.4.9.1 Definition

A policy guard is a mechanism deployed on behalf of an owner that enforces a policy (or set of policies).

#### 2.3.4.9.2 Relationships to other elements

a policy guard is a [p.63]

a mechanism that enforces policies [p.54]

a policy guard has a [p.65]

an owner responsible for establishing the guard

#### 2.3.4.9.3 Explanation

A policy guard is an abstraction that denotes a mechanism that is used by owners of resources to enforce policies.

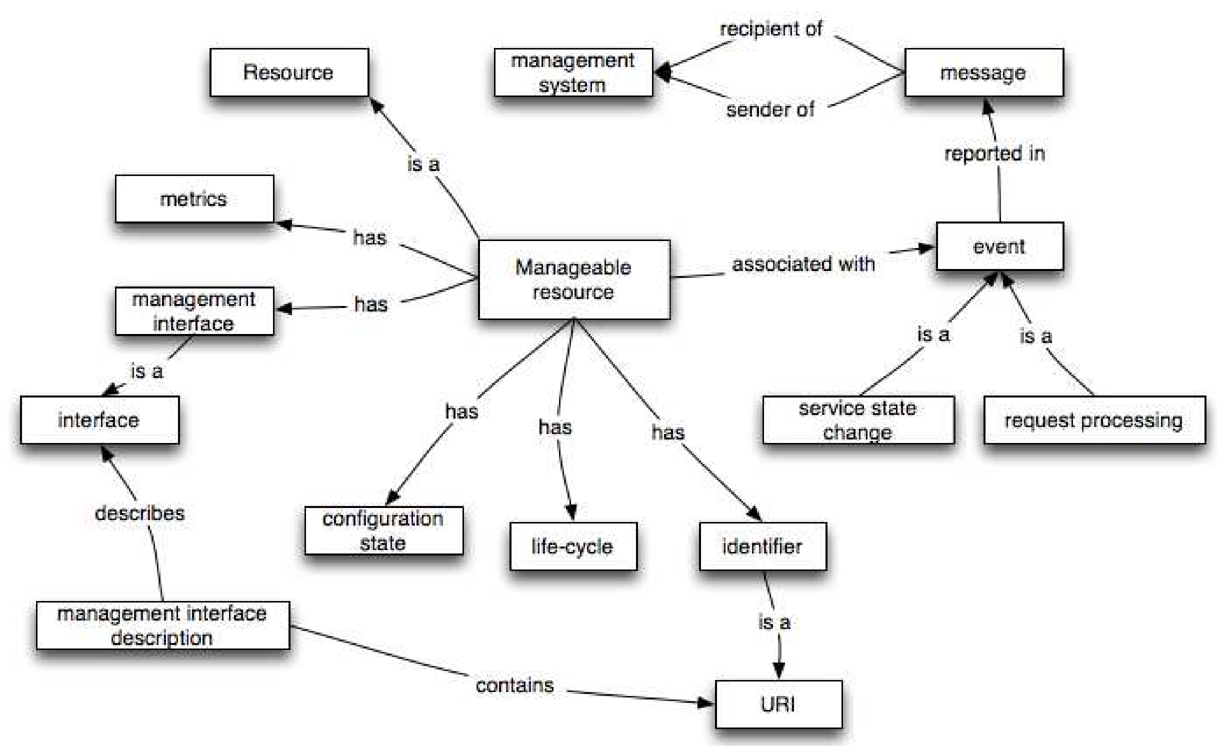
The architecture identifies two kinds of policy guards: audit guards [p.49] and permission guards [p.53] . These relate to the core kinds of policies (obligation and permission policies respectively).

### 2.3.5 The Management Model

The Management model focuses on those aspects of the architecture that relate to the management of Web services; in particular with an emphasis on using the infrastructure offered by Web services to manage the resources needed to deliver that infrastructure.

The core concept that underlies the management model is Deployed element [p.57] . A deployed element is a resource that is, at one level, an actual physically present resource [p.47] that needs to be managed.

The concepts and relationships in the management model are illustrated in the figure:




### 2.3.5.1 Deployed element

#### 2.3.5.1.1 Definition

A deployed resource is a resource that exists in the physical world. There are many kinds of deployed elements, including agents, services and descriptions.

#### 2.3.5.1.2 Relationships to other elements

A deployed element is [p.63]

a physically existing resource

A deployed element has [p.65]

an owner

A deployed element has [p.65]

a physical location, such as a file in a computer or a process executing on a computer.

A deployed element may realize [p.66]

a service [p.35]

A deployed element may be [p.63]

a service description [p.37]

A deployed element may be [p.63]

a managed element [p.61]

#### 2.3.5.1.3 Explanation

Deployed elements are resources that exist physically and are potentially manageable. They are the realization of other elements of this architecture the services, the agents and the descriptions that form the logical aspect of this architecture.

Deployed elements have locations and a physical presence in the real world for example, as a set of files on a disk, or an executing process on a computer.

Deployed elements are the unit of manageability; they have owners and may have other properties that are subject to management.

### 2.3.5.2 Life cycle

#### 2.3.5.2.1 Definition

A life cycle is a set of states and transition paths for an element.

#### 2.3.5.2.2 Relationship to other elements

a life cycle is [p.63]

the set of externally observable states that an entity can be in, together with the transitions between them.

a Web service may have [p.65]

an explicit description of its life cycle

a life cycle of a service [p.35] may be managed

by a manager [p.61]

a life cycle of a service [p.35] may be described [p.64]

in a manageability interface [p.62]

#### 2.3.5.2.3 Explanation

The life cycle of a Web service is a key target for Web service management capabilities.

### 2.3.5.3 Management capability

#### 2.3.5.3.1 Definition

Management capabilities are the capabilities required by a manager to be able to effectively manage.

#### 2.3.5.3.2 Relationship to other elements

the status of a manageable element [p.61] is [p.63]

a management capability

the configuration of a manageable element [p.61] is [p.63]

a management capability

the events associated with a manageable element [p.61] are [p.63]

management capabilities

the operations on a manageable element [p.61] are [p.63]

a management capability

Management capabilities may be described [p.64]

in a manageability interface [p.62] description

#### 2.3.5.3.3 Explanation

The key managability capabilities include the ability to identify a manageable element, the ability to acquire status information about a manageable element, the ability to configure it and the ability to control its life-cycle.

### 2.3.5.4 Management configuration

#### 2.3.5.4.1 Definition

A management configuration is a collection of properties of a manageable elements which may be changed.

#### 2.3.5.4.2 Relationship to other elements

a management configuration is [p.63]

> a set of property values that denotes a particular state of a manageable element [p.61]

#### 2.3.5.4.3 Explanation

Setting a property may influence the behavior of a manageable element. Configuration mechanisms that are common for each type of manageable element [e.g. Web service endpoint] should conform to the Web services architecture [e.g. description and interaction]. Configuration for a manageable element, beyond the common configuration, should be defined by an administrative interface.

### 2.3.5.5 Management event

#### 2.3.5.5.1 Definition

Events are changes in the state of a manageable element that are relevant to the element's manager.

#### 2.3.5.5.2 Relationship to other elements

a management event is [p.63]

> an event that denotes an event associated with a manageable element [p.61] that is relevant to a manager [p.61] .

a problem event is [p.63]

> a management event

a life-cycle event is [p.63]

> a management event

a state change event is [p.63]

> a management event

a request processing event is [p.63]

> a management event

#### 2.3.5.5.3 Explanation

Event descriptions are messages that indicate a problem, a lifecycle state change, or a state change. For a manageable element there are two classes of events: State Changed and Request Processing events. State change events occur whenever lifecycle state transitions happen. A request processing event description indicates that the state of a request has been changed and should include the previous state and the transition time. The event may also include any context associated with the request, reply, or failure message and any part or the complete content of these messages. Event descriptions provide valuable information for managers and can be used to calculate many of the metrics.

### 2.3.5.6 Manager

#### 2.3.5.6.1 Definition

A manager is an entity that is capable of and has an interest in managing manageable elements.

#### 2.3.5.6.2 Relationship to other elements

a manager is [p.63]

  an agent [p.33]

a manager manages

  a manageable element [p.61]

a manager uses

  a manageability capabilities [p.59]

#### 2.3.5.6.3 Explanation

A manager is an agent that is able to manage a set of manageable elements and has the interest and authority in so doing. Managers use the manageability capabilities and interfaces to control the life cycle and other properties of manageable elements on behalf of the owners of those elements.

A manager may have additional capabilities than those within the scope of this architecture; for example, a manager may be able to directly control the computational resources that are used to offer Web services. This architecture focuses on those aspects of management that are central to Web services.

### 2.3.5.7 Manageable Element

#### 2.3.5.7.1 Definition

A manageable element is a deployed element that is manageable. I.e., a physically existing resource that is capable of being managed. A key attribute of manageable elements is that they provide an interface to allow their management.

#### 2.3.5.7.2 Relationships to other elements

a manageable element is [p.63]

  deployed element [p.57]

a manageable element has [p.65]

a life cycle [p.58]

a manageable element has [p.65]

a manageability interface [p.62]

a manageable element has [p.65]

a description [p.64] of its manageability interface [p.62]

a manageable element is

discoverable [p.44]

#### 2.3.5.7.3 Explanation

Manageable elements expose an interface that permits their state — especially their life cycle state to be monitored and potentially modified. The simplest such modification being to delete the element.

There are many potentially manageable elements in this architecture, including services, agents and descriptions as well as discovery providers and service requesters and providers.

As with Web services themselves, manageable elements may also be discovered using similar mechanisms. It is expected that discovery of manageable elements would permit managers to determine if an element is manageable.

### 2.3.5.8 Manageability Interface

#### 2.3.5.8.1 Definition

A manageability interface is a description of the means by which a management system can manage a manageable element.

#### 2.3.5.8.2 Relationships to other elements

a manageability interface has [p.65]

controls

a manageability interface has [p.65]

identification information

a manageability interface has [p.65]

events that correspond to the life cycle [p.58] of a manageable element

a manageability interface has [p.65]

metrics

a manageability interface has [p.65]

status

a manageability interface has [p.65]

configuration controls

#### 2.3.5.9 Management metric

##### 2.3.5.9.1 Definition

Management metrics are raw atomic unambiguous information. For manageability metrics, the information is for managmement purposes.

##### 2.3.5.9.2 Relationship to other elements

a manageability metric is [p.63]

a raw atomic point of data

##### 2.3.5.9.3 Explanation

The value of the metric captures the information at a point in time. Generally these values are numeric, but may be strings as well. This can be contrasted with Measurements that are calculated with a formula based on metrics, e.g. Average response time during the last hour of execution. The metrics requirements do not enforce any implementation pattern. A managed element should allow any available metrics and measurements to be reported according to configurable time intervals, such as cumulative, sliding window, and interval. A managed element must declare which interval types are supported.

## 2.4 Relationships

The relationships between concepts in the architecture are laid out in this section. These relationships represent the core modeling concepts used in the architecture itself.

### 2.4.1 The *is a* relationship

#### 2.4.1.1 Summary

The *X* is a *Y* relationship denotes the relationship between concepts *X* and *Y*, such that every *X* is also a *Y*.

#### 2.4.1.2 Relationships to other elements

Assuming that *X* is a *Y*, then:

true of

if *P* is true of *Y* then *P* is true of *X*

contains

if *Y* has a [p.65] *P* then *X* has a [p.65] *Q* such that *Q* is a [p.63] *P*.

transitive

if *P* is true of *Y* then *P* is true of *X*

### 2.4.1.3 Description

Essentially, when we say that concept *X* is a *Y* we mean that every feature of *Y* is also a feature of *X*. Note, however, that since *X* is presumably a more specific concept than *Y*, the features of *X* may also be more specific variants of the features of *Y*.

For example, in the service [p.35] concept, we state that every service has an identifier. In the more specific Web service [p.35] concept, we note that a Web service has an identifier in the form of a URI identifier.

## 2.4.2 The *describes* relationship

### 2.4.2.1 Summary

The concept *X* is described by *Y* relationship denotes that *Y* is an expression of some language *L* and that the value of *Y* is an instances of *X*.

### 2.4.2.2 Relationships to other elements

Assuming that *X* is described by *Y*, then:

valid

if *Y* a valid expression of *L*, then the value of *Y* is an instance of concept *X*

### 2.4.2.3 Description

Essentially, when we say that concept *X* is described by *Y* we are saying that the expression *Y* denotes instances of *X*.

For example, in the service description [p.37] concept, we state that service descriptions are expressed in a service description language. That means that we can expect legal expressions of the service description language to be instances of the service description concept.

## 2.4.3 The *is expressed in* relationship

### 2.4.3.1 Summary

The concept *X* is expressed in *L* relationship denotes that instances of *X* are values of the language *L*.

### 2.4.3.2 Relationships to other elements

Assuming that *X* is expressed in *L*, then:

valid

> if *E* a valid expression in *L* then *E* is an instance of concept *X*

### 2.4.3.3 Description

When we say that concept *X* is expressed in *L* we use the language *L* to express instances of *X*.

For example, in the service description [p.37] concept, we state that service descriptions are expressed in a service description language. That means that we can expect legal expressions of the service description language to be instances of the service description concept.

## 2.4.4 The *has a* relationship

### 2.4.4.1 Summary

The concept *X* has a *Y* relationship denotes that every instance of *X* is associated with an instance of *Y*.

### 2.4.4.2 Relationships to other elements

Assuming that *X* has a *Y*, then:

valid

> if *E* is an instance of *X* then *Y* is valid for *E*.

### 2.4.4.3 Description

When we say that concept *X* has a *Y* we mean that whenever we see an *X* we should also see a *Y*

For example, in the Web service [p.35] concept, we state that Web services have URI identifiers. So, whenever we the Web service concept is found, we can assume that the Web service referenced has an identifier. This, in turn, allows implementations to use identifiers to reliably refer to Web services. If a given Web service does not have an identifier associated with it, then the architecture has been violated.

### 2.4.5 The *realized* relationship

#### 2.4.5.1 Summary

The concept *X* is realized as *Y* relationship denotes that the concept *X* is an abstraction of the concept *Y*. An equivalent view is that the concept *X* is implemented using *Y*.

#### 2.4.5.2 Relationships to other elements

Assuming that *X* is realized as *Y*, then:

implemented

if *Y* is present, or true of a system, then the concept *X* applies to the system

reified

*Y* is a reification of the concept *X*.

#### 2.4.5.3 Description

When we say that the concept or feature *X* is realized as *Y*, we mean that *Y* is an implementation or reification of the concept *X*. I.e., if *Y* is a valid concept of a system then we have also ensured that the concept *X* is valid of the same system.

For example, in the correlation [p.20] feature, we state that message correlation requires that we associate identifiers with messages. This can be realized in a number of ways — including the identifier in the message header, message body, in a service binding and so on. The message identifier is a key to the realization of message correlation.

# 3 Stakeholder's perspectives

In this section we examine how the architecture [p.16] meets the Web services requirements. We present this as a series of stakeholders' viewpoints; the objective being that, for example, security represents a major stakeholder's viewpoint of the architecture itself.

| **Editorial note** | |
|---|---|
| When developing a particular stakeholder's viewpoint, one should make sure that the concepts discussed are properly documented in the architecture. | |

## 3.1 Web integration

Goal AG003 of the Web Services Architecture Requirements [WSA Reqs] [p.89] identifies Web services must be consistent with the current and evolving nature of the World Wide Web.

This goal can be divided into two sub-goals relating to the architectural principles of the Web [Web Arch] [p.89] and, more pragmatically, relating the architecture to the various technologies in use.

Critical Success Factor AC011 notes that the architecture should be consistent with the architectural principles and design goals of the Web. For our purposes we use the Architecture of the World Wide Web [Web Arch] [p.89] as our reference for the architecture of the Web. It defines the architecture of the Web to be founded on a few basic concepts: agents that are programs that represent people, identification of resources using URIs representations of resources as data objects and interaction via standard protocols — most notably of course HTTP.

## 3.2 Information and service

Unlike the World Wide Web in general, it is of the essence that Web services, like service oriented architectures in general, are essentially about the provision of action. I.e., whereas the World Wide Web is a networked information system, the Web service World Wide Web is a networked service system: information is exchanged between Web service agents [p.33] for the purpose of requesting and provisioning service [p.35] , not simply information.

This is a key specialization of the World Wide Web in general; and it drives a number of the specific features of this architecture. However, it is also the case that requests for action and the various possible responses are also information and have representations.

The key representational requirement of this architecture is that messages exchanged between Web service agents is encoded in XML. This is consistent with the general principles of the World Wide Web [Web Arch] [p.89] .

## 3.3 Web service agents

This architecture also uses the concept of an agent [p.33] to identify the computational resource actually involved with Web services. The key properties of agents required to model Web services are that:

1. It is a computational activity
2. It has an owner
3. It engages in message exchanges with other Web service agents, those messages counting as equivalent messages between the agents' owners.
4. It uses standard protocols to:
    a) describe the form and semantics of messages
    b) describe the legal sequences of messages
5. The distinction between a service requester and service provider is one of the agent's roles; i.e., it is not intrinsic to the concept of agent that it is bound to be solely a provider or requester of services. Of course, in many cases, particular agents will be bound to particular functions that will fix the role of the agent to be a service provider or requester.

However, it is clear that there is a strong correspondence between a Web service agent and a Web agent. Both are computational resources that represent people; our definition requires sufficient detail to be able to account for the greater degree of indirection expected between Web service agents and Web agents.

## 3.4 Web Service Discovery

| **Editorial note** | |
|---|---|
| Based on Web Services Architectural Roles | |

Before a requester agent and a provider agent can interact, the corresponding enties that own them must first agree on the service description and semantics that will govern the interaction, as depicted in Figure 1. There are many ways this can be achieved. Some require discovery, others don't.

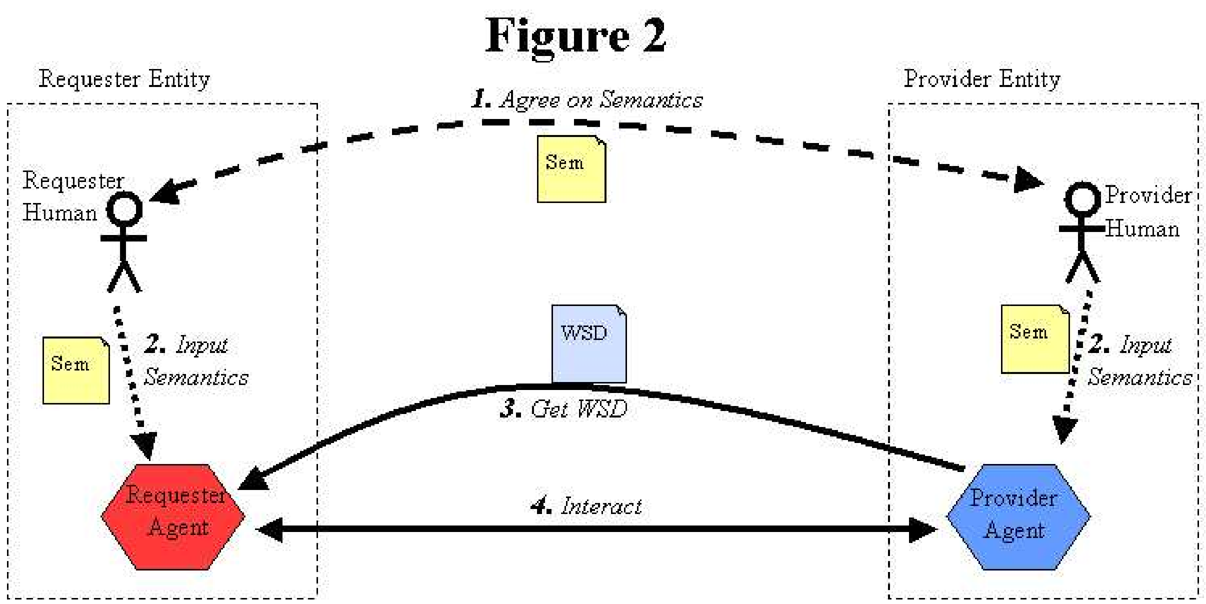
### 3.4.1 Scenarios Not Requiring Discovery

If the requester and provider entities are already known to each other, then one common way for them to agree on the service description and semantics is for a requester human and a provider human to communicate directly. For example, the provider human might send the proposed service description and semantics directly to the requester human. Or the parties might develop them collaboratively. In these situations, there is no need for discovery.

If the provider entity is furnishing the service description unilaterally, then a variation on this approach is for the requester entity to retrieve the service description dynamically from the provider agent, at the start of their interaction, as depicted in Figure 4: Parties Known, Dynamically Getting WSD. This allows the requester agent to be assured of using the latest version that the provider agent supports, again without requiring discovery.

[Figure 5: Parties Known, Dynamically Getting WSD]

| **Editorial note** | |
|---|---|
| dbooth: The label at the top of this diagram needs to be fixed. It should be "Figure 5" instead of "Figure 2". | |




### 3.4.2 Scenarios Requiring Discovery

On the other hand, if the requester entity does not already know what provider agent it wishes to engage, then it may need to "discover" a suitable candidate. Discovery is "the act of locating a machine-processable description of a Web service that may have been previously unknown and that meets certain functional criteria." [WS Glossary] [p.89] Two common ways to approach this are human discovery, and autonomous selection.

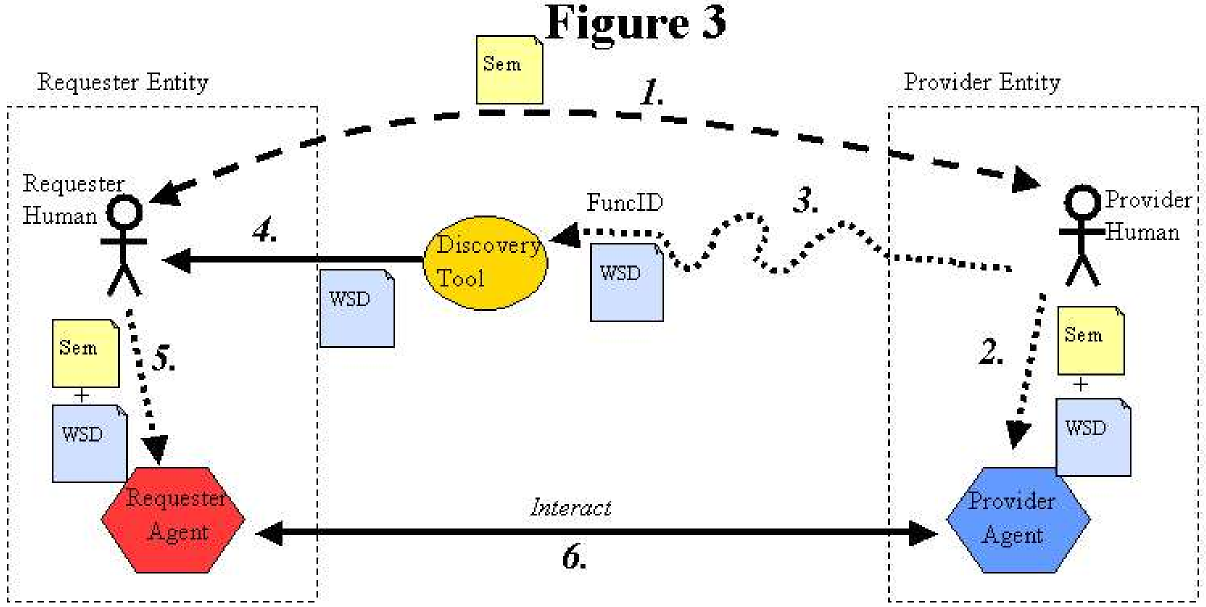
#### 3.4.2.1 Human Discovery

With human discovery, a requester human uses some kind of discovery tool or agent to help locate a suitable service description, i.e., a description representing a service that meets the desired functional criteria, as shown in Figure 6: Human Discovery.

[Figure 6: Human Discovery]

| **Editorial note** | |
|---|---|
| dbooth: The label at the top of this diagram needs to be fixed. It should be "Figure 6" instead of "Figure 3". | |

**Figure 3**




There are several points to note about this situation.

- Regardless of how the service description is obtained, somehow the requester and provider entities must agree on the semantics of the planned interaction. There are several ways this can be done, and the WSA does not specify or care what way is used. For example, the provider entity may publish both the service description and semantics on a take-it-or-leave-it basis, which the requester must accept unmodified as a condition of engaging the provider agent. Or the parties could negotiate the desired semantics. Or the semantics might be defined by an industry standards body that both parties have chosen to follow.

- Somehow the discovery agent must obtain both the service description (or at least a reference to it), and sufficient information (labeled "FuncID" in Figure 6) to describe or identify the semantics of the service that the provider entity offers. The FuncID is necessary to allow the requester human to find a service having the desired semantics. In practice, the FuncID might be as simple as a few words or a URI, or it may be more complex, such as a TModel (in UDDI) or a collection of RDF, DAML-S or OWL statements.

- The WSA also does not specify or care how the discovery agent obtains the service description and FuncID. For example, if the discovery agent is implemented as a search engine (such as Google), then it might crawl the Web, collecting service descriptions wherever it finds them, with the provider entity having no knowledge of it. Or, if the discovery agent is implemented as a registry (such as UDDI), then the provider entity could publish the service description and FuncID directly to the discovery agent.

-

| **Editorial note** | |
|---|---|
| dbooth: Need to add mention of the trust decision here -- the fact that making use of a previously unknown service involves a significant decision to trust that service. | |

| **Editorial note** | |
|---|---|
| dbooth to finish explaining the diagram in this section | |

### 3.4.2.2 Autonomous Selection

With autonomous selection, the requester agent uses a selection agent to select a service from among several known services, as shown in Figure 7: Autonomous Selection.

There are five important roles involved in Autonomous Selection:

- Requester Human, Provider Human, Requester Agent and Provider Agent, as described before; and
- Selection Agent, which is a software application or component that may be operated by the Requester Entity, the Provider Entity or a third party entity.

The following artifacts or documents are relevant in Autonomous Selection:

- The WSD, as previously described.

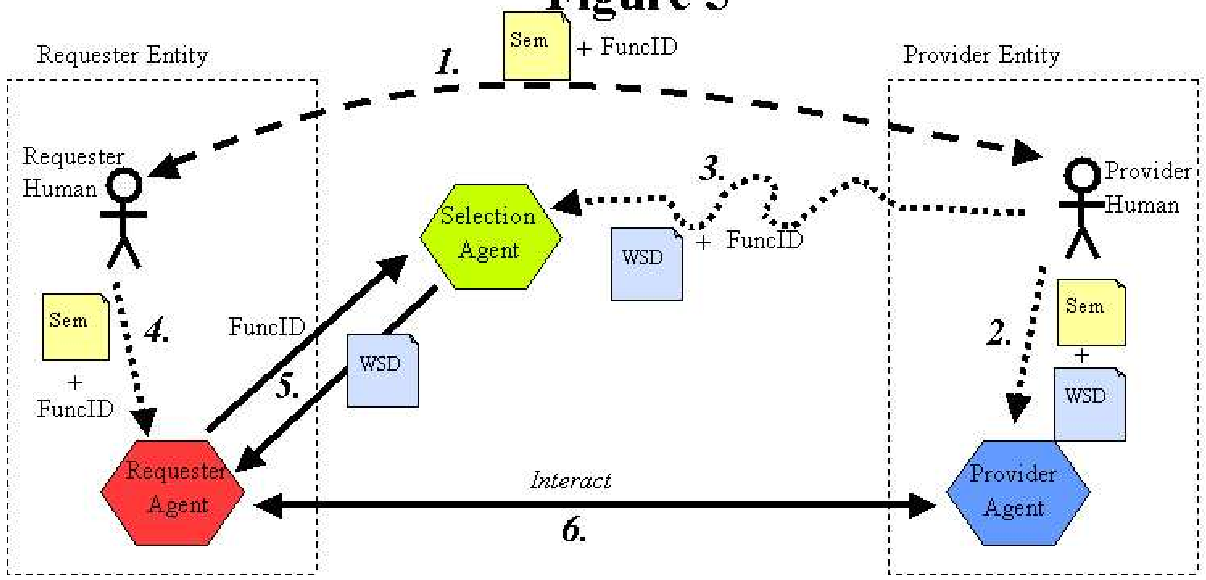
  The Semantics, as previously described.

  A FuncID, which represents any information that is sufficient to allow the Requester Agent to unambiguously identify the Semantics. The FuncID could be represented by the entire Semantics. In practice, the FuncID is expected to be represented by a URI, a TModel, an RDF description, or some other information that is sufficient to unambiguously identify the Semantics.

[Figure 7: Autonomous Selection]

| **Editorial note** | |
|---|---|
| dbooth: The label at the top of this diagram needs to be fixed. It should be "Figure 7" instead of "Figure 5". | |

**Figure 5**




In general, Autonomous Selection involves the following steps. Although the ordering of these steps may vary somewhat, all are important.

1. Agree on Semantics. Again, the requester human and the provider human must somehow agree on the Semantics of the service. In addition, the parties need to agree on what FuncID will be used to identify the Semantics or functionality of the service, so that it can be referenced later.

2. Input Semantics and WSD to Provider Agent. The Provider Human somehow creates the WSD, and inputs the WSD and the Semantics into the Provider Agent.

3. Selection Agent gets WSD and FuncID. The Selection Agent somehow obtains the WSD and FuncID from the Provider Human.

4. Input Semantics to Requester Agent. The Requester Human inputs the desired Semantics and FuncID to the Requester Agent.

5. Select WSD. The Requester Agent uses the FuncID to query the Selection Agent for an acceptable WSD document that corresponds to the desired Semantics.

6. Interact. The Requester Agent and the Provider Agent interact using whatever means they have agreed upon.

There are two key differences between Autonomous Selection and the previously described "Human Discovery":

- The service selection is made without human intervention.

- The selection is restricted to services that are already known and trusted by the requester entity.

In theory, there is no need for these two characteristics to be bundled together. (I.e., one could operate a requester agent that would autonomously find and engage previously unknown services.) But in practice autonomous service selection is often limited to previously known (and trusted) services, because engaging a service almost always involves some degree of risk -- for example, the risk of transmitting funds or sensitive information to another party.

For this reason, Autonomous Selection is most applicable when the decision to engage the service involves little risk, such as a PDA looking for the nearest printer service in an office. For services involving greater risk (such as those involving the purchase of goods or transmission of credit card numbers), it is less likely that the requester entity would be willing to delegate this trust decision to an autonomous agent to engage a previously unknown service.

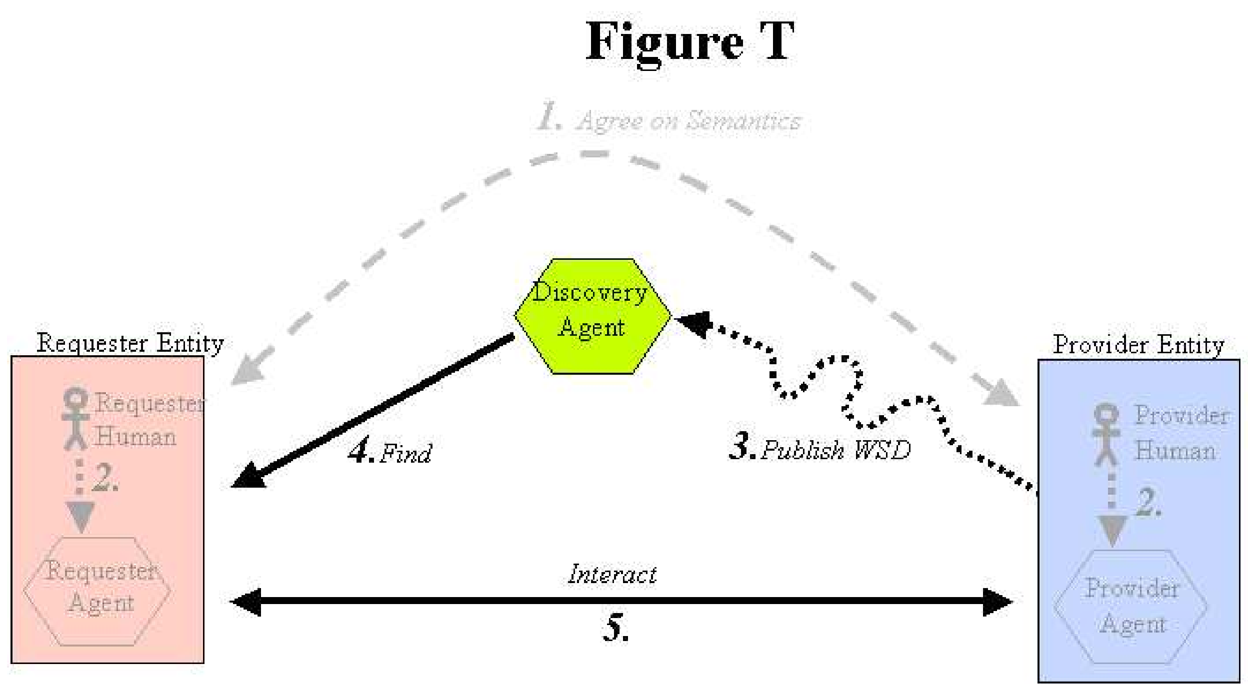
| **Editorial note** | |
|---|---|
| dbooth to finish this section | |

### 3.4.2.3 Triangle Diagram

This section describes the correspondence between the "Triangle Diagram" previously included in our architecture document and the more detailed way that discovery is now described in this document.

[Figure 8: Triangle Diagram]

| **Editorial note** | |
|---|---|
| dbooth: The label at the top of this diagram needs to be fixed. It should be "Figure 8" instead of "Figure T". | |




| **Editorial note** | |
|---|---|
| dbooth to finish this section | |

## 3.5 Web service semantics

For computer programs to successfully interact with each other a number of conditions must be established:

1. There must be a physical connection between them, such that data from one process may reach another
2. There must be agreement on the *form* of the data — such as whether the data is lines of text, XML structures, etc.
3. The two (or more) programs must share agreement as to the intended meaning of the data. For example, whether the data is intended to represent an HTML page to be rendered, or whether the data represents the current status of a bank account; the expectations and the processing involved in processing the data is different — even if the form of the data is identical.

As we shall see below, more is required, but for now this list is sufficient.

The architecture addresses the first two of these requirements in the Message oriented Model [p.20] . That model focuses on how Web service agents (requesters and providers) may interact with each other using a message oriented communication model. The form of the messages is XML, and hence any Web service agent is expected to be able to process XML data.

The intention, or semantics, of the communications between Web service agents is *partially* addressed in the Service Oriented Model [p.32] and the Resource Oriented Model [p.42] .

The Service Oriented Model [p.32] builds on the basics of message communication by adding the concept of action [p.32] and service [p.35] . Essentially, the service model allows us to interpret messages as requests for actions and as responses to those requests. Within the architecture, and in particular using technologies such as WSDL, a Web service can be described in a machine readable document as to the forms of expected messages, the datatypes of elements of messages and — using a choreography description language — the expected flows of messages between Web service agents.

The Resource Oriented Model [p.42] extends this further by adding the concepts of Resource [p.47] , together with a minimal set of standard actions on resources. The Resource Oriented Model [p.42] mimics that of the World Wide Web itself which regards HTML pages as representations [p.47] of resources [p.47] (Web pages in Web sites) that may be accessed using the standard actions (GET [p.45] , PUT [p.46] , etc.) of the HTTP protocol.

However powerful the Resource Oriented Model [p.42] model is, it is not sufficient, in general, to capture the semantics of all Web service interactions. That is because the interactions between Web service agents is considerably richer than simple resource management. There is currently a gap in the architecture relating (sic) to the generalized issues of the semantics of interapplication communication. However, we can postulate certain features that we might expect to see in a more complete accounting of the semantics of Web services:

- It should be possible to identify the real-world entities referenced by elements of messages. For example, when using a credit card to arrange for the purchase of goods or services, the element of the message that contains the credit card information is fundamentally a reference to a real-world entity: the account of the card holder.

  We expect that the Semantic Web technologies will be a sound starting point for addressing this requirement.

- It should be possible to identify the expected effects of any actions undertaken by Web service requesters and providers. That this cannot be captured by datatyping can be illustrated with the example of a Web service for withdrawing money from an account as compared to depositing money (more accurately, transferring from an account to another account, or vice versa). The datatypes of messages associated with two such services may be identical, but with dramatically different effects — instead of being paid for goods and services, the risk is that one’s account is drained instead.

  We expect that a richer model of services, together with technologies for identifying the effects of actions, is required. Such a model is likely to incorporate concepts such as contracts (both legally binding and technical contracts) as well as ontologies of action.

- Finally, a Web service program may "understand" what a particular message means in terms of the expected results of the message, but, unless there is also an understanding of the relationships of the owners of the Web service requester and provider agents, the Web service provider (say) may not be able to accurately determine whether the requested actions are *warranted*.

For example, a Web service provider may receive a valid request to transfer money from one account to another. The request being valid in the sense that the datatypes of the message are correct, and that the semantic markers associated with the message lead the provider to correctly interpret the message as a transfer request. However, the transaction may still not be valid, or fully comprehensible, unless the Web service provider can properly identfy the relationship of the requester to the requested action. Currently, such concerns are often treated simply as security considerations — which they are — in an ad hoc fashion. However, when one considers issues such as delegated authority, proxy requests, and so on, it becomes clear that a simple authentication model cannot accurately capture the requirements.

As with semantics as a whole, there is a gap in the architecture relating to this level of semantic descriptions. However, we expect that a model that formalizes concepts such as institutions, roles (in business terms), "regulations" and regulation formation will be required. With such a model we should be able to capture not only simple notions of authority, but more subtle distinctions such as the authority to delegate an action, authority by virtue of such delegation, authority to authorize and so on.

This architecture encourages precision of semantic description by ensuring that the various aspects of the semantics of the information exchanged between agents can be properly *identified* — as opposed to being fully described. Where appropriate, and possible, it also identifies a number of description languages which can be used to describe different semantic aspects of this exchanged information.

An important technology for realizing the description of the semantics of Web services is the Semantic Web. "The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation." -- Tim Berners-Lee, James Hendler, Ora Lassila, The Semantic Web, Scientific American, May 2001.

| **Editorial note: fgm** | |
|---|---|
| More justice needs to be made to the technologies coming out of the Semantic Web effort. | |

In summary, Web services can be considered to be a specialization of World Wide Web in general; a specialization that reflects the intended purpose of the exchanged information to be about services. Similarly, this architecture can be viewed as an elaboration of the general WWW architecture; albeit with a significant number of additional concepts.

## 3.6 Web services security

| **Editorial note: fgm** | |
|---|---|
| This is a working draft of a proposed section on Security for the stakeholder’s section. | |

Goal AG004 of the WSA requirements states that "The WSA must provide a secure environment for online processes". This is further analyzed into two critical success factors: security (AC006) and privacy (AC020). [WSA Reqs] [p.89]

Security, and privacy, issues are often extremely detailed and often quite implementation specific. For example, there are many famous cases of security being compromised through the incorrect implementation of fixed-length buffers (so-called buffer overruns permit an intruder to corrupt a target's address space and sometimes to gain unauthorized access). This architecture is necessarily at a level of abstraction where such considerations are moot; since implementation itself is out of scope for the architecture.

In the context of security, the prime concerns in the development of the WSA are the identification threats to security and what architectural features are necessary to respond to those threats.

### 3.6.1 Threats to security and privacy

Security issues tend to revolve around access and use of resources; the primary task being to ensure that it is not possible for intruders to access resources for which they do not have the appropriate rights. We can summarize the principal security threats as:

1. Inappropriate access on behalf on unauthorized entities. It should be possible to reliably determine the identity of entities such as agents, service providers, resources and so on.

2. Corruption of communication and resources. It should be possible to ensure the data integrity of communications and transactions.

3. Information leaking. It should be possible to ensure that information is accessible only to intended parties. This certainly includes the content of messages; but may also include the mere fact that a communication between particular parties has taken place.

4. Inappropriate access to resources. It should be possible to ensure that resources and actions are not possible for entities not properly authorized. Again, this often extends to the right to even see that a given resource exists: unauthorized persons may not even be permitted to know the existence of certain resources or certain actions.

5. Denial of service. It should not be possible for parties to prevent legitimate parties to access resources and perform actions that they have the right to.

Note that certain theoretical limitations will prevent us from ever guaranteeing that these risks do not become realities. However, it is certainly a viable goal to significantly reduce the risk of security breaches.

There are other security-related threats that this architecture is not designed to combat:

1. Non-repudiation. The risk associated with this is that a party may subsequently deny its involvement with a transaction. Non-repudiation is important; however, this architecture regards this threat as primarily an application-level threat.

2. Mis-information. A malicious party may attempt to corrupt a Web service by deliberately feeding it incorrect information. For example, by communicating invalid credit card information, a fraudulent Web service requester may attempt to gain service that it would not normally have the right to. This is similar to the buffer overrun style of security breach; however, generally this security breach involves sending well-formed but false information.

3. Copy and replay. Given a copy of a Web service agent, it may be possible to execute it in an environment that could give a malicious party information not ordinarily available. For example, given a copy of a Web service requester agent that acts on behalf of a customer, a malicious Web service provider may simulate a series of transactions with the captured agent in order to determine the maximum price the end user is expected to pay for goods and services.

Privacy issues tend to revolve around the use of personal information, in particular the abuse of personal information; again, this can often be expressed in terms of the wrong people having access to the wrong information. We can summarize the threats to privacy as:

1. Information use. An end user may have the right to know how, when, and to what extent their personal or sensitive information will be used by the Web services processing nodes. Protected usage includes the sharing of personal or sensitive information obtained by a processing node with any third party. These rights are often founded on legislation that varies on a global basis.

2. Confidentiality. Similar to above security threat: third party access access to sensitive information represents a threat to the privacy of the end user.

Also central is that these practices should be exposed by the processing nodes prior to a service invocation, allowing a service requestor to factor a processing node's privacy practices in the decision to use a particular Web service or to follow a particular message route. Hence, the publishing and accessibility of a Web service processor's privacy practices will aid an end user to retain control over his personal information. This is contingent on the compliance to the published privacy by the Web service processor and is outside of the scope of technology solutions.

### 3.6.2 Policies

The approach adopted in the WSA to address both security and privacy concerns revolves around the concept of policy [p.54] . A policy is a document, preferably machine-processable, that expresses some constraint on the behavior of the overall system; generally on the behavior of agents. Closely connected to the concept of policy is the concept of policy guard [p.55] . A policy guard is some mechanism that is used to enforce policy. Hence security is expressed in terms of the policies that the owners of resources [p.47] wish to enforce, together with mechanisms put in place to enforce those policies. Similarly, privacy is expressed in terms of the policies that the owners of data -- typically the users of Web services -- have, together with mechanisms necessary to ensure that the owners' rights are respected.

While many policies relate to actions, and to resources, it is not always the case. Many security and privacy-related constraints are concerned with maintaining certain kinds of state. For example, a Web service provider may have a constraint that any P3P tags associated with a use of one of its Web services are appropriately propagated to third parties. Such a constraint cannot easily expressed in terms of the allowed actions that the Web service provider may perform; it is really an obligation to ensure that the publicly observable condition (the proper use of P3P tags) is always maintained (presumably maintained in private also). Similarly, a Web service provider may (will) link the possible actions that a Web service requester may perform to the Web service requester maintaining a particular level of secure access (e.g., administrative tasks may only be performed if the request is using secure communications).

Policies can be logically broken down into two main types: permissive policies and obligatory policies. A permissive policy concerns those actions and accesses that entities are permitted to perform and an obligation policy concerns those actions and states that entities are required to perform. These are closely related, and dependent: it is not consistent to be obliged to perform some action that one does not have permission to perform. A given policy document is likely to contain a mix of obligation and permission policy statements.

The two kinds of policies have different enforcement mechanisms: a permission guard is a mechanism that can be used to verify that a requested action or access is permitted; an audit guard can only verify after the fact that an obligation has not been met. The precise form of these guards is likely to vary, both with the resources being controlled and with the implementation technologies deployed. The architecture is principally concerned with the existence of guards and their role in the architecture. In a well engineered system it may be possible to construct guards that are not directly visible to either the requester nor the provider of Web services.

A permission guard acts as a guard enabling or disabling action to a resource or action. In the context of SOAP, for example, one important role of SOAP intermediaries is that of permission guards: the intermediary may not, in fact, forward a message if some security policy is violated.

An audit guard acts as a monitor; watching resources and agents, validating that obligations that have been established are respected and/or discharged. Due to the nature of obligations it is often not possible to prevent obligations; instead the focus is on observing that obligations are respected. If an audit guard detects a policy violation, then it normally cannot prevent the violation; instead some form of retribution or remediation must be enacted. The precise forms of this are, of course, beyond the scope of this architecture.

### 3.6.2.1 Policies and security

The threats enumerated above can be countered by a combination of suitable mechanisms and policy documents that govern the enforcement mechanisms. For example, the unauthorized access threat may be countered by a mechanism that validates the identity of potential agents who wish access the controlled resource. That mechanism is, in turn, controlled by the policy document which expresses what evidence must be offered by which agents before the access is permitted.

Not all guards are active processes. For example, confidentiality of information is encouraged (we hesitate to claim guaranteed) by encryption of messages. As noted above, it is potentially necessary to encrypt not only the content of SOAP messages (say) but also the identities of the sender and receiver agents. The guard here is the encryption itself; although this may be further backed up by other active guards that apply policy.

### 3.6.2.2 Policies and privacy

Privacy policies are typically much more of the obligatory form than access control policies. A policy that requires a Web service provider to properly propagate P3P tags, for example, represents an obligation on the provider. It is not possible to prevent a rogue Web service provider from leaking private information; it should be possible, however, to monitor the public actions of the Web service to ensure that the tags are propagated.

### 3.6.3 Policies beyond security

Policies have application beyond security and privacy. For example, many Quality of Service requirements can be expressed in terms of policies (especially obligation-style policies). Furthermore, many application-level policies may also apply. Thus the fundamental concepts of policies and guards may also benefit Web services applications as well as helping to ensure their security.

It should be noted that the focus of the Architecture is those elements that are needed to ensure that policies are adhered to. Currently, the architecture does not address the issues associated with enacting policies; except in so far that enactment can be modeled in terms of Web services that are themselves subject to policies.

The issues involved with enacting policies include determining how policies are established, in particular who has the right to establish a policy and who does a policy relate to. This can be modeled in terms of meta-policies; however, issues of legal responsibility, relationships between agents and between owners. These are beyond the scope of the Architecture.

## 3.7 Scalability and extensibility

According to [WSA Reqs] [p.89] , it is a major goal of the architecture that any implementations are inherently scalable and extensible.

This goal is broken down into five critical success factors:modularity,extensibility,simplicity,migration from EDIand peer to peer.

## 3.8 Modularity

The critical success factor AC002focuses on the modularity of the architecture; with appropriate granularity. This is reduced to an overall conceptual integrity with appropriate decomposition and easy comprehension.

Our architecture is laid out using the simple style of concepts [p.18] and relationships [p.63] . This modeling technique is simple, and yet allows us to expose the critical properties of Web services. A major design goal of the architecture has been the appropriate separation of concerns. In general, this is achieved by rigorous minimalism associated with each concept: only associating those properties of a concept that are truly necessary to meet the requirements.

The overall themes in this architecture can be summarized as:

- Web services are used and presented by agents [p.33] interacting on behalf of real-world actors.
- Message structures [p.28] , service interfaces, conversations [p.34] are first of all explicitly identified and potentially described [p.64] using using a variety of description languages. This has the effect of documenting the various aspects involved in two or more interacting Web services.

- Minimal assumptions about required components. For example, although Web services may be documented, it is not required. Similarly, although descriptions may be published, that is also not required.

## 3.9 Extensibility

## 3.10 Peer to peer interaction

To support Web services interacting in a peer to peer style, the architecture must support peer to peer message exchange patterns, must permit Web services to have persistent identity, must permit descriptions of the capabilities of peers and must support flexibility in the discovery of peers by each other.

In the message exchange pattern [p.24] concept we allow for Web services to communicate with each other using a very general concept of message exchange. Furthermore, we allow for the fact that a message exchange pattern can itself be identified -- this permits interacting Web service agents to explicitly reference a particular message pattern in their interactions.

A Web service wishing to use a peer-to-peer style interaction may use, for example, a publish-subscribe form of message exchange pattern. This kind of message exchange is just one of the possible message exchange patterns possible when the pattern is explicitly identifiable.

In the agent [p.33] concept we note that agents have identifiers [p.46] . The primary role of an agent identifier is to permit long running interactions spanning multiple messages. Much like correlation, an agent's identifier can be used to link messages together. For example, in a publish and subscribe scenario, a publishing Web service may include references to the Web service that requested the subscription, separately from and additionaly to, the actual recipient of the service.

The agent [p.33] concept also clarifies that a given agent may adopt the role of a service provider [p.39] and/or a service requester [p.40] . I.e., these are roles of an agent, not necessarily intrinsic to the agent itself. Such flexibility is a key part of the peer to peer mode of interaction between Web services.

In the service [p.35] concept we state that services have [p.65] a semantics [p.41] that may be identified in aservice description [p.37] and that may be expressed [p.65] in a service description language [p.37] . This identification of the semantics of a service, and for more advanced agents the description of the service contract itself, permits agents implementing Web services to determine the capabilities of other peer agents. This is turn, is a critical success factor in the architecture supporting peer-to-peer interaction of Web services.

Finally, the fact that services [p.35] have descriptions [p.37] means that these descriptions may be published in discovery agencies [p.44] and also retrieved from such agencies. In effect, the availability of explicit descriptions enables Web services and agents to discover each other automatically as well as having these hard-coded.

## 3.11 Long running transactions

In CSF AC017 are identified two requirements that support applications in a similar manner to traditional EDI systems: reliable messaging and support for long-running stateful choreographed interactions. This architecture supports transactions by allowing messages to be part of message exchanges and extended choreographies. It also permits support for message reliability.

## 3.12 Conversations

Conversations are supported in this architecture at two levels: the single use of a Web service and the combination of Web services.

A message exchange pattern [p.24] is defined to be the set of messages that makes a single use of a service. Typical examples of message exchange pattern are request-response, publish-subscribe and event notification.

The details of the message exchange pattern may be documented in a service description [p.37] expressed in a service description language [p.37] such as WSDL.

In addition, the architecture supports the correlation [p.20] of messages by permitting messages to have identifiers.

Web services may be combined into larger scale conversations by means of choreographies [p.34] . A choreography is the documentation of the combination of Web services, leaving out the details of the actual messages involved in each service invocation and focusing on the dependencies between the Web services.

Of particular importance, both to individual message exchange patterns and combined services, is the handling of exceptions.

| **Editorial note** | |
|---|---|
| Please also consider Mark Jones's input and Geoff Arnold's stuff on synchronous/asynchronous | |

## 3.13 Message reliability

Critical Success Factor AC017 of the requirements notes that the architecture must satisfy the requirements of enterprises wishing to transition from traditional EDI and more specifically AR017.1 requires that the architecture must support reliable messaging.

The goal of reliability is to both reduce the the error frequency for interactions and, where errors occur, to provide a greater amount of information about either successful or unsuccessful attempts at service.

In the context of Web services, we can address the issues of reliability at three distinct levels: of reliable and predictable interactions between services, of the reliable and predictable delivery of infrastructure services and of the reliable and predictable behavior of individual service providers and requesters. This analysis is generally separate from concerns of fault tolerance, availability or security, but there may of course be overlapping issues.

The architecture addresses the requirements for the highest level of reliability identified here by accomodating the descriptions of the choreographies of the interactions between Web service requesters, providers. In effect, reliability at this level becomes a measurable property of the descriptions of choreographies: in effect, assuming that the infrastructure is reliable, and assuming that the services are reliable, do the descriptions of the choreographies describe situations which will behave in predictable ways?

The reliability of the individual service providers and requesters is out of scope of this architecture as we do not comment on the realization of Web services. However, reliability at this level is often enhanced by service providers adopting deployment platforms that have strong management capabilities.

The reliability of the infrastructure services refers to the reliability of the messaging infrastructure and the discovery infrastructure; the former is often referred to as reliable messaging. In general, this refers to a predictable quality of service related to the delivery of the messages involved with the Web service.

In more detail, we identify two properties of message sending that are important: the sender of the message would like to be able to determine whether a given message has been received by its intended receiver and that the message has been received exactly once.

Knowing if a message has been received correctly allows the sender to take compensating action in the event the message has not been received. At the very least, the sender may attempt to resend a message that has not been received.

The general goal of reliable messaging is to define mechanisms that make it possible to achieve these objectives with a high probability of success in the face of inevitable but unpredictable network, system and software failures.

The goals of reliable messaging can be made more explicit by considering the issues related to multiple receptions of a message and message intermediaries. If there is an intermediary, does the sender want to know whether the message got to the intermediary or to the intended end recipient? Does the receiver care whether it receives a message more than once? The following classification of reliable messaging expectations is taken from [ebXML MSS] [p.89] .

| | Duplicate-Elimination | Ack Requested From End Receiver | Ack Requested from Next Receiver | Comment |
|---|---|---|---|---|
| 1 | Y | Y | Y | Once-And-Only-Once End-To-End, At Least Once to Intermediate |
| 2 | Y | Y | N | Once-and-only-Once End-to-End based on end-to-end retransmission |
| 3 | Y | N | Y | At-Least-Once at the intermediate level, Once-and-only-once end-to-end if all the intermediaries are reliable, no end-to-end notification |
| 4 | Y | N | N | At-Most-Once end-to-end, no retries at the intermediate level |
| 5 | N | Y | Y | At-Least-Once with duplicates possible both end-to-end and at intermediate level |
| 6 | N | Y | N | At-Least-Once with duplicates possible both end-to-end and at intermediate level |
| 7 | N | N | Y | At-Least-Once to the intermediaries and the end. No end-to-end notification |
| 8 | N | N | N | Best Effort |

The goals of reliable messaging may also be examined with respect to whether one wishes to confirm only the receipt of a message, or perhaps also to confirm the validity of that message. Three questions may be asked about message validity:

1. Was the message received the same as the one sent? This may be determined by such techniques as byte counts, check sums, digital signatures.

2. Does the message conform to the formats specified by the agreed upon protocol for the message? Typically determined by automatic systems using syntax constraints (eg xml well formed) and structural constraints (validate against one or more xml schemas or WSDL message definitions).

3. Does the message conform to the business rules expected by the receiver? For this purpose additional constraints and validity checks related to the business process are typically checked by application logic and/or human process managers.

Of these, first and second are considered to be part of reliable messaging, the last is partly addressed by Web service choreography.

Message reliability is most often achieved via an acknoweldgement infrastructure, which is a set of rules defining how the parties to a message should communicate with each other concerning the receipt of that message and its validity. WS-Reliability and WS-ReliableMessaging are examples of specifications for an acknowledgement infrastructure that leverage the SOAP Extensibility Model. In cases where the underlying transport layer already provides reliable messaging support (e.g. a queue-based infrastructure), the same reliability Feature can be achieved in SOAP by defining a binding that relies on the underlying properties of the transport.

## 3.14 Web service manageability

Goal AG007 of the requirements [WSA Reqs] [p.89] identifies that manageability of Web services is an important goal of this architecture. Since the architecture defines how to define information, operations and discovery of Web services, it is consistent to use Web services to provide access to and manageability of Web services also.

Management in this case is defined as a set of capabilities for discovering the existence, availability, health, and usage, as well the control and configuration of manageable elements, where these are defined as Web services, descriptions, agents of the Web services architecture, and roles undertaken in the architecture.

This architecture does not attempt to specify completely how Web services are managed; that is be the role of a separate specification. The architecture does, however, identify the key concepts and relationships involved in modeling manageability. These key concepts include the manageable element [p.61] , its management capabilities [p.59] , the manageability interface [p.62] and the manager [p.61] .

For example, an executing Web services agent [p.33] is a potentially managable element that may require management, as is an actually deployed Web service [p.35] and the Web service's service description [p.37] .

The manager [p.61] is an agent [p.33] that has the responsiblity for managing on behalf of its owner and the owners of the resources that it is managing. The manager uses the manageability interface [p.62] to aquire metrics [p.63] of managed elements and to manage the configuration [p.59] , the life cycle [p.58] , and to monitor the status of those elements. The manager [p.61] is also a prime recipient of management events [p.60] .

The key relationship that ensures that the architecture models management appropriately is the realizes [p.66] relationship. The entities under management the manageable elements [p.61] have a realizes [p.66] relationship with other elements of the architecture. For example, a Web service [p.35] is provided by an agent [p.33] . Both the Web service itself and the agent are realized [p.66] in some fashion; as are any descriptions of the service; as physically deployed resources, and those deployed resources are themselves potentially manageable.

As with Web services themselves, it may be important for scalability reasons for managers [p.61] to be able to automatically discover [p.44] both the manageable elements [p.61] it may be responsible for and their manageability interfaces [p.62] .

| **Editorial note** | |
|---|---|
| Discovery has a management aspect as well as a service aspect. This needs to be made clearer. | |

Of course, managers that are deployed in order to help the management of Web services are also potentially subject to management; however, to the extent that such managers are already modeled as Web services, their management will be handled as any other Web service.

## 3.15 Web services technologies

There are a number of technologies that are in widespread use in the deployment of Web services; and other technologies that will arise in the future. In this section we describe some of those technologies that seem critical and the role they fill in relation to this architecture. This is a necessarily bottom-up perspective, since, in this section, we are looking at Web services from the perspective of tools which can be used to design, build and deploy Web serivces.

The technologies that we consider here, in relation to the Architecture, are XML, SOAP, WSDL. SOAP provides an extensible framework for the XML data that is interchanged. The format that SOAP defines has restrictions and places of extensibility. The Web Services Description Language (WSDL) provides a format for defining the allowable formats for messages to and from agents. These include SOAP, XML, MIME, and simple HTTP requests.

| **Editorial note: fgm** | |
|---|---|
| There are almost certainly others we should mention here, such as BPEL, WS-Security, WS-Policy, WS-Reliability, ... | |

### 3.15.1 XML and Web services

As previously stated, a Web services interaction is two, or more, software agents exchanging information in the form of messages. The data that is exchanged is usually XML carried over an underlying transport or transfer protocol, such as HTTP. Similarly, XML is also the foundation for many of the descriptive technologies — such as SOAP, WSDL, OWL, and many others. In effect, the use of XML is critical to the overall picture of Web services.

The reason for the importance of XML is that it solves a key technology requirement that appears in many places: by offering a standard, flexible and inherently extensible data format, XML significantly reduces the burden of deploying the many technologies needed to ensure the success of Web services.

The important aspects of XML, for the purposes of this Architecture, are the core syntax itself, the concepts of the XML Infoset [XML Infoset] [p.90] , XML-Schema and XML-Namespaces.

XML Infoset is not a data format per se, but a formal set of information items and their associated properties that comprise an abstract description of an XML document [XML 1.0] [p.90] . The XML Infoset specification provides for a consistent and rigorous set of definitions for use in other specifications that need to refer to the information in a well-formed XML document.

Serialization of the XML Infoset definitions of information MAY be expressed using XML 1.0 [XML 1.0] [p.90] . However, this is not an inherent requirement of the architecture. The flexibility in choice of serialization format(s) allows for broader interoperability between agents in the system. In the future, a binary encoding of XML should be a suitable replacement for the textual serialization — such a binary encoding may be more efficient and more suitable for machine-to-machine interactions.

Many of the uses of XML in the architecture relate to the description of machine processable elements; such as message formats, interface descriptions, choreography descriptions, policy descriptions and so on. While XML-Schema is not sufficiently powerful (nor intended to be) to capture all these, it does provide a strong typing foundation for other kinds of description. Hence, XML-Schema is an integral part of XML — and Web services, as far as this archicture is concerned.

Similarly, XML-Namespaces is also a critical part of XML technology in relation to the Architcture. It allows developers to partition all the different types of XML document that a large-scale Web service installation is likely to need.

### 3.15.2 SOAP

SOAP Version 1.2 is a simple and lightweight XML-based mechanism for creating structured data packages that can exchanged between network applications.

SOAP is the standard for XML messaging for a number of reasons. First, SOAP is relatively simple, defining a thin layer that builds on top of existing network technologies such as HTTP that are already broadly implemented. Second, SOAP is flexible and extensible in that rather than trying to solve all of the various issues developers may face when constructing Web services, it provides an extensible, composable framework that allows solutions to be incrementally applied as needed. Thirdly, SOAP is based on XML. Finally, SOAP enjoys broad industry and developer community support.

[SOAP 1.2 Part 1] [p.89] defines an XML-based messaging framework: a processing model and an exensibility model. SOAP messages can be carried by a variety of network protocols; such as HTTP, SMTP, FTP, RMI/IIOP, or a proprietary messaging protocol.

[SOAP 1.2 Part 2] [p.89] defines three optional components: a set of encoding rules for expressing instances of application-defined data types, a convention for representing remote procedure calls (RPC) and responses, and a set of rules for using SOAP with HTTP/1.1.

An extension of the SOAP messaging framework is called a SOAP feature. One special type of SOAP feature is the Message Exchange Pattern (MEP). A SOAP MEP is a template that establishes a pattern for the exchange of messages between SOAP nodes. Examples of MEPs include: request/response, oneway, peer-to-peer conversation, etc. A MEP MAY be supported by one or more underlying protocol binding instances either directly, or indirectly with support from software that implements the required processing to support the SOAP Feature as expressed as a SOAP Module.

### 3.15.3 WSDL

WSDL is an [p.63] example of implementation technology [p.66] for service descriptions [p.37]

WSDL 1.2 [WSDL 1.2 Part 1] [p.90] is an XML document format for describing Web services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented (RPC) messages.

WSDL describes Web services starting with the messages that are exchanged between the service provider and requester. The messages themselves are described abstractly and then bound to a concrete network protocol and message format.

WSDL is sufficiently extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate. [WSDL 1.2 Part 3] [p.90] describes SOAP Version 1.2, HTTP/1.1, and MIME bindings.

Web service definitions can be mapped to any language, object model, or messaging system. Simple extensions to existing Internet infrastructure can implement web services for interaction via browsers or directly within an application. The application could be implemented using COM, JMS, CORBA, COBOL, or any number of proprietary integration solutions.

Both the sender and receiver of a Web services message must have access to the same service description. The sender needs the service description to know how to format the message correctly and the receiver needs the service description to understand how to receive the message correctly.

As long as both the sender and receiver have the same service description, (e.g. WSDL file), the implementations behind the Web services can be anything. Web services typically are implemented using programming languages designed for interaction with the web, such as Java Servlets or Application Server Pages (ASPs) that call a back-end program or object. These Web service implementations are also typically represented using a Web services description language.

**Issue (wsdwg_service):**

A WSDWG:Service is a collection of *equivalent* endpoints *bound* to the same interface.

**Resolution:**

None recorded.

# A Acknowledgments (Non-Normative)

This document has been produced by the Web Services Architecture Working Group

The editors would like to thank Heather Kreger of IBM for her substantial contributions to this document.

Members of the Working Group are (at the time of writing, and by alphabetical order): Geoff Arnold (Sun Microsystems, Inc.), Daniel Austin (W. W. Grainger, Inc.), Mukund Balasubramanian (Infravio, Inc.), Mike Ballantyne (EDS), Abbie Barbir (Nortel Networks), David Booth (W3C), Mike Brumbelow (Apple), Doug Bunting (Sun Microsystems, Inc.), Greg Carpenter (Nokia), Tom Carroll (W. W. Grainger, Inc.), Jun Chen (MartSoft Corp.), Alex Cheng (Ipedo), Michael Champion (Software AG), Martin Chapman (Oracle Corporation), Ugo Corda (SeeBeyond Technology Corporation), Roger Cutler (ChevronTexaco), Jonathan Dale (Fujitsu), Suresh Damodaran (Sterling Commerce(SBC)), Glen Daniels (Macromedia), James Davenport (MITRE Corporation), Paul Denning (MITRE Corporation), Zulah Eckert (Hewlett-Packard Company), Gerald Edgar (The Boeing Company), Chris Ferris (IBM), Shishir Garg (France Telecom), Hugo Haas (W3C), Hao He (The Thomson Corporation), Dave Hollander (Contivo), Yin-Leng Husband (Hewlett-Packard Company), Mario Jeckle (DaimlerChrysler Research and Technology), Mark Jones (AT&T), Tom Jordahl (Macromedia), Heather Kreger (IBM), Sandeep Kumar (Cisco Systems Inc), Steve Lind (AT&T), Mark Little (Arjuna), Hal Lockhart (OASIS), Michael Mahan (Nokia), Francis McCabe (Fujitsu), Michael Mealling (VeriSign, Inc.), Jeff Mischkinsky (Oracle Corporation), Himagiri Mukkamala (Sybase, Inc.), Don Mullen (TIBCO Software, Inc.), Eric Newcomer (IONA), Mark Nottingham (BEA Systems), David Orchard (BEA Systems), Srinivas Pandrangi (Ipedo), Leo Parker (Computer Associates), Mark Potts (Talking Blocks, Inc), Waqar Sadiq (EDS), Igor Sedukhin (Computer Associates), Hans-Peter Steiert (DaimlerChrysler Research and Technology), Katia Sycara (Carnegie Mellon University), Bryan Thompson (Hicks & Associates, Inc.), Steve Vinoski (IONA), Jim Webber (Arjuna), Prasad Yendluri (webMethods, Inc.), Jin Yu (MartSoft Corp.), Sinisa Zimek (SAP).

Previous members of the Working Group were: Assaf Arkin (Intalio, Inc.), Mark Baker (Idokorro Mobile, Inc. / Planetfred, Inc.), Tom Bradford (XQRL, Inc.), Allen Brown (Microsoft Corporation), Dipto Chakravarty (Artesia Technologies), Alan Davies (SeeBeyond Technology Corporation), Ayse Dilber (AT&T), Colleen Evans (Sonic Software), Daniela Florescu (XQRL Inc.), Sharad Garg (Intel), Joseph Hui (Exodus/Digital Island), Marcel Jemio (DISA), Timothy Jones (CrossWeave, Inc.), Jim Knutson (IBM), Mark Hapner (Sun Microsystems, Inc.), Michael Hui (Computer Associates), Nigel Hutchison (Software AG), Bob Lojek (Intalio, Inc.), Anne Thomas Manes (Systinet), Jens Meinkoehn (T-Nova Deutsche Telekom Innovationsgesellschaft), Nilo Mitra (Ericsson), Joel Munter (Intel), Henrik Frystyk Nielsen (Microsoft Corporation), Duane Nickull (XML Global Technologies), David Noor (Rogue Wave Software), Kevin Perkins (Compaq), Fabio Riccardi (XQRL, Inc.), Don Robertson (Documentum), Darran Rolls (Waveset Technologies, Inc.), Krishna Sankar (Cisco Systems Inc), Jim Shur (Rogue Wave Software), Patrick Thompson (Rogue Wave Software), Scott Vorthmann (TIBCO Software, Inc.).

# B References (Non-Normative)


ebXML MSS
: *ebXML Message Service Specification Version 2.0*, OASIS Technical Specification, R. Berwanger et al., 1 April 2002 (See http://www.ebxml.org/specs/ebMS2.pdf.)

Fielding
: *Architectural Styles and the Design of Network-based Software Architectures*, PhD. Dissertation, R. Fielding, 2000 (See http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.)

SOAP 1.2 Part 1
: *SOAP Version 1.2 Part 1: Messaging Framework*, W3C Recommendation, M. Gudgin, M. Hadley, N. Mendelsohn, J-J. Moreau, H. Nielsen, 24 June 2003 (See http://www.w3.org/TR/2003/REC-soap12-part1-20030624/.)

SOAP 1.2 Part 2
: *SOAP Version 1.2 Part 2: Adjuncts*, W3C Recommendation, M. Gudgin, M. Hadley, N. Mendelsohn, J-J. Moreau, H. Nielsen, 24 June 2003 (See http://www.w3.org/TR/2003/REC-soap12-part2-20030624/.)

RFC 2119
: *Key words for use in RFCs to Indicate Requirement Levels*, IETF RFC 2119, S. Bradner, March 1997 (See http://ietf.org/rfc/rfc2119.txt.)

RFC 2396
: *Uniform Resource Identifiers (URI): Generic Syntax*, IETF RFC 2396, T. Berners-Lee, R. Fielding, L. Masinter, August 1998 (See http://ietf.org/rfc/rfc2396.txt.)

Web Arch
: Architecture of the World Wide Web, W3C Working Draft, Ian Jacobs, 27 June 2003 (See http://www.w3.org/TR/2003/WD-webarch-20030627/.)

WS Glossary
: *Web Services Glossary*, W3C Working Draft, H. Haas, A.Brown, 8 August 2003 (See http://www.w3.org/TR/2003/WD-ws-gloss-20030808/.)

WSA Reqs
: *Web Services Architecture Requirements*, W3C Working Draft, D. Austin, A. Barbir, C. Ferris, S. Garg, 14 November 2002 (See http://www.w3.org/TR/2002/WD-wsa-reqs-20021114.)

WSDL 1.2 Part 1
*Web Services Description Language (WSDL) Version 1.2 Part 1: Core Language*, W3C Working Draft, R. Chinnici, M. Gudgin, J-J. Moreau, S. Weerawarana, 11 June 2003 (See http://www.w3.org/TR/2003/WD-wsdl12-20030611/.)

WSDL 1.2 Part 3
*Web Services Description Language (WSDL) Version 1.2 Part 3: Bindings*, W3C Working Draft, J-J. Moreau, J. Schlimmer, 11 June 2003 (See http://www.w3.org/TR/2003/WD-wsdl12-bindings-20030611/.)

XML 1.0
*Extensible Markup Language (XML) 1.0 (Second Edition)*, W3C Recommendation, T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler. 6 October 2000 (See http://www.w3.org/TR/2000/REC-xml-20001006.)

XML Infoset
*XML Information Set*, W3C Recommendation, eds. J. Cowan, R. Tobin, 24 October 2001 (See http://www.w3.org/TR/2001/REC-xml-infoset-20011024/.)

XML Schema Part 1
*XML Schema Part 1: Structures*, W3C Recommendation, H. Thompson, D. Beech, M. Maloney, N. Mendelsohn, 2 May 2001 (See http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/.)

XML Schema Part 2
*XML Schema Part 2: Datatypes*, W3C Recommendation, P. Biron, A. Malhotra, 2 May 2001 (See http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/.)

# C Web Services Architecture Change Log (Non-Normative)

| | | |
|---|---|---|
| 2003-08-06 | HH | Integrated new text for Web service, choreography, MEP; renamed "legal entity" |
| 2003-07-18 | HH | Integrated new text for message sender, message recipient, intermediary |
| 20030715 | fgm | Added section on security in Stakeholder's viewpoints |
| 20030713 | fgm | (Refactored document) Refactored concepts section |
| 20030419 | dbooth | (Refactored document) Wrote/reworked Introduction |
| 20021202 | DBO | Incorporated f2f changes from the first morning - use of agents, duplicate diagrams, document overview - and the correlation/reliability feature summary |
| 20021114 | CBF | incorporate MTF overview proposal. merged with daveo's changes based on 11/13/2002 f2f session. |
| 20021029 | CBF | tweaked intro per Hugo's suggestion/comment. |
| 20021028 | CBF | incorporated Jean-Jacques' and Hugo's comments on description. amended status and abstract. some restructuring of the references and appendicies. |
| 20021025 | CBF | incorporated Heather's description prose. Incorporated Hugo's comments |
| 20021024 | CBF | incorporated feedback from Joel Munter and some from Jean-Jacques Moreau. |
| 20021022 | CBF | incorporated revised graphics |
| 20021020 | CBF | incorporated Eric's section 1.2 updates. Converted to xmlspec-v2.2. |
| 20020720 | CBF | incorporated SOAP harvest, weaved into the revised flow. added normative biblio. |
| 20020604 | DBO | Initial Rev |