OMA Web Services Enabler (OWSER): Overview

Approved Version 1.1 – 28 Mar 2006

Open Mobile Alliance

OMA-AD-OWSER_Overview-V1_1-20060328-A

Contents

Figures

1. Scope (Informative)

This document, entitled "OMA Web Services Enabler (OWSER): Overview," is part of a group of documents [OWSERSpec] and [OWSERBP] that together describe and specify components of the OMA Web Services Enabler (OWSER). This OWSER Overview is informative and provides an overview of the OWSER and the Web Service technologies that will be used to publish, discover and use components that support Mobile Web Services in a secure manner. It is intended to complement the normative text found in the companion OWSER Specifications.

The OMA Web Services Enabler defines the means by which OMA applications can be exposed, discovered and consumed using Web Services technologies. The purpose of the OWSER is to provide designers of Web Services within OMA with solutions to common functionality using Web Services technologies. Without such a framework, designers of Web Services within OMA would almost certainly try to solve these problems on their own, each in their own way. Such an effort would inevitably lead to brittle implementations, interoperability problems, and increased time to market. The goals of the OWSER are consistent with those expressed in the OMA Service Enabler Strategy White Paper [OMASES].

The document entitled "OMA Web Services Enabler (OWSER): Core Specifications" [OWSERSpec] provides the specifications of the components needed to provide the capabilities of the OWSER as identified in this document, in particular the normative use of Web Service technologies to implement such capabilities.

The document entitled "OMA Web Services Enabler (OSWER) Best Practices: WSDL Style Guide" [OWSERBP] provides informative guidelines on the use of the Web Services Description Language [WSDL] that may be used by OMA-defined Web Services.

While the normative definition of technologies specified in the OWSER is rightly provided in the specification documents themselves ([OWSERSpec] and the Overview provides some information on technologies which are considered to be within the scope or outside the scope of the current OWSER release. In particular, Sections 7, 7.1, and 7.2.2 contain specific scope information.

2. References

2.1 Normative References

[IOPPROC] "OMA Interoperability Policy and Process", Version 1.1, Open Mobile Alliance™, OMA-IOP-Process-V1_1, URL:http://www.openmobilealliance.org/

[RFC2119] "Key words for use in RFCs to Indicate Requirement Levels", S. Bradner, March 1997, URL:http://www.ietf.org/rfc/rfc2119.txt

[RFC2234] "Augmented BNF for Syntax Specifications: ABNF". D. Crocker, Ed., P. Overell. November 1997, URL:http://www.ietf.org/rfc/rfc2234.txt

2.2 Informative References

[MWSREQ] "Mobile Web Services Requirements, Version 1.1", Open Mobile Alliance™, URL:http://www.openmobilealliance.org/

[OMADICT] "Dictionary for OMA Specifications Version 1.0", Open Mobile Alliance™, URL:http://www.openmobilealliance.org

[OMASES] "OMA Service Enabler Strategy White Paper, Version 1.1", Open Mobile Alliance™,, URL:http://www.openmobilealliance.org/

[OWSERBP] "OMA Web Services Enabler (OWSER) Best Practices: WSDL Style Guide", Version 1.1, Open Mobile Alliance™, OMA-OWSER-Best_Practice-WSDL_Style_Guide-V1_0, URL:http://www.openmobilealliance.org/

[OWSERSpec] "OMA Web Services Enabler (OWSER): Core Specifications, Version 1.0", Open Mobile Alliance, URL:http://www.openmobilealliance.org/

[RFC 2828] "Internet Security Glossary", RFC 2828. URL:http://www.ietf.org/rfc/rfc2828.txt

[Saltzer] "End-To-End Arguments In System Design", Jerome H. Saltzer, David P. Reed, David D. Clark, 1984 ACM Transactions on Computer Systems, URL:http://web.mit.edu/Saltzer/www/publications/endtoend/endtoend.pdf

[SOAP] "Simple Object Access Protocol (SOAP) 1.1", Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Nielsen, Satish Thatte, Dave Winer, W3C Note, May 8, 2000, URL:http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

[WebArch] "Web Services Architecture Requirements", W3C Note, Feb 11, 2004, URL: http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/

[WSDL] Web Services Description Language 1.1, W3C Note, 15 March 2001, URL: http://www.w3.org/TR/wsdl

[WSGloss] "Web Services Glossary", W3C Note, 11 February 2004, URL: http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/

[X.800] CCITT Recommendation X.800 (1991), Security architecture for Open Systems Interconnection for CCITT applications, URL:http://online.vsi.ru/library/ITU-T/original/recs/x/x.0800e.zip

3. Terminology and Conventions

3.1 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 [RFC2119].

This is an informative document, which is not intended to provide testable requirements to implementations.

Clarifications intended to augment some text are shown as follows:

> **NOTE:** This is an example of a note.

Such notes are always informative.

Text with nouns in capitals (e.g., Web Service, Policy etc.) in this document is used as defined in section 3.2.

3.2 Definitions

Access control	Protection of system resources against unauthorized access; a process by which use of system resources is regulated according to a security policy and is permitted by only authorized entities (users, programs, processes, or other systems) according to that policy. [RFC 2828]
Authentication	The process of verifying an identity claimed by or for a system entity. [RFC 2828]
Authorization, Authorize	(1.) An "authorization" is a right or a permission that is granted to a system entity to access a system resource. (2.) An "authorization process" is a procedure for granting such rights. (3.) To "authorize" means to grant such a right or permission. [RFC 2828]
Business relationship	This is a formal agreement between two entities (such as between two Providers) to provide services to one another, and may include an establishment of trust between the entities as well as a description of liability aspects between them
Data confidentiality	The property that information is not made available or disclosed to unauthorized individuals, entities, or processes [i.e., to any unauthorized system entity]. [Documents] SHOULD NOT use this term as a synonym for "privacy", which is a different concept. [RFC 2828]
Data integrity	The property that data has not been changed, destroyed, or lost in an unauthorized or accidental manner. [RFC 2828]
Delegation	The ability for multiple entities to perform a set of capabilities represented as a single entity, by assigning subsets of the total capabilities to different entities. In the context of the OMA architecture, delegation refers to the assignment of common, cross-Enabler capabilities to independent entities other than the Service Enabler.
Denial of Service	The prevention of authorized access to a system resource or the delaying of system operations and functions [RFC 2828]
Discovery	Published information, including Web Service Provider information and service descriptions, retrieved from a registry.
Discovery Service	A service that allows requesters to discover resources.
Enabler	A technology intended for use in the development, deployment or operation of a Service; defined in a specification, or group of specifications, published as a package by OMA. [OMADICT]
Firewall	Any of a number of security schemes that prevent unauthorized users from gaining access to a computer network or that monitor transfers of information to and from the network.
Gateway	A network node that terminates a message on an inbound interface with the intent of presenting it through an outbound interface as a new message. Unlike a proxy, a gateway receives messages as if it were the final receiver for the message. Due to possible mismatches between the inbound and outbound interfaces, a message may be modified and may have some or all of its meaning lost during the conversion process. For example, an HTTP PUT has no equivalent in SMTP.

Intermediary	An intermediary is capable of acting in both the receiver and sender role relative to a specific messaging protocol. It may modify message contents, typically in the message header package model. It may apply routing logic when sending the message to the next receiver, which may or may not be the final message target. Note that a gateway is never an intermediary, since gateways terminate messages and intermediaries relay them instead. Being a gateway is typically a permanent role, whilst being an intermediary is message specific.
Key Management	The process of handling and controlling cryptographic keys and related material (such as initialisation values) during their lifecycle in a cryptographic system, including ordering, generating, distributing, storing, loading, escrowing, archiving, auditing, and destroying the material. [RFC 2828]
Message	Content conveyed from one party to another using a protocol. The meaning of the message must be determined in the context of the protocol layer discussed.
Non-Repudiation	The prevention of denying by one of the entities involved in a communication having participated in all or part of the communication. (Source: [X.800])
OMA Web Services Enabler (OWSER)	The OMA Web Services Enabler is a set of common protocols, schemas, and processing rules using Web Service technologies that are the elements that can be used to create or interact with a number of different services.
Policy	A policy is a set of statements that express preferences, requirements, capabilities or other properties
Principal	An entity that has an identity, that is capable of providing consent and other data, and to which authenticated actions are done on its behalf. Examples of principals include an individual end user, a group of end users, a corporation, service enablers / applications, system entities and other legal entities. [OMADict]
Privacy	The right of an entity (normally a person), acting in its own behalf, to determine the degree to which it will interact with its environment, including the degree to which the entity is willing to share information about itself with others. ISDs SHOULD NOT use this term as a synonym for "data confidentiality" or "data confidentiality service", which are different concepts. Privacy is a reason for security rather than a kind of security. [RFC 2828]
Proxy	A computer process that relays a protocol between client and server computer systems, by appearing to the client to be the server and appearing to the server to be the client. [RFC 2828]
Publishing	Web Services offered by a Web Service Provider are made available through a publishing process to a registry where the Web Service Provider has registered. The service description is then made available by the registry. Services may also be unpublished from a registry.
Registration	A Web Service Provider provides information on its business through a registration process with a Web Services Registry. Registries support creation, modification and removal of registrations.
Security Policy	A set of rules and practices that specify or regulate how a system or organization provides security services to protect sensitive and critical system resources. [RFC 2828]
Service	A selection from the portfolio of offerings made available by a service provider, which the user may subscribe to and be optionally charged for. A service may utilize one or more Service Enablers [OMADICT]
Service Description	The information published for a service, containing a text description of the service and additional information that may be used to determine interfaces supported by the service and may also reference additional information that describes requirements or constraints for use of the service.
Service Enabler	See Enabler. [OMADICT]
Service Oriented Architecture	An architectural style that describes message-based interfaces to services which promotes an asynchronous, conversational and loosely-coupled interaction style.
Service provider	An Entity that provides services and/or goods to Principals
Single Sign-On (SSO)	The ability to use proof of an existing authentication with Provider A to create a new authentication session with Provider B.
SOAP	Simple Object Access Protocol, (SOAP) is an XML-based protocol which is used to exchange data between computers on a network.
tModel	A tModel is a data structure representing a service type in the UDDI registry.
Trust	The extent to which someone who relies on a system can have confidence that the system meets its specifications, i.e., that the system does what it claims to do and does not perform unwanted functions. [RFC2828]

Web Service	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. [WSGloss]
Web Service Provider	A Service Provider that exposes a capability as a Web Service
Web Service Requester	A software system that makes use of a Web Service
WSDL	Web Services Description Language, or WSDL, is a platform-independent XML grammar which is used to define the specific method used to invoke the service. It communicates the details of the publicly available functions, the data types and messages that are transmitted, the network location and the transport mechanism used on the network.

3.3 Abbreviations

API	Application Programming Interface
CORBA	Common Object Resource Broker Architecture
CSP	Communication Service Provider
DCE	Distributed Computing Environment
DCOM	Distributed Component Object Model
HTTP	HyperText Transfer Protocol
HTTPS	HTTP Security protocol
ISD	Internet Standards Documents
MWS	Mobile Web Services, a working group of OMA
OAM&P	Operations, Administration, Management, and Provisioning
OMA	Open Mobile Alliance
OMG	Object Management Group
OSF	Open Software Foundation
OWSER	OMA Web Services EnableR
PDF	Portable Document Format
RMI	Remote Method Invocation
SMTP	Simple Mail Transfer Protocol
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SP	Service Provider
SSL	Secure Sockets Layer
SSO	Single Sign-On

TLS	Transport Layer Security
UDDI	Universal Description, Discovery, and Integration
URI	Universal Resource Identifier
VASP	Value-Added Service Provider
W3C	World Wide Web Consortium
WS	Web Services or Web Service
WSDL	Web Services Description Language
WSP	Web Service Provider
WSR	Web Service Requester
XML	Extensible Markup Language

4. Goals

The primary goals of the OMA Mobile Web Services (MWS) effort are to:

- Provide specifications and guidelines for Web Services (WS) technologies, implementations and deployments to integrate and interoperate within the OMA architecture.
- Ensure interoperability across servers and terminals supporting web services protocols through the use of standardized protocols. Profiled Web Services protocols and technologies are outlined in the OWSER specification documents, [OWSERSpec]. As a result of such protocol standardization, we can expect diverse enablers and Internet services to interoperate "on the wire."

Implicit in these goals is the overarching focus of MWS, to define the technology to support a business and technology model for Web Services, using open standards.

In addition to supporting these goals, this Overview document is intended to demonstrate how the OWSER model addresses known requirements that require a systems approach. Many of these requirements are presented in the document "Mobile Web Services Requirements" [MWSREQ]. Note that this current release of the OWSER meets only a subset of the full range of requirements presented in [MWSREQ].

Given these goals, the purpose of this OWSER Overview is to provide understanding of the benefits of Web Services in the broader OMA context, including:

- Create a common understanding of definitions and concepts necessary for understanding the specification in a system context, including security threats and services, privacy, messaging and transactions, policy, and system management.
- The ability to deploy OMA Web Services such that the processing associated with common, cross-Enabler capabilities may be factored out of individual Service Enablers and delegated to other entities in the system (removal of technology silos).
- The ability to automate the discovery and use of policy information that governs Web Services interactions.

Beyond the overview presented in this document, the OWSER specification document [OWSERSpec] details technology standards selection and profiling necessary to create interoperable Web Services solutions, while reusing existing standards efforts as much as possible.

5. Introduction

The ability to deliver appealing, low cost data services is an important driver of success in the mobile market. However, this is proving difficult to achieve with current infrastructure and tools. Much effort has been expended to standardize over-the-air interfaces between wireless devices and elements in the mobile network, yet the interfaces between Internet/web applications and mobile network services remains fragmented, characterized by a complex mixture of standard and proprietary interfaces. In today's environment, Internet applications must be tailored to the particular mobile infrastructure in which they are deployed and, in the worst case, to proprietary interfaces to network elements deployed in those environments. This drives up the cost of application development and ultimately the cost of services offered to mobile users.

Many of the technical working groups within OMA address, each in their own way, the need to support standardized interactions. Aside from a transport and message encoding definition, a particular Enabler release may also address security concerns, discovery , and other requirements common to a distributed environment As a result there are now a number of standards for how a client should interact with the components and/or Enablers defined in each of these groups. In the development of these mobile Internet standards there has been little consideration for the dependencies between the separate standards these Enabler "silos" have created, and how they should work together.

For the Service Provider this fragmentation increases the complexity of managing the heterogeneous group of Enablers in the network and the support requirements for enabling integration. Inconsistencies in billing implementations across Enablers make it extremely difficult for the Service Provider to implement the desired business model(s).

Applications are not the only consumers of Enablers. Enablers may also consume other Enablers to provide the functionality required by an Application. The lack of consistency described above also complicates Enabler integration in the Service Provider's network.

Applications that interact with several OMA Service Enablers – and even those that interact with only one Enabler– will use a number of common capabilities. This approach makes them accessible in a uniform way by all applications; and it also enables the Service Provider more flexibility in implementing business models.

Examples of functionality common across Service Enablers may fall into the following categories (*Note: this list is not all-inclusive*):

- **Service Discovery.** The publication, service description, and discovery of Service Enablers.
- **Messaging.** Correlation, guaranteed exchange, transaction support, and routing of messages between the WSR and the WS.
- **Security.** Services such as authentication, authorization, data integrity and confidentiality, privacy, and key management that provide a trusted environment for all parties involved.
- **Charging.** How the appropriate parties are charged for the services they consume or establish the charges for the services they offer.
- **OAM&P.** Management, monitoring, and provisioning of the Service Enablers that exist within the Service Provider's network.
- **Service Level Agreement.** Establishing and ensuring a desired level of service.
- **Service Agreement Management.** Providing the appropriate parties with the information and tools to enforce the contractual commitments of the Service Level Agreement.

6. Service-Oriented Architecture

The architecture underlying the technologies that form the basis for the OMA Web Services Enabler (OWSER) is that of a Service-oriented Architecture (SOA). A SOA is an approach to describing a distributed computing environment where software resources are made available as services over a network. Such attempts at distributed computing are not, of course, new, but there are several distinct features of SOAs that are different from the previous approaches.

One approach to understanding the use and value of SOAs is to contrast them with previous approaches to distributed computing. As with many aspects in this area, it is not possible to offer a definition of a SOA that categorically distinguishes it from any other architecture. Rather, it is best understood by contrasting it to the characteristics of other distributed computing architectures, by highlighting where it differs. (Note, however, that not all these aspects need to be satisfied for a given architecture to be called a SOA.) In the following, we outline some of these distinctions:

1. **Messages versus Application Programming Interfaces (API):** In a SOA, the interaction between a service and its consumer is based on conversational, message-based interactions. That is, the requester and the provider of a service exchange messages, conforming to a particular protocol and data description syntax (more formally, a schema). In contrast to this, traditional distributed computing techniques like CORBA, DCOM, and Java RMI have been based on (language dependent or language-independent) APIs, where the paramount goal is to provide a programmatic model of the interaction between a service requester and a service provider.

2. **Coarse-grained versus fine-grained interfaces:** APIs, by definition, deal with programmatic interfaces and hence programming level constructs, i.e., programming objects, and their methods. A SOA, by contrast, does not deal with programming constructs, but with services that offer a different granularity for their interface. Such services are defined not by any underlying programming objects but *by the facet of the business that the service exposes*, e.g., submitting an invoice to a purchasing system, updating a travel reservation in a calendar application etc. Such interfaces offer descriptions of what messages are suitable input and output to and from the service, and under what conditions they are exchanged. How messages are processed or generated are deliberately out-of-scope of a SOA, and viewed as an implementation matter. Another way of stating this difference is to say that the coarse-grained interfaces of SOAs are more like contracts (a description of what an offering party is willing to do under what conditions) rather than the more traditional programming interface description as in CORBA or DCOM.

3. **Asynchronous versus synchronous interactions:** Such conversational exchanges in a SOA are defined to be asynchronous, in that there is no predictable timing relationship between the exchange of messages, that responses may be delivered long after a request was sent, and responses may be received on different transport connections than the one on which a request was sent. This contrasts with most programmatic interactions, which are synchronous by definition.

4. **Peer-to-peer versus client-server interactions:** SOAs are geared towards application integration where two *independent* applications exchange well-defined data and messages to complete some task at hand. API driven distributed computing are invariably client-server interactions, which in itself is a programming construct, where there exists a pre-defined and a very well defined coupling between the two sides of the *distributed* application.

5. **Interoperability versus application portability:** Entities in a SOA interact through well-defined (or extensible) messages and the aim is to ensure that the two peer applications can support the message semantics and syntax in completing their independent tasks. By contrast, much of the work in traditional distributed computing has been to ensure both client-side and server-side application portability across heterogeneous languages and platforms through standardized language bindings (for the client side) and standardized object adapters (for the server side).

6. **Discovered partners versus pre-defined partners:** In its most general usage, a SOA allows parties to discover useful and compatible services. In practice, however, at least in terms of early adoption of this concept, most interacting parties are typically already known to each other, and some sort of prior business relationship governs their partnership. However, typical programmatic interactions in traditional distributed computing are not based on such a form of *service* discovery, but rather between well-defined entities.

7. **Internet versus intranet usage:** SOAs, which not restricted to interactions between business partners that take place across the public Internet, offer much of their advantages in such an environment. In contrast, the traditional distributed computing technologies have primarily been restricted to applications in an intranet environment owing to difficulties in scalability and firewall traversal when used across the Internet.

These features, which distinguish a SOA from the traditional distributed computing paradigms, constitute what is often referred to as a “loosely coupled” distributed system.

As described in the previous section, the loosely coupled distributed computing problem is tackled using a SOA. To actually implement the entities participating in a SOA, the various aspects of a SOA need to be defined in terms of concrete technologies.

In contrast to the numerous attempts to solve the distributed computing problem – such as, for example, OSF DCE, Microsoft’s DCOM, OMG’s CORBA, and Java RMI – none of which achieved universal acceptance, it appears that the industry has coalesced around a set of technologies to support SOAs. "Web Services" is a recent industry initiative to address the needs of SOAs.

In contrast to the previous efforts cited above, Web Services was defined from the ground up to address loosely coupled, service-oriented architectures. In so doing it provides a solution that makes integration both easy for the application developer and cost-effective for the systems that populate the architecture. Web Services achieves these goals by basing its implementation on Internet-based standards to provide platform and programming language neutrality. Choosing XML as the universal data description language, and utilizing the most common application protocol – HTTP – for data transfer, make the task and cost of integration much more tractable, particularly for smaller players.

Although it is beyond the scope of this document to provide a detailed description of the evolution of Web Services and the organizations which have contributed to that continuing evolution, the following groups (in alphabetical order) are some of those which have influenced or provided important standards on which the OWSER is based:

- OASIS, the Organization for the Advancement of Structured Information Standards, http://www.oasis-open.org
- The Parlay Group, http://www.parlay.org
- The Third Generation Partnership Project (3GPP), http://www.3gpp.org/
- The World-Wide Web Consortium (W3C), http://www.w3.org
- The Web Services Interoperability Organization (WS-I), http://www.ws-i.org/

In addition to providing support for programmatic interactions like remote procedure calls, Web Services allows for a more asynchronous “document exchange” mode of interaction that provides flexibility and extensibility. Web Services seems to be well on the way to achieving broad, if not universal, acceptance (the kind of universal acceptance that TCP/IP, HTTP, HTML, and XML, for example, have achieved) as a way to implement service-oriented architectures.

A Web Services-based architecture contains three functions, as depicted in Figure:

- A Web Service requester (WSR),
- A Web Service , and
- A Web Service Registry

A Web Service provider’s responsibility is to create a service description, publish that service description to one or more means of Discovery, and have the WS ready to receive messages from WSRs. A WSR *find*s, possibly through one of the many available service registries, the service description of interest and uses this service description to *bind* to the Web Service provided by the WSP.

A Web Service registry's responsibility is to "advertise" the Web Service descriptions published to it by WSPs and to help WSRs search through its registry to find a service description of interest. In this context, the service registry is similar to a matchmaker; once a match is found, it is no longer needed, and all subsequent interactions are strictly between the WS and a WSR.

A Web Service registry is, strictly speaking, an optional part of the Web Service architecture, as a service description may be obtained by other, out-of-band means. However, as one of the goals of Web Services is program-to-program communications, one of the advantages of using a Web Service registry is so that service descriptions may be

programmatically "consumed" *at design time* by WSRs to create the service invoking code. Another use of such a registry is at run-time when such a registry can be consulted to determine the current address at which a service is being offered.

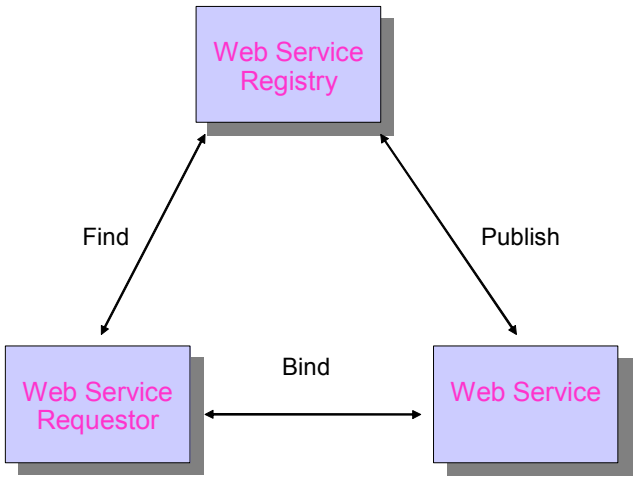


Figure 6.1: Web Service Architecture

Web Service registries may be partitioned in many ways, with access rights that range from allowing access to the service descriptions to any and all WSRs, to those that allow access to WSRs within a single trust domain. By controlling visibility to its service descriptions, a service provider can offer Web Services to selected parties.

7. The OMA Web Services Enabler (OWSER)

Within a service provider's domain, many of a provider's "assets" are deployed as large application "silos". These applications often use unique data formats, proprietary interfaces, and may exist on multiple platform technologies. They are very difficult and expensive to change, let alone integrate with other systems. The service-based architecture model makes it possible to put wrappers around these applications, transforming them into Web Services, which can support flexible business models and can be more easily integrated with other systems within the service provider's domain. Internal application integration will likely be implemented as Web Services interacting over fixed networks, e.g., an intranet or VPN.

In order to mitigate the challenge developers face in achieving robust, distributed, decentralized, and interoperable implementations with rapid time-to-market, the OWSER defines capabilities that are expected to be common across most (if not all) Web Services within OMA. These capabilities are referred to collectively as the OMA Web Services Enabler. Enabler capabilities can be implemented in various ways involving WSRs, WSPs, Web Service intermediaries, or legacy elements. The OWSER is designed to be extensible, so that newly-identified capabilities may be defined and included in the OWSER.

Work is underway elsewhere in OMA to define "common functions". A common function is (loosely) defined as processing that is (or should be) common across all OMA Service Enablers. Typical examples include authentication, authorization and charging. Many existing Enabler specifications define Enabler-specific solutions to such common processing. The existence of multiple, non-interoperable, solutions to common processing problems is a major contributor to the silo problem described above. In order to break down these silos, individual Enablers should "delegate" the processing of common functions to other system entities assigned to those functions. Section 7.1 below describes how delegation may be realized within the OWSER.

The OWSER leverages work within the broader industry where applicable. The OWSER reuses solutions (through normative and informative reference) that have achieved widespread consensus in the industry at large, and which have achieved (or are expected to achieve) standards status in various industry forums.

Web Services in general constitute a relatively new technology domain. As this document is being written, many interesting and vital Web Services capabilities have not yet been standardized. In order to minimize divergence from the industry at large, consideration of such capabilities is deferred from inclusion in the OWSER until such time as industry standards emerge. Examples of deferred capabilities include, but are not limited to:

Messaging

- **Routing.** Explicit routing of a message by the requester through one or more intermediaries. The responder may route a message through any number of downstream intermediaries but how such routing is accomplished is beyond the scope of the specification.
- **Message correlation.** How can one associate a given WS request with its response, particularly when multiple responses are generated?
- **Guaranteed message exchange.** How can one ensure that a message was received, and only once?

Security

- **Availability –** Ensure that a resource may be used at the expected service level.
- **Security Policy** – Define a set of criteria for provisioning security services and its enforcement
- **Non-repudiation –** Ensure that the transaction mechanism and records support resolution of potential disputes, reducing the risk that a party may disavow their actions or commitments.

Policy

- **Policy.** How can an Enabler convey its policies to Web Service Requesters?
- **Policy Management & Access Policy Provisioning**

Service Management

- **Monitoring.** How are elements of the OWSER monitored for availability and health?
- **Auditing.** How is logging and auditing supported?

Service Level Agreement

Coordination

- **Transactions.** How can one reverse the effects of a previous transaction request?
- **Orchestration.** In what sequence, and under what conditions, does a Web Service invoke other Web Services in the course of fulfilling its assigned function?

7.1 How the OMA Web Services Enabler Works

The process leading up to and including an application (acting as a WSR) invoking a Service Enabler (offering some capability as a Web Service) in an OWSER environment can be described by the following three phases:

- **Service Publication** – how the Service Enabler makes its capabilities (including its endpoint address) known to potential consumers
- **Service Discovery** – how an application finds the Service Enabler it wishes to consume
- **Service Invocation** – how an application consumes the desired functionality provided by a Service Enabler

These are also the three basic aspects of the Web Service interaction model shown in Figure in section 6, there identified by the more informal terms – publish, find and bind.

Figure 7.1 illustrates these three phases in the context of an environment where the capabilities provided by the OWSER may be used. Note that Figure 7.1 serves a pedagogical purpose in identifying the features of the Web Services architecture mentioned above superimposed on a generic deployment environment. Obviously, in real deployments, there may be additional roles, or different boundaries based on the business relationships between those who adopt these roles. This section does not deal with such deployment-related issues.

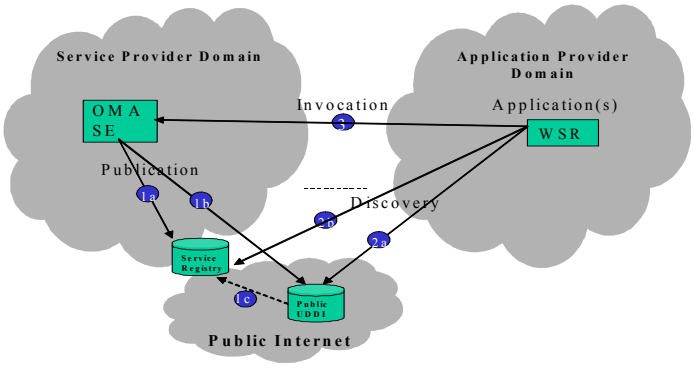


Figure 7.1: How the OWSER Works

Service Publication. Service publication is the necessary first step for making a Service Enabler available for consumption by applications. This step consists of publishing the information required to use the Service Enabler such as location where the Service Enabler may be accessed, the Enabler's message interface description, the protocols by which the messages are exchanged, as well as any additional information, such as the security environment between the WSR and the Service Enabler, that is required for using the Service Enabler.

Some of this information, such as the message interface description, the binding of the messages to concrete protocols and the service access point may be described in a formal notation known as WSDL, but other aspects, such as policies required by the service provider (i.e. what security protocols are in use) can only be specified at this time as textual descriptions. In the case of UDDI, tModels provide a means for referencing such descriptions. The use of a formal notation allows development tools to automate the creation of some of the code necessary for the messaging interface between the WSR and the WS. The long-term goal of the Web Services community is, of course, to be able to define all aspects of the Web Service through formal notations so as to allow greater automation in tying together WSRs and WSPs. As noted before, only a small part of such notational support is available at this time in any standardized form; therefore, this version of the OWSER does not offer any greater support in this area than that offered by WSDL.

The information on locating and making use of the capabilities of a Service Enabler may be published to an internal service registry such as a private service registry (see (1a) in Figure 7.1), and/or to an external service registry such as an public UDDI directory (see (1b) in Figure 7.1). In typical usage, the entry in the public UDDI directory provides some basic information about the service provider and points to more detailed service information (see (1c) in Figure 7.1) – which is available as read-only information to anyone on the public Internet. (This is shown in Figure 7.1 by placing the service provider's service registry at the boundary of the service provider network and the public Internet to show that the data may be partitioned or different access controls may apply to access from within and outside the service provider's domain.)

The means of registering the service information may be automated, for example by using the UDDI publishing API or may be done by other means such as via a webpage, or a PDF document, creating and delivering a developer CD, etc.

Note that only (1a) and (1b) are in scope for OMA MWS. Step (1c) is more a matter of deployment practice and may vary from Service Provider to Service Provider.

Once the Service Enabler description has been published, it is then ready to be discovered and used by an application.

Service Discovery. The discovery options available to an application will vary depending on how the service provider chooses to make available such information.

In many cases, there may already be some business relationships between the application provider domain and the service provider domain. In such a case, the service provider may make available the access point to (some parts of) its service registry to its business partners. The service registry data may then be browsed (as read-only) using the UDDI Discovery API (see (2a) in Figure 7.1) if such support is provided by the service registry, or the service provider may make available Service Enabler information using other means such as developer websites, promotional CDs, email, or other off-line means. The latter are not in scope of the OWSER specifications.

Application providers who do not have any prior business relationships with the service provider may discover the service provider via a public UDDI registry (see (2b) in Figure 7.1). Typically, public UDDI registries describe the sorts of business the service provider is in, rather than detailed technical information, and provide contact information to which viewers may be directed to further explore the potential for forming a business relationship. Those with whom a service provider forms business relationships are directed to more detailed technical service information as described in the preceding paragraph.

The information that an application provider can discover will consist of a description of the Service Enabler's message interface (i.e., the message exchanged, the binding of such messages to underlying protocols, and the access points at which message exchange occur) as well as all additional information – often called policy – that describes how the Service Enabler may or is required to be used. Such policies include, among other things, information on the security environment that needs to exist between the WSR and WS.

As mentioned in the service registration feature, some of this information may be described in industry-standardized formal notation such as WSDL, but there are other aspects, such as those that describe policies, for which there is no standardized formal notation at this time. Note also, that not all aspects of the interaction are manifest on the wire, i.e., as a part of the message exchange or protocol. For example, there may be Service Level Agreements (SLA) in place that are agreed to by the service and application providers that are govern their Web Service interactions such as, for example, the number of

messages per time period that an application provider is able to send, or the maximum time that may be taken to complete a particular message exchange. These represent technical constraints on both the requester and provider of the Service Enabler.

Service Invocation. The technical information relevant to the Service Enabler interface is used to implement the WSR in the application provider domain. This consists not only of implementing the appropriate messaging interface conforming to the service's description, but also setting up the appropriate security infrastructure as described by the service provider. For example, the application may be required to provide authentication credentials to the Service Enabler that conforms to the supported authentication methods defined in the Service Enabler's security policy. Additional security constraints may involve encrypting all or parts of a message.

The service invocation is represented as (3) in Figure 7.1. While this is shown as a single arrow between the WSR and the Service Enabler, actual deployment architectures may make use of any of the deployment patterns described in section **Error! Reference source not found.**, many of which involve the services of intermediate elements – more formally, SOAP intermediaries – in the message path.

In the current Web Services messaging model, a SOAP intermediary is an entity that sits on the path between a WSR and a WS and typically only operates on relevant SOAP message headers targeted at it. Sometimes it may operate on the SOAP message body too, e.g. for encryption/decryption. SOAP provides a distributed processing model, which means that it allows different parts of the message to be evaluated and processed by different SOAP-capable entities (i.e., SOAP intermediaries) that may be encountered in the message path between a sender and an ultimate receiver. SOAP intermediaries can be used for authorization, encryption, load balancing, message routing, etc. Therefore, having SOAP intermediaries handle common tasks allows for flexible service composition and may relieve a Service Enabler from performing these common functions.

SOAP intermediaries also allow for introducing new features in the message exchange between a WSR and a WS in a modular way without hard-wiring the deployment pattern or changing the implementation of the WS. These new or additional features are signalled using headers in the SOAP message. The new features introduced can be handled by one or several SOAP intermediaries on the path between the WSR/WS as well as by the WSR/WS themselves, if that is desired. To identify who needs to deal with which part, the SOAP message provides an attribute, "role", as well as another, "mustUnderstand", which allows various SOAP nodes which have taken on such roles to perform the functions they have been configured for. Furthermore, because of the deployment "invisibility" from the point of view of a WSR, this mechanism also lends itself naturally for factoring out common functionality required by a WS for handling by SOAP intermediaries.

Delegation. Assume that there is an authentication header, which contains a security token by which the WSR authenticates itself to the Service Enabler. As mentioned before, it must be qualified by the "role" attribute. (If it does not, it means that this header is destined for the ultimate receiver of the message - the Service Enabler, in our case.) Therefore, if there is a particular deployment where the handling of the security functions is delegated to a trusted intermediary rather than performed by the Service Enabler, the following steps can be used to effect delegation:

- Define a particular role (e.g., "Authenticator", but more formally a URI) and
- Provision only the SOAP node in the service provider domain that plays the "Authenticator" role to take on that role (and ensure that the Service Enabler does not assume that role)
- In the specification for the authentication header (which is *not* a part of the OWSER specification at this time, but could be at some future point if it were felt that a future OWSER was the best place to define it), specify the rule that the header be removed before the message is passed on downstream.

Thus, the Service Enabler would never see the authentication header, because the intermediary assuming the "Authenticator" role has processed it, and its specification requires that it be removed before the message is sent along. The Service Enabler would act on the SOAP payload (by default intended for the role = "ultimateReceiver") and possibly any other headers with additional roles that it has assumed.

The distributed processing model for service invocation therefore allow different service provider networks to configure systems in their networks that allow Service Enablers to delegate the handling of common functions, such as authentication, or assume those depending on circumstances, choice or ability.

The OWSER does not mandate any of these choices but offers the flexibility to support any of these choices.

Many examples of SOAP message exchanges described in current literature portray SOAP as a serialization format for carrying remote procedure calls (RPC) and their returns. Also, many examples of deployed SOAP applications use a simple

request-response message exchange pattern borrowed from the underlying HTTP transfer, further reinforcing a specific usage pattern as being a more natural usage pattern for SOAP based interactions.

In fact, SOAP is a one-way message exchange format with a distributed processing model based on the concept of a message path from a initial sender to an ultimate receiver via zero or more intermediaries. An application can compose such one-way message exchanges into more complex interaction patterns, such as request-response, and multiple to-and-fro "conversational" exchanges. Thus, concepts such as "synchronous" and "asynchronous" are not applicable to SOAP.

It is often the case that a particular Web Service is commonly used as a precursor to using another, such as an Authentication service before a Purchase service. Such common and frequently used Web Services (such as the above-mentioned hypothetical Authentication service) are sometimes referred to as "framework" services.

Whilst such a taxonomy can be useful in certain circumstances, in general, a Web Service is an independent, stand-alone entity. If it can be discovered, and the requester is given access, it may be consumed. The order in which web services are used, or chained together, is independent of the web service and is part of a higher-level "choreography" description. Therefore, where a particular Web Service (i.e., a Purchase service) is found to be general enough so that it could meet the needs of many applications, it may find itself being reused frequently in a chain of web service interactions. Whether or not such a service is a part of a framework becomes a relatively uninteresting question.

7.2 OMA Web Services Enabler Architecture Models

There are two primary modes of interaction between a Web Service Requester (WSR) and a Web Service (WS) in the OWSER.

- **Direct** – addresses the cases where both the WSR and the WS support the full Web Services stack, i.e., a complete implementation of the Web Services messaging and discovery functionality (without any OMA-specific subsetting). Normative definition of the required functionality is included in the OWSER specification documents - [OWSERSpec].
- **Indirect** – addresses the case where the WSR does not support the full Web Services stack (e.g., a legacy device), yet wishes to consume the services provided by a WS.

In some cases an intermediary may exist in a particular deployment case. Intermediaries (more precisely SOAP intermediaries) are elements located in the path between a WSR and a WS. The interactions between a WSR and a WS and its relationship with intermediaries can be classified in three categories:

- **WS interactions without any intermediaries** - This means that there is no intermediate processing (aside from transport routing) of the messages exchanged between a WSR and a WS.
- **WS interactions with intermediaries not visible to the WSR** – In general, a SOAP intermediary may always be interposed in the message path between a WSR and a WS. However, traversal of the message through intermediaries does not necessarily require source-routing capabilities in SOAP.
- **WS interactions with intermediaries visible to the WSR and WS** –There are instances where it may be necessary to route a message between a WSR and a WS through an (a series of) intermediary (intermediaries). Such intermediaries may provide value-added services beyond those that are of direct concern to the WSR and WS. This requires that the WSR and/or WS be able to indicate the full or partial route in the message through such intermediaries (source-routing). Note that currently no standard solutions for this type of source level-routing exist.

7.2.1 Direct Architecture Model

The main elements in the Direct architecture model (see Figure 7.2 below) are two types of roles: Web Service requesters (WSR) and Web Service providers (WS). A WSR communicates with a WS to obtain a service.

The direct model does not assume anything about the implementation assigned to the WSR and WS roles. Both types of roles can be realized in terminals, servers, or any kind of computing device so long as the appropriate capabilities can be

supported. The direct model assumes that both the consumer and the provider support standard Web Services stacks therefore it excludes those terminal devices which cannot support these capabilities.

The direct model allows cascading of WSRs/WSs, where one WS is at the same time the WSR of another Web Service. However, each logical edge between a WSR and a WS constitutes a separate Web Services interface.

7.2.2 Indirect Architecture Model

The indirect model (see Figure 7.2 below) serves a requester device (in this case, depicted as a mobile device) that doesn't have a complete standard Web Services stack, but wishes to use the services provided by a WS. This type of requester device communicates with a WS via an intermediate element, usually referred to as a proxy. A proxy in turn, acting in the role of a WSR, will communicate with the WS using a standard WS interface. This indirect relationship requires protocol conversion functions in the proxy to convert the protocol/interface used between the requester device and the proxy to the Web Services-based protocol/interface used between the proxy and the WS. Security considerations may require that a proxy be in the same trust domain as the requester device which lacks Web Services capabilities.

In the particular example below, a legacy mobile device (i.e.- one that does not implement a full Web Services stack) assumes the role of the requester that requires the use of an intermediate element to consume the services of the WS. This case is not restricted to mobile devices, nor will all mobile devices require the Indirect model.

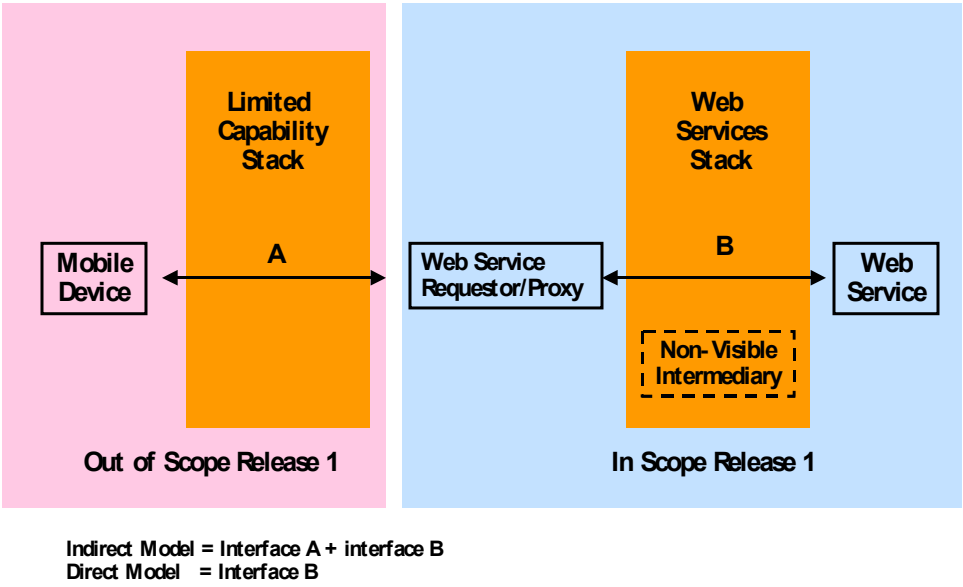


Figure 7.2: Direct and Indirect OWSER models

Note that the proxy-based functionality depicted in the left side of Figure 7.2 is outside the scope of the current OWSER release. Also note that the presence of an intermediary is included within the Direct model, and is within the scope of the current OWSER release.

7.3 Technologies of the OMA Web Services Enabler

This section provides an implementation-neutral overview of the various technologies of the OMA MWS Enabler Release.

7.3.1 Web Service Registry

As noted in the discussion of service publication in section 7.1, there are a variety of requirements or policies that may imposed on the use of a particular web service by the service provider. At the current time, there exists no standard formal notation for expressing policies beyond that provided by WSDL (message interface description, the binding of the messages to concrete protocols, service endpoint). Textual description of such policies may be published to the Web Services registry.

Such information may include, for example, security policies indicating the type of security tokens that a Web Service Requester must present in a request to a Web Service, or policies governing the processing of individual elements of a message as in Delegation.

This section provides an overview and example of the use of UDDI as a Web Services registry.

7.3.1.1 Use of UDDI as the Web Service Registry

This information applies to both public and private UDDI registries.

The Web Service Provider selects a Registry, connects to a UDDI node hosting that Registry and, assuming authentication and access control verification, creates an account. The account provides the Web Service Provider with access to create entries representing the business, its services, and the information required to access its services.

7.3.1.2 White Pages - Publishing the Web Service Provider Information

The information on the Service Provider is made available through the 'white pages'. This information includes traditional information, and may be used to identify businesses that a company may wish to do business with. For example, a business may pre-qualify businesses it is willing to accept services from by validating their information through their white pages information.

7.3.1.3 Yellow Pages - Publishing the Service Definitions

Services are offered through the 'yellow pages' entries. Queries for services may result in multiple entries, which may be further qualified through the examination of criteria published with the service definition.

7.3.1.4 Green Pages - Publishing the Access Information

Information required for initiating a bind operation is provided through the 'green pages' entries associated with the service. Additional information may be published here, including information regarding any requirements on the Web Service Requester to access the service.

7.3.2 An Example End-to-End Scenario

A Service Provider, "Any Telco", wants to offer access to a Location service that is deployed as a Web Service and supports a standardized Location WSDL interface.

7.3.2.1 Publishing the Company Information

The first step is for "Any Telco" to select a Registrar and to connect to the UDDI node hosting the Web Service Registry. They may choose any public UDDI registrar as their Registrar. An administrator from "Any Telco" registers the company with the selected node, and logs on to the registry. The administrator then creates the white pages entry for the company including the following information:

- Business Name (Any Telco Sample Company)
- Description (English, Sample Telco Company)
- Contact (John Smith, Customer Service, john.smith@anytelco.com)

The next step is to define the taxonomy into which the business belongs to enable searching by business type. The following information is included in the yellow pages entry:

- Locator (UNSPSC 83.11.22.00.00 Enhanced telecom services)

Lastly, the set of services to be offered by "Any Telco" is defined by creating a green pages entry for the Location service. This entry will reference the tModels that correspond to this service offering.

The following information is included in the green pages entry:

- Name (Location)

- Description (OMA Location Service version 2.1)
- Access Point (http://www.anytelco.com/location)
- Access Point Description (Public access point for AnyTelco Location service)
- Service Locator (UNSPSC 83.11.22.00.00 Enhanced telecom services)

After receiving the information from the green pages entry, the Web Service Requester may proceed with the binding process to gain access to the calling features.

7.4 Security Model

Web Services may be protected using security mechanisms designed to reduce the risks associated with security threats. Which threats are important depend on application risk and security policy considerations. Different mechanisms may be used to address similar concerns, having different detailed properties.

One security mechanism category is transport layer security i.e., security mechanisms applied at the protocol layers used to convey SOAP messages. An example is HTTPS (HTTP with SSL/TLS). Another category is SOAP Message Security i.e., the use of security technologies in conjunction with SOAP messaging. SOAP message security allows portions of the SOAP message to be protected, making it suitable for use with SOAP intermediaries while transport security protects an entire SOAP message. A third category is application security, i.e., use of security mechanisms with application data without consideration for how that data is transferred, for example without regard for SOAP messaging. Transport and SOAP message security are in OWSER scope, application security is not. Specific transport and SOAP message security mechanisms are profiled in the security section of the OWSER specification.

The following security sections summarize the high level considerations that influence the use of the security mechanisms outlined in the specification. This includes the threats that must be considered to create a security policy. The security policy defines which risks are likely and important, which security requirements must be met, and what system considerations must be addressed. Such a policy should address system wide requirements for privacy and availability as well as deployment architecture considerations, such as the use of intermediaries and proxies. These issues are outlined in the following sections.

7.4.1 Threat Model

Security technologies are used to manage the risk and vulnerability associated with attacks taken on the systems, the information and data, and the services. The costs associated with the risks and costs of handling the vulnerabilities justify the cost of the security mechanisms. Security mechanisms are deployed to countermeasure the vulnerability by reducing the threat and the risks of known attacks. The following list describes common security threats:

- Inappropriate **content modification** is a risk, either due to a malicious attack or due to an inadvertent mistake. Although a checksum can detect a change, it cannot detect tampering since the checksum may also be modified. Technologies such as digital signatures or Message Authentication Codes (MAC) (such as a keyed hash) may be used to detect changes and support source authentication. Such technologies may be deployed to protect information in transit (SSL/TLS), end-to-end at the application-messaging level (for instance, using WS-Security) or end-to-end at the application content level (for instance, using XML Digital Signature).
- **Denial of service** is an attack to either disable or degrade the ability of a server to provide services to clients. Overwhelming the server with requests that require excessive processing or that consume excessive resources, are two examples. Denial of service is the condition when a service falls below the required committed level, including unavailability of the services. Such denial of service may be cause by an intentional attack or by accidental conditions. Availability is a condition in which there is no denial of service or degraded communication quality
- **Eavesdropping** is where information is viewed that should not be, either by examining messages in-transit or by examining content stored at a server. Using confidentiality features such as encryption of data or messages may prevent this. Encrypting data in transit, such as by using SSL/TLS, does not protect it when stored at a server or routed through application level intermediaries. SOAP message confidentiality protects the content when routed through application level intermediaries but not when stored at the destination application server. Application level encryption can protect information even when stored at an end server or when processed by additional applications.
- A **man-in-the-middle** attack may be used to add, remove and change messages between two parties. Requiring authentication of both end parties may be used to avoid this problem.

- A **masquerade attack** hides the actual entity and impersonates to be a different entity that may have the authorization and privileges to access resources. This attack is usually used with reply and content modification. For example, authentication information can be captured and replayed after a valid authentication sequence has taken place.
- A **replay attack** is when someone captures and resends a message to obtain an anticipated result. Including some freshness material with messages, such as a timestamp or a unique non-repeating value, and checking this material before acting on a message at an endpoint can prevent this.
- **Trojan Horse** attacks have introduced quite significant impact in recent years. When introduced to the system, a Trojan horse performs an unauthorized function within its authorized function. One of the examples is the virus and worm attack.

NOTE: The above is not an exhaustive list of possible threats. The ones listed here are those that are considered most relevant to the initial version of the OWSER. Other threats and their countermeasures will be added as they are identified.

7.4.2 Security Policy

In order to address threats of accessibility attacks, impersonation, eavesdropping, content modification, and inappropriate disclosure of private information, certain security requirements, must be met, forming the basis of a security policy. These include the following, consistent with the requirements listed in the W3C Web Services Architecture requirements [WebArch]:

- It must be possible to transfer information maintaining integrity and confidentiality
- It must be possible to authenticate all parties in every transaction
- Support must be provided for both persistent and session authentication of content
- Technical support for non-repudiation should not be precluded, including audit support.

A security policy is a set of statements that describe what is and is not permitted with regards to privacy, authentication, integrity, confidentiality, access control, and other security requirements. Policy suggests what is of paramount importance in the control of security, and it sets the topmost level of a security specification. A generic policy might say "information may not be given to, accessed by, or permitted to be inferred by, nor may any resource be used by, those not appropriately authorized." The nature of authorization is what distinguishes various policies. Different entities define their own set of security policy. However, a complete security policy is required in every entity that will necessarily address many concerns, which are outside the scope of the MWS.

Despite the initial scope of the OWSER to server-to-server interactions, care must be taken not to preclude terminal involvement in interactions, considering the potentially constrained capabilities of terminals. Specifically, policy components must not be mandated that would prevent terminal participation. A complete security policy will necessarily address many concerns for a particular implementation. In addition, as policy is an emerging area for web services, it is premature to normatively specify policy components at this time.

Mechanisms are outlined in the OWSER specification that may be used to address security requirements, including authentication, integrity, confidentiality, access control and authorization and support for non-repudiation. Additional system-wide concerns that must be addressed include privacy, trust and key management, and availability.

7.4.3 Systemic security considerations

Some requirements, such as privacy and availability require system wide consideration and cannot be met by simply deploying certain security mechanisms. For this reason they are identified here instead of the security specification.

7.4.3.1 Privacy

Privacy is a broad area that includes the topic of information privacy, the main issue in the OMA Web Services security context. Information privacy is about data protection, and the end users' rights to determine how, when and to what extent their personal identifiable information, is communicated to other parties, and sometimes to control this use. Such information can include personal identifiable information, as well as other personal information such as preferences, age, gender, account numbers, user equipment-related information and so on. Legislation is a major driver of privacy requirements in some

countries. Data protection aspects must be considered before a service is used, during service usage, and after the service is used so that the service provider does not misuse the data collected.

Privacy–enhancing may technologies include mechanisms to ensure anonymity, ensuring that the identity of a party is not disclosed to another party. In practice this means that only a subset of a user's attributes may be accessible to the other party. Pseudonyms are a way to obfuscate the identity of the subject but allow the appropriate information about the user to be communicated. An example of such a pseudonym usage is a web site offering some content to be downloaded from a third party. A pseudonym is given to the third party, so that the party can share information about the download with the web site, but cannot track the downloads of individual parties over time.

The rules that govern the management, release, and use of user information are defined in a privacy policy. Privacy policies may be used to proactively define how and when personal information may be released, to whom and, for how long. Personal information may be compromised if appropriate data confidentiality of the information is not maintained, such as during transit or storage. As a result, mechanisms designed to ensure confidentiality might be used to reduce the risks of inappropriate information disclosure.

7.4.3.2 Trust and Key Management

An infrastructure must be arranged so that system entities can exchange secret information securely and trust one another. This can range from establishing a public key infrastructure (PKI), using a symmetric key system such as Kerberos, or other approaches appropriate to the deployment.

The security and reliability of any communication process is directly dependent on the quality of key management and protection afforded to the keys used for authentication and encryption. The functions of key management are to provide secure key generation, storage, renewal, revocation, exchange and use. The security of encrypted or authenticated data is strictly dependent upon the prevention of unauthorized disclosure, modification, substitution, insertion and deletion of keys. If these are compromised, the security of the data can no longer be assured.

Key management methods can be either manual or automatic distribution. In the automatic key management, the technologies include centralized symmetric key management systems such as Kerberos as well as distributed key management systems such as using public key technology systems, which includes public key infrastructure (PKI). Differences in the methods and technologies result in different mechanisms, but the goals are the same, to reduce the risks of inappropriate key use and to provide a uniform, scalable system for key management.

7.4.3.3 Availability

Another security requirement is that a service be available when requested by users or entities. Denial of service attacks can disable a server from providing services to legitimate users by overloading it with request processing or other incoming events that overwhelm the services. In the lighter case, the attack may also degrade the performance of the services provided.

7.4.4 Deployment Considerations

A security policy and deployment must take into account deployment considerations, such as whether intermediaries or proxies are used, since this has implications on which security mechanisms are appropriate.

7.4.4.1 Threat Context: Security Persistence and Protocol Context

Security threats may be addressed in different ways depending on assumptions about the threat.

Security may be applied at different protocol levels, including the IP transport layer (e.g. IPSec, HTTPS), message (e.g. secure SOAP) and application (e.g. XML Digital Signature and XML Encryption of XML payload elements). The choice of level has implications for security depending on the use case.

The main aim is to apply security features to a transaction between two parties in the interaction so that there is adequate protection with respect to the underlying trust and business models. In some cases, these models might dictate the form of the most convenient and natural solution (e.g., at what level to apply which security feature). Once the application designer has an understanding of the underlying trust model within which an application is to be deployed, the next question is the level of granularity with which security protection is to be applied to the entities involved, the persistence required and the appropriate tradeoffs. That is, should it be applied on a per-application, per host, per site, or per domain basis. Does integrity

or confidentiality apply to an entire message, or portions of a message? Does a record of a signature need to be maintained, and for how long? The choice has to achieve a balance of efficiency and scalability with an appropriate level and type of protection.

Transport level security such as SSL/TLS, depicted in Figure 7.3 below, offers the benefit of ease of deployment but the risks associated with the lack of end-to-end security. Transport layer security establishes a security context between the two nodes running SSL/TLS, but not at the application level. Once an SSL server has received an SSL payload, that payload no longer has the integrity or confidentiality features applied during SSL transport. This may be adequate in some cases, but when content is stored and routed between applications, it may not be suitable.

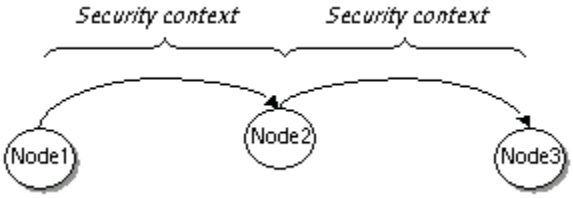


Figure 7.3: Transport Security

End-to-End security (see Figure 7.4) is critical when messages are passed through intermediaries, as may be expected with SOAP intermediary processing, as well as with proxies, gateways and delegated functions. If a message must pass through multiple application-level message servers, then transport layer security is terminated at each one. This means that all content is decrypted at each intermediary server and integrity and confidentiality cannot be guaranteed while the information is passing through the server. There is also no guarantee that adequate security and authentication will be provided at each link.

In general, transport layer security cannot be as effective as application level security because of a lack of application level understanding [Saltzer]. End-to-end security creates a security context between the two application endpoints, allowing security benefits regardless of the processing in between.

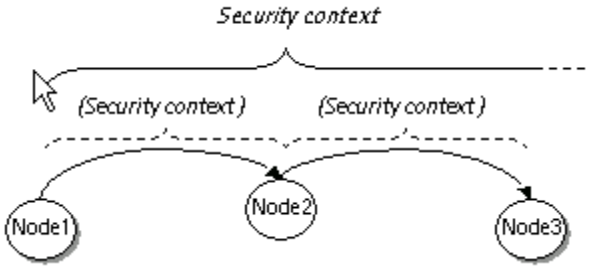


Figure 7.4: End-to-End Security

7.4.4.2 Proxy Security Considerations

The ability to support source sender to end receiver security services (end to end) can be complicated by a proxy that acts as a source SOAP sender on behalf of an element that does not support web services protocols (such as a mobile terminal for example). Although SSL/TLS can be used from the terminal to the proxy and web services security from the proxy to the end receiver (or SSL/TLS if only one link is traversed) security services are not in effect at the proxy, creating a security weakness at the proxy.

One potential solution is to use application level security (XML Signature, XML Encryption) but this may not be a solution for a terminal that is unable to support web services directly, since such a limited client may also not support XML security.

The need and means to address this issue associated with use of such a proxy depend upon assumptions about threats, adversary capabilities and risks.

8. Practical deployment patterns for Web Services

An understanding of practical deployment architectures is essential for employing Web Services in service provider networks. To that end, this section offers some common deployment patterns that have been synthesized from descriptions of Web Service deployments encountered in practice. It is hoped that patterns such as those described here can be used as a common language when discussing the sorts of deployment architectures that will most likely be encountered in OMA Web Service-based environments.

A pattern is a solution that can be reapplied time and again to solve a problem that the pattern identifies. In what follows, certain Web Service deployment patterns have been identified that cover the vast majority of common deployments encountered in practice. The problems that each of these patterns solves are those that recur repeatedly in every practical deployment of Web Services, namely

- Where to place trust boundaries
- How to ensure that messages can be funnelled through some controlled and auditable ingress to each trust domain

The deployment patterns for Web Services that are described are categorized from the point of view of

- The Web Service requester (WSR)
- The Web Service (WS)
- Any additional Web Service role required by the pattern.

For each case, the implications of choosing each pattern and their areas of applicability are provided.

Note that it is not the goal of this section to

- Describe Web Service *communications or message exchange patterns* like request-response, one-way, call backs etc., or Web Service *interaction patterns* such as register, discover and bind.
- Provide tutorial information on Web Services or design patterns
- Describe pattern implementations or describe end user and service provider interactions
- Describe implementations of these abstract patterns, or prescribe any particular implementation.
- Provide an exhaustive list of every conceivable pattern, covering all edge cases

While not exhaustive, this section provides the commonest cases that form the building blocks of deployment architectures being used in current practice.

Each pattern is given a name which is descriptive of its function, and, in the absence of industry specific nomenclature for such patterns, every attempt has been made as far as possible to not use terms that are (too) overloaded. However, as with all attempts at naming, the name may not be fully comprehensive and the textual description should be the arbiter of the meaning that is intended.

Note that the patterns described below may be mixed and matched in any realistic deployment because there are always multiple problems to solve. Thus, there is no implication that each pattern is stand alone, or a silver bullet, or a definitive or complete solution for the varied problems faced in practical deployments.

Note also that while many of these patterns are supported by current deployments in the IT industry, the techniques (protocols, mechanisms) used are proprietary. Such cases are pointed out where applicable so that the use of such patterns (no matter how attractive) may be deferred to future releases of the OWSER when one may expect standardized support or widespread industry consensus on the techniques that implement them

A pictorial representation of each pattern is provided in the pattern descriptions. Double circle notation (around the WS) indicates that it is discoverable and also the ultimate receiver of the Web Service request message. Color variation among the

circles in a diagram is a presentation device used to emphasize visually the distinctions that are captured in the text labels. There are no pattern semantics expressed or implied by the use of any specific color.

8.1 The Routing Pattern

Figure 8.1 shows an example of the routing pattern.

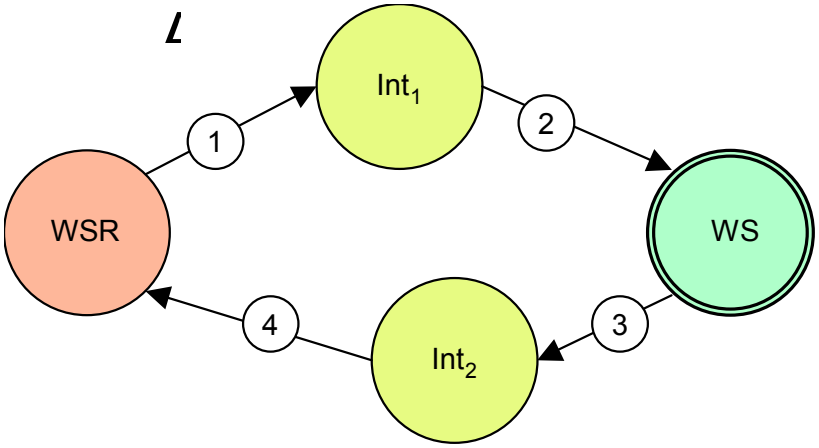


Figure 8.1 The Routing Pattern

Description:

The Routing pattern allows a Web Service request from a WSR and response from a Web Service to be routed through a number of Web Service intermediaries. (The number of intermediaries is irrelevant to understanding the Routing pattern.) (Note also that in Figure 8.1, the messages marked 1 and 2 correspond to the Web Service request and 3 and 4 to the response.) An intermediary may be within the same network as the WSR, within the same network as the Web Service, or in a network exclusive of either the WSR or Web Service networks.

Problem addressed:

How can one make Web Service messages follow a particular "path" so that Web Service intermediaries may be interposed to provide value-added services?

WSR point of view:

- It is a request-response interaction, although
- The response message *may be* received from a different Web Service node than the one to which the request message was sent, and
- The WSR may know the full or partial route to the Web Service, which it includes as a part of the Web Service request message, or
- It may always send a Web Service request to a given Web Service intermediary (known perhaps through configuration data), leaving further onward routing to that Web Service node

WS point of view:

- It is also a request-response, but again
- The response message *may be* sent to a different Web Service node than the one from which the request message was received
- The WS may know the full or partial route to the WSR which it includes as a part of the Web Service response message, or
- It may be informed, in the received message, of a reverse message path to follow for the response message, or
- It may always send a Web Service response to a given Web Service intermediary (known perhaps through configuration data), leaving further onward routing to that Web Service node

Intermediary point of view:

- It may, in addition to any other service that it provides, be able to dynamically route the Web Service message onward based on routing information in the Web Service messages
- It may be configured to always forward a Web Service message to an "adjacent" intermediary Web Service node
- It may use a different underlying protocol binding to transfer the Web Service request on the sending side than used on the receiving side.

Implications:

- The most general application of the Routing pattern requires the support of a Web Service routing protocol in Web Service intermediaries and preferably also in the WSRs and WS
- In the most general case, the WSR receives the response on a different connection, which may be difficult to achieve in some deployment scenarios involving firewalls
- Routing may always be pre-configured in the WSR, WS, and intermediaries, i.e., the route is not indicated in the message itself, but this is naturally a less flexible alternative
- There is at this time no standardized solution in the industry for indicating routing paths in Web Service messages. Proprietary solutions exist.
- Transport security cannot be used if the Intermediaries, WS and WSR all belong to different trust domains, as security has to be terminated at each Intermediary.
- Message based security mechanisms may allow different parts of the message to be accessed by the different actors on the message path.
- The addition of intermediaries can have an impact on response times, or the Web service performance can indirectly be impacted by intermediary failures, etc.

Areas of applicability:

The situations when one may wish to employ the Routing pattern include ones

- Where business logic is distributed, often dynamically, at Web Service intermediaries and control over the message path is needed so that specific intermediaries are not overlooked.
- When message path optimizations may be necessary, such as sending future requests to a "closer" provider
- Where intermediaries in the same trust domain as the WSR are needed to impose/add additional checks on outbound Web Service messages.

Related patterns:

None identified.

8.2 The Gateway Pattern

Figure 8.2 below shows an example of the typical Gateway pattern. In the figure, the gateway and the Web Service belong to the same trust domain. The WSR does not.

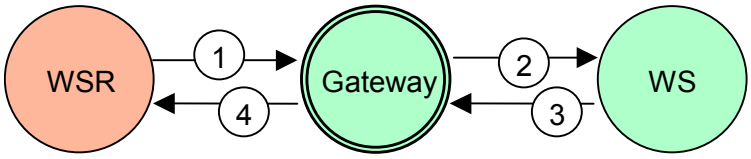


Figure 8.2: The Gateway Pattern

Description:

The Gateway sits in front of a WS, whose service interface is exposed at the Gateway. That is, the operator of this trust domain exposes, in the service registry accessible to those outside the trust domain, the Web Service provided by the WS at

the address of the Gateway. This allows for access to the services of the WS by WSRs outside the trust domain of the Gateway. The Gateway pattern allows for incoming requests to terminate at a single Web Service "node" to better concentrate the ability to verify aspects of incoming requests. It also hides aspects of the deployment structure of the WS, such as its URI. The termination of the request at the Gateway allows for some business logic (such as, for example, security policies) to be applied to the request before being relayed to the WS implementing the actual service logic.

Problem addressed:

- How may a service provider provide a single point of contact to all WSRs outside its trust domain for all the Web Services that it hosts?
- How can one hide the deployment structure of a WS to allow for future re-structuring without changing the service contracts towards WSRs?

WSR point of view:

- It is necessary that all information - in particular, additional information such as credentials, assertions, etc., - that is necessary for successful processing of the request be fully populated by the WSR or any intermediary to allow the message to be fully inspected at the gateway.
- From the WSR point of view, the Gateway ***is*** the WS , however the WSR has no knowledge that this is the case, since the WS configuration is transparent to the WSR. The WSR's discovery process returns the Gateway as the access point to the WS's interface.

WS point of view:

- The WS performs the core service logic, but some – typically deployment-specific functions such as security etc - are provided by the Gateway.
- From the WS point of view, the Gateway is the WSR. (Note that message 2 is a separate Web Service request than 1, even though it may resemble the former in many ways.)

Gateway point of view:

- The Gateway pattern hides all back-end architectures, protocols (see also the Adapter pattern in section 8.5) and deployments from being exposed to the WSR.
- The Gateway may use a different underlying protocol binding to transfer the Web Service request to the WS than used on the receiving side.
- The Gateway may implement some business logic such as authentication, authorization and privacy handling, and, if used in conjunction with the Adapter pattern, may actually involve quite a lot of logic (state handling etc).

Implications:

- Implementing the Gateway pattern requires that the access point of all WSs *that need to be exposed to external WSRs* terminate at the Gateway. This means that the administrator of this domain is responsible for populating the externally visible service directory (UDDI) with the service descriptions of the WS(s) re-exposed by the Gateway, and that the actual WSs or other (i.e., non Web Service) systems in the back-end are not visible to a WSR.
- Note that the internal service directory could offer the access point of the WS to WSRs within the internal trust domain of the WSP, i.e., they need not be forced to use the gateway.

- The use of the Gateway pattern does not imply a monolithic deployment or product solution, but care must be taken in deployments using this pattern to ensure scalability and performance

Areas of applicability:

- The Gateway pattern is best suited in deployment environments where there is a need to provide a single, logical ingress to resources exposed as Web Services for purposes of security, audit etc. The actual implementations may use legacy (i.e., non Web Service) techniques/solutions.
- A Gateway may act as a security bridge between different trust domains, offering the full weight of a WS-Security based interface to external WSRs but using HTTPS internally.

Related patterns:

The Adapter pattern

8.3 The Proxy Pattern

Figure 8.3 below shows an example of the Proxy pattern.

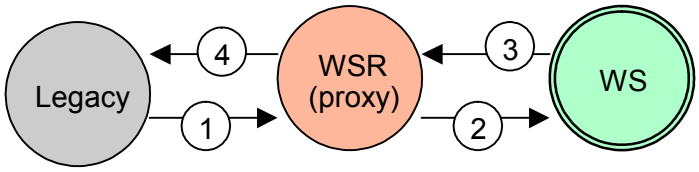


Figure 8.3: The Proxy Pattern

Description:

The Proxy pattern allows a system that does not have WSR capabilities to make use of services offered by a WS.

Problem Addressed:

Is it possible for systems that are not Web Service aware to take part in Web Service interactions?

WSR (proxy) point of view:

- The WSR has to adapt the interactions with the legacy system to the interface offered by the WS

WS point of view:

- The WS does not know (or care) if the request is proxied by the WSR

Legacy system point of view:

- The legacy system interacts with another system (the WSR) belonging to its own legacy domain that provides a service *similar* to the WS.

Implications:

- Typically, the WSR (proxy) will belong to the same trust domain as the legacy system. In some cases the legacy domain has its own security mechanisms that could be used between the legacy system and the WSR. However, it is usually not possible to have end-to-end security between the legacy system and the WS if the WSR (proxy) belongs to its own trust domain, as security has to be terminated when crossing technology domains such as it is in this case.
- Typically, the WSR (proxy) is configured to proxy for one particular type of WS and is seen by the legacy system as a service belonging to the same legacy domain as the legacy system but providing a service similar to the WS.
- The legacy system will use the service discovery mechanism appropriate to the legacy domain to find the proxy (WSR).

Areas of Applicability:

- This pattern is likely to be used for systems that cannot provide the required WSR capabilities.

Related patterns:

None identified.

8.4 The Interceptor Pattern

The figure below shows an example of the Interceptor pattern.

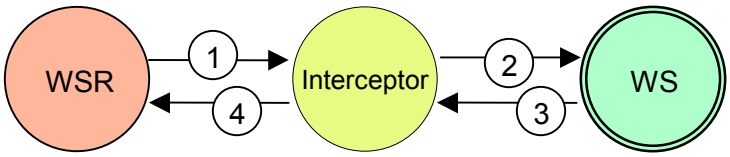


Figure 8.4: The Inteceptor Pattern

Description:

The Interceptor pattern represents a "lighter" version of the Gateway pattern in that it does not act as the ultimate receiver for Web Service request messages. That is, the service registry exposes the Web Service provided by the WS at the WS's address. If the Interceptor belongs to the same trust domain as the WS, it acts as an "invisible" gateway by intercepting all Web Service requests from an external trust domain, possibly for audit, control or filtering, before forwarding them to the WS. If the Interceptor belongs to the trust domain of the WSR, it can also do similar functions for outbound messages from that domain.

Problem addressed:

Is it possible to expose to external WSRs each Web Service at their original access points, while still retaining the ability to control access to them through some common ingress?

WSR point of view:

- It is necessary to include all information – in particular, additional information such as credentials, assertions, etc., – in the Web Service request message that is necessary for successful processing of the request.
- The WSR is normally not aware of the interceptor.
- The WSR's discovery process has returned an access point to the actual WS.

WS point of view:

- The WS needs to be discoverable.

- The WS is normally not aware of the interceptor.

Interceptor point of view:

- The Interceptor may use a different underlying protocol binding to transfer the Web Service request to the WS than used on the receiving side.

Implications:

- Interceptors are typically implemented on top of L3/L4 switches.
- It is possible that Interceptors will not be transparent for all parties if transport level security is used.

Areas of applicability:

- The Interceptor pattern can be used to funnel all incoming Web Service traffic through a particular Web Service node, in which security and access control may be applied.
- The Interceptor pattern can be used when some additional service logic needs to be inserted transparently in front of a WS without having to affect the WSR's perception of the WS's end-point and/or without having to change the publish/lookup mechanism for the WS/WSR.
- The Interceptor pattern can also be used for adding additional logic to outbound messages from a WSR without having to change the WSR's service-specific interaction with the WS.
- The Filter pattern may be used in conjunction with the Interceptor pattern to "route" Web Service requests based on some filter criteria to the appropriate instance of the WS.

Related patterns:

The Filter pattern.

8.5 The Adapter Pattern

The figure below shows an example of the Adapter pattern.

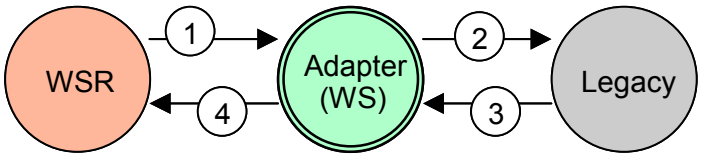


Figure 8.5: The Adapter Pattern

Description:

The Adapter pattern converts a Web Service based interface to that offered by a legacy (i.e., non Web Service based) system. The Adapter typically belongs to the same trust domain as the legacy system. This pattern could be seen as the reverse of the Proxy pattern.

Problem addressed:

Is it possible to offer a Web Service interface to a system that supports a legacy (i.e., non Web Service based) interface/protocol?

WSR point of view:

- It is necessary to include all information in the request message – in particular, additional information such as credentials, assertions, etc., - that is necessary for successful processing of the request.

WS (Adapter) point of view:

- The WS performs the necessary protocol conversion to the legacy interface.
- The WS has to be published in a service directory (i.e., be discoverable)

Implications:

- The Adapter pattern is commonly used in conjunction with a Gateway pattern.
- The implications of matching interfaces at different levels of granularity need to be understood.
- To re-expose a web service interface for a legacy system, maybe using a different interaction style than supported by the legacy system, usually demands additional logic and state-handling.

Areas of applicability:

- The Adapter pattern is used when there is a need to make services offered through legacy interfaces available to WSRs.
- Most service logic today resides in legacy systems and using the Adapter pattern may be the most cost (time and money) effective way to move to a web service paradigm.

Related patterns:

The Gateway pattern

8.6 The Delegate Pattern

The figure below shows the Delegate pattern.

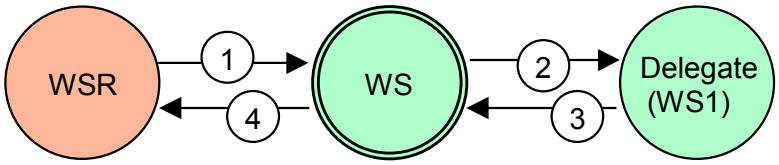


Figure 8.6: The Delegate Pattern

Description:

The Delegate pattern allows a Web Service to delegate the processing of some aspects of a Web Service request to another (helper) Web Service (WS1 in Figure 8.6). The (helper/delegate) WS belongs to the same trust domain as the advertised WS.

Problem addressed:

Is it possible to evolve the services of an existing WS by delegating the processing of some functions to another WS?

WSR point of view:

- It is necessary to include all information in the Web Service request message – in particular, additional information such as credentials, assertions, etc., - that is necessary for successful processing of the request.
- The WSR only has a trust relationship with the WS (if applicable), and is not aware of the Delegate (WS1)

WS point of view:

- The WS has to be discoverable.
- The WS makes a separate Web Service request for those functions that it delegates. (In Figure 8.6, 2 and 3 form a separate Web Service request-response pair based on the interface provided by the Delegate (WS1))

Implications:

- This pattern may be employed in a deployment context where some functions may be delegated for administrative reasons, even if the WS is capable of performing them.
- The trust implications of delegating to another trust domain (i.e., WS and WS1 belong to different trust domains) needs to be carefully understood, particularly in the case when end-user data may be involved. That is because the WSR and the WS establish some trust relationship for their interactions which may not apply to WS1. Thus, the safest usage of delegation is within a single trust domain.

Areas of applicability:

- There may be occasions when an existing service interface may be evolved by offering a newer interface supporting additional functionality, where the actual processing (business logic) of these additional functions is delegated to another WS.

Related patterns:

None identified.

8.7 The Filter Pattern

The figure below shows an example of the Filter pattern. Note that this pattern applies to *individual, inbound* messages from a WSR to a WS, shown as messages 1, 2, and 3. In **Figure 8.7**, the three WS_1 elements show three instances of an implementation of the same interface, each catering to a particular Web Service request message.

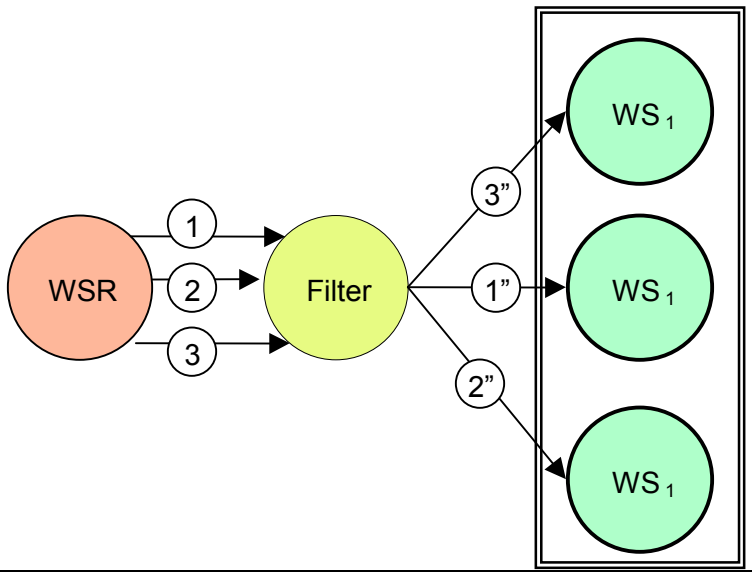


Figure 8.7: The Filter Pattern

Description:

In many practical scenarios, Web Services are deployed in a server farm typically behind a load balancer. In such a scenario, a particular WS is not tied to a particular WSR. However, because a message from a particular WSR contains some specific data that needs to be accessed by the WS, or because the particular request is tied to some security or transaction context, it is necessary to divert the message to a particular instance of a WS that is tied to that data/context.

A Filter pattern allows a Web Service request to be routed to one of several instances of a WS based on some "filter" criteria applied to the message's contents/data. The Filter typically belongs to the trust domain of the WS.

Problem addressed:

Is it possible to route a Web Service request to a different WS (providing the same interface) based on the contents of the request?

WSR point of view:

- The interaction is a (series of) request-response(s) but it is necessary to include all information – in particular, additional information such as credentials, assertions, etc., - that is necessary for successful processing of the request.

WS point of view:

- The access point of the WS is that of the Filter.

Filter point of view:

- The logic for filtering and dispatching needs to be provided either through configuration or dynamically.

Implications:

- The Filter pattern is typically used in conjunction with the Gateway pattern.
- It may also be possible to use the Filter pattern at a Routing intermediary.

Areas of applicability:

- The Filter pattern is typically used to do some form of "load sharing" or "data partitioning" so that requests for a particular Web Service may be filtered on the presence and/or value of certain data elements in a request and diverted to an appropriate WS that hosts the resource to which the data in question is relevant.
- The Filter pattern may be used to filter Web Service messages on namespaces used within the message and direct requests for different versions of an interface to the appropriate implementation.
- Filtering may be used to redirect Web Service request messages based on (end user or WSR) identity information carried in the message.

Related patterns:

The Gateway pattern

8.8 The Orchestrator Pattern

The figure below shows an example of the typical Orchestrator pattern.

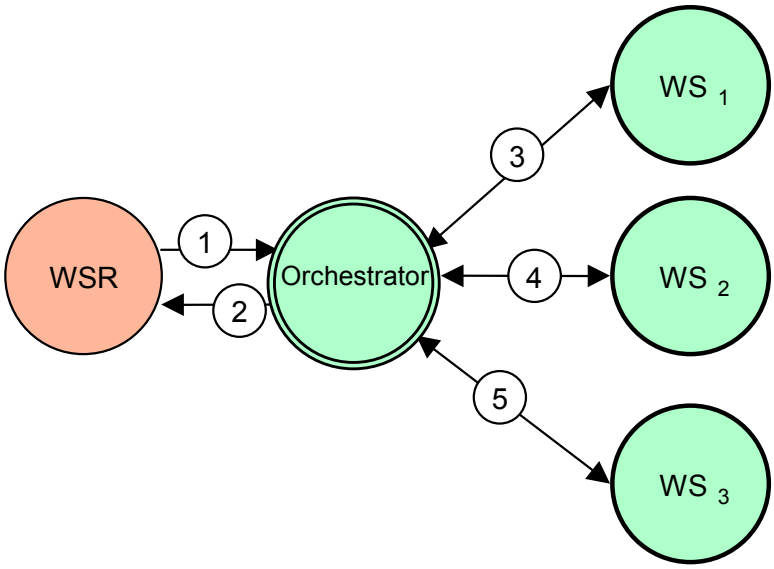


Figure 8.8: The Orchestrator Pattern

Description:

The Orchestrator pattern offers a composite service interface built from the coordination of a set of individual Web Services. Requesters see the composed service, not the component services, because it is the composed service interface that is advertised in the *external* service registry. In Figure 8.8, the WSR interacts with the Orchestrator service interface. The Orchestrator provides its services by interacting with other Web Services, WSs 1, 2 and 3, in this example, and possibly other (i.e., non Web Service based) systems not shown in this figure.

This should be seen in contrast to the Gateway pattern, because a Gateway typically does not change the core service logic of the WS that it exposes. An Orchestrator typically adds additional logic beyond that of merely orchestrating the individual requests to the component services.

Problem addressed:

Is it possible to provide a **new** Web Service that is the composition of two or more Web Services?

WSR point of view:

- It is necessary to include all information in the Web Service request message – in particular, additional information such as credentials, assertions, etc., - that is necessary for successful processing of the request.
- For the WSR, the Orchestrator *is* the WS. It does not know of service composition, or how the WS provides its functions.

Component WS point of view:

- It can offer a simple interface towards its core functions and plays a part in a larger composite service. (However it most probably does not know how it is used in a composed service.)
- The request is received from an Orchestrator in the same trust domain, it may safely assume that the request has been properly authenticated and authorized.
- From the component WS point of view, the Orchestrator is the WSR for its service.
- The component WSs have to be discoverable by the Orchestrator, in general.

Orchestrator point of view:

- It executes the business logic to fulfill the composite Web Service, i.e., in addition to any value-added functions it may provide (e.g., security for the component WSs in its trust domain), it provides the correct sequence of separate Web Service requests to component WSs to achieve certain functions.
- The Orchestrator should publish its (composite) service in the external service registry (i.e., it is discoverable to WSRs).
- An Orchestrator should provide error logic for handling failures or errors in its component services.

Implications:

- The same (component) WS could be part of more than one orchestrated service.
- Note that some of the component WSs could handle value-added functions such as integrity, authentication, etc.
- The composed service is typically very different than that offered by the component services.
- Even though the component WSs may not be aware of the overall service of which they are a part, their services may be used under an overall transactional context.

Areas of applicability:

- As component WSs can be reused in many orchestrated services, this allows great flexibility in creating complex services from simple ones.

Related patterns:
None identified.

8.9 The Referral Pattern

The figure below shows an example of the Referral pattern.

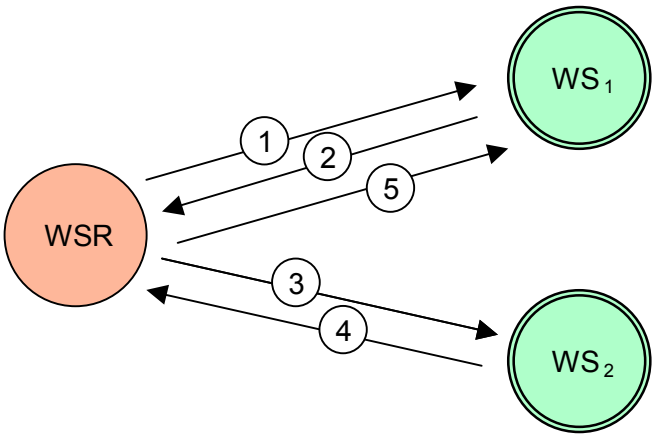


Figure 8.9: The Referral Pattern

Description:

A WSR when interacting with a WS is referred to another WS to retrieve some data before the initial WS can continue/complete the request. For example, in Figure 8.9, when the WSR invokes WS1, WS1 recognizes that the request is not properly authorized, say, and therefore refers the WSR to WS2 to retrieve a credential before coming back to WS1. (A credential could be some type of assertion stating that some necessary pre-condition has been satisfied.)

Problem addressed:

- How can a WSR, when sending an incomplete request to a WS be referred back to another WS to retrieve additional information to be able to complete the request.

WSR point of view:

- The WSR has to have mechanisms to manage the referral. (Thus far we have seen patterns, which provide – except perhaps for the Routing pattern – a simple interface from the WSR's perspective. The Referral pattern, as the name suggests, requires a more active role for the WSR.)

- WSR does not have to know the order or sequence or indeed that it needs to be referred elsewhere, as each self-contained response tells it what to do next. In other words, referral is a dynamic interaction pattern.

WS point of view:

- The initial WS must be able to indicate the secondary WS to whom the WSR must be referred.
- A WS may need to verify the received credential with the provider of the credential. This may be done using additional Web Service interactions between WS1 and WS2.

Implications:

- An agreement on how this referral is conveyed back to the WSR has to be explicit in the service interface exported by the WS.
- We are not aware of any standardized *automated* mechanisms (e.g., in the same manner as HTTP redirects) by which such referrals are handled implicitly at the WSR.
- Referrals can be done once and subsequent requests may be directly handled by the WS, or there may be trust requirements that require that every request be referred to another WS before execution.
- Referrals can be done on a case-by-case basis, based on the trust requirements of the service. For example, a WS may refer untrusted WSRs to an authorization service while trusted WSRs may be serviced directly.

Areas of applicability:

- Any scenario where a service granting credential is required by a WSR before being allowed to access the WS.
- When certain functions related to the execution of a Web Service are delegated to other service providers (possibly in other trust domains outside that of the WSP), the Referral pattern can be used to complete the interaction.

Related patterns:

The Sequence pattern

8.10 The Sequence Pattern

The figure below shows an example of the Sequence pattern.

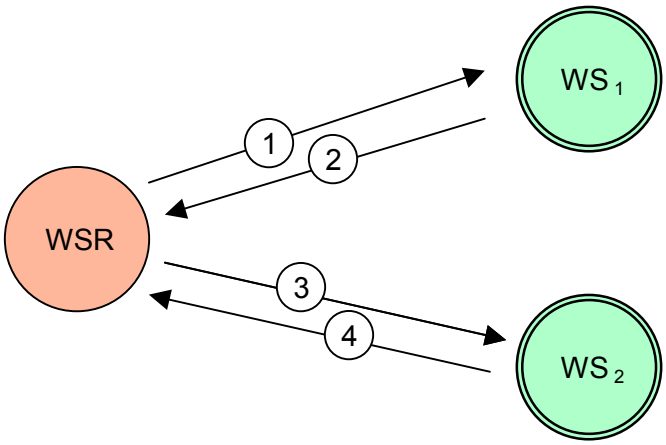


Figure 8.10: The Sequence Pattern

Description:

The Sequence pattern is related to the Referral pattern. A WSR needs to interact with more than one WS in a defined sequence to complete a request. Typically some parts or responsibilities for the service are distributed over a number of WSs and the WSR has to invoke each and every one of those WSs in some order to accomplish the task at hand.

Problem addressed:

How can a WSR be the controlling entity to obtain a service that requires the invocation of multiple Web Services?

WSR point of view:

- The WSR discovers all WSs taking part in the service offered. In addition, the WSR needs to know the sequence in which the WSs should be invoked.

WS point of view:

- The WSs need to be discoverable.
- In this pattern the WS, besides offering a simple interface towards its core functions, *may* expect some credential (from earlier invocations of WSs in sequence) to carry out its request. This is when the WS in question is aware that it is a part of some sequence of invocations. However this is not necessary for this pattern to operate.

Implications:

- Note that the Sequence pattern places a lot of responsibility on the WSR. There is a similarity to the Orchestrator role in that the WSR "orchestrates" the services that it needs, rather than use the single service interface of a composite service offered by the Orchestrator.
- There appears to be no clear or consensus view of notational support in the industry to describe such sequencing.
- In contrast to the Orchestrator case, the WSR has to take on the burden of composing the service by determining the WSs that it needs to compose the ultimate service, and the order of invocation by which to achieve it.

Areas of applicability:

- Any scenario where a WSR needs to go to an Authorization service to request a service granting token before being allowed access the WS.

Related pattern:

The Referral pattern

8.11 The Workflow Pattern

The figure below shows an example of the Workflow pattern. Note that in Figure 8.11, the lines 1 through 4 reference individual Web Service requests, and not a request or response as in previous similar diagrams. (In other words, 1 is a Web Service request from WSR1 to a WS2, 2 is a *different* Web Service request from WSR2 to WS2, etc.)

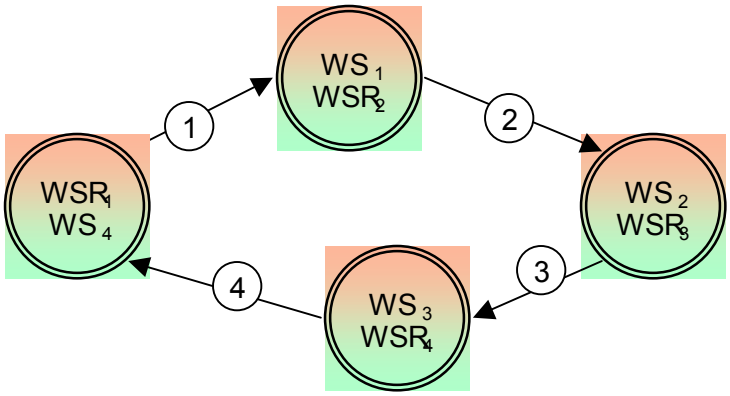
Problem addressed:

How can business systems exposing capabilities as Web Services, typically distributed over different business (and trust) domains, work together to complete some overall task?

WSR point of view:

- Each Web Service node in the workflow is both a WSR as well as a WS for a (different) service.
- Each WSR in the workflow knows the next WS in the chain as this is defined as a part of the overall workflow.

WS point of view:



- Each Web Service node in the workflow is both a WS as well as a WSR for a (different) service.

Figure 8.11: The Workflow Pattern

Implications:

- There are several proposals in the industry on how to describe a workflow. None have yet been standardized.
- Note that each interaction in the workflow may make use of one or more of the previously defined patterns.

Areas of applicability:

- Service management applications can be arranged as a workflow.

Related patterns:

Any of the previous patterns may be interposed on any leg of the workflow.

9. Mobile Terminal Considerations

It is understandable that initial developments in the Web Services space tend to concentrate on server-to-server interactions in fixed networks. Nonetheless, in order for Web Services to become a universal communications paradigm, its scope should also include mobile devices with wireless connectivity.

The use of Web Services on mobile devices encompasses three basic usage patterns.

1. Invocation of network-based Web Services (e.g. push notification service, location service, billing service) from mobile devices.
2. Hosting of Web Services (e.g. personal authentication service, payment service) on mobile devices.
3. Mobile-to-mobile communication using Web Service technology end-to-end (effectively a combination of the first two).

We suggest that these usage patterns will become partly relevant following the more conventional adoption of Web Services for business integration tasks.

Basic protocol conventions and other characteristics of the Web Services infrastructure should remain largely agnostic as to whether mobile devices are involved as service consumers or service providers (platform neutrality). However, we foresee architectural implications and room for specific practices in areas such as:

- *Small-footprint implementations of Web Service suitable for mobile devices*: Limited resources on the mobile device require optimized implementations of Web Service stacks, including SOAP message handling, message dispatching, and XML serialization/deserialization. For instance, compressed over-the-air encodings for serialized SOAP traffic may help to reduce bandwidth requirements and/or parsing overhead on (constrained) mobile devices. The latter holds because some compression schemes may rely on passing pre-parsed versions of the XML infoset (e.g. sequence of SAX events) instead of verbatim XML text. There may be architectural implications as well, such as the need for additional proxies.
- *Interoperability issues*: SOAP stack implementations may not fit all sizes, and there are different runtime environments to consider. Furthermore, device constraints may necessitate particular interoperability profiles for certain classes of mobile devices.
- *Tuned programming model*: Unlike current bandwidth limitations, which we can expect will decrease over time, we expect that communication latencies over wireless links will remain significant even after the rollout of 3G networks. This requires a programming discipline that favours calls to coarse-granular methods and aggregation of functionality within such methods, in order to keep the overall number of over-the-air message exchanges low. It may lead to alternative (or restructured) versions of some APIs; for instance, formerly separate authentication steps may be folded into ensuing calls. Some automation through pre- and post-processing, or batch processing, for Web Services may be feasible to approach the programming model.
- *Scalable means of service advertisement*: A single, central UDDI registry is likely unable to accommodate the large aggregate number of Web Services that may be hosted on mobile devices; protocols for ad-hoc service discovery (along the lines of UPnP or Bluetooth SDP) or federations of registries with localized coverage may be more appropriate in terms of administration, resiliency, and performance. This will also allow other devices or services to discover/invoke Web Services originating from the device in a peer-to-peer fashion.
- *Location-based service discovery*: This will allow mobile devices to get instant access to Web Services that are particularly relevant to their present location/context, by taking device location information into account during the discovery process in order to list most relevant/proximate services (first). This may work, for example, by broadcasting lookup requests only within a local network perimeter or by consulting a (local) registry.
- *Service instance disambiguation:* When Web Services become ubiquitous many similar candidate service instances may be available inside close perimeters; for instance, there may be many on-device payment services in proximity of a single point of sale. Convenient and natural ways for identifying appropriate service instances are then required (e.g. relying on closeness or pointing rather than identification by cumbersome unique names).

- *Quality of Service:* It is plausible that Web Services on mobile devices also help to drive user interfaces. Quality of service parameters such as bounds on delays, bandwidth reservations, error levels, etc., become critical, because infringements directly impact user experiences.

- *Intermittent connection:* Mobile devices may be temporarily switched off or become otherwise unreachable during normal use. The communication software should offer means to mask such intermitted connectivity as far as possible, e.g. by replicating state information belonging to Web Services.

Appendix A. Change History (Informative)

A.1 Approved Version History

Reference	Date	Description
OMA-OWSER-Overview-V1_1	28 Mar 2006	OWSER 1.1 enabler package approved
OMA-OWSER-Overview-V1_0	15 Jul 2004	Approved for release with OWSER 1.0 enabler package