Reference Model for Service Oriented Architectures

Working Draft 08, September 9, 2005

Document identifier:

wd-soa-rm-08

Location:

http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm

Editors:

C. Matthew MacKenzie, Adobe Systems Incorporated, mattm@adobe.com
Ken Laskey, Mitre Corporation, klaskey@mitre.org
Francis McCabe, Fujitsu Limited, fgm@fla.fujitsu.com
Peter Brown, Individual, peter@justbrown.net
Rebekah Metz, Booz Allen Hamilton, metz_rebekah@bah.com

Abstract:

This Service Oriented Architecture Reference Model is an abstract framework for understanding significant entities and relationships amongst them within a service-oriented environment, and for the development of consistent standards or specifications supporting that environment. It is based on unifying concepts of SOA and may be used by architects developing specific services oriented architectures or for education and explaining SOA. A reference model is not directly tied to any standards, technologies or other concrete implementation details, but it does seek to provide a common semantics that can be used unambiguously across and between different implementations.

While service-orientation may be a popular concept found in a broad variety of applications, this reference model scopes itself to the field of software architecture.

Status:

This document is updated periodically on no particular schedule. Send comments to the editor(s).

Committee members should send comments on this specification to the soa-rm@lists.oasis-open.org list. Others should visit the SOA-RM TC home page at http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm, and record comments using the web form available there.

The errata page for this specification is at:

http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm.

Table of Contents

1 Introduction

1.1 What is a reference model

A reference model consists of a set of clearly defined basic concepts, axioms and relationships within a particular problem domain, independently of specific implementations, conventions, activities or organizations. The purpose of a reference model is to facilitate the design of systems, to establish a common set of terminology as it applies to the domain and to encourage best practice where possible.

In the field of information technology, specific *architectures* may be developed to promote a common approach to solving particular problems. A group of such architectures – Service Oriented Architectures or SOAs – have developed over recent years with a specific mission to improve interoperability and collaboration between otherwise dissimilar services.

SOAs have received significant attention within the software design and development industry in recent times resulting in many conflicting definitions of “service-oriented architecture”. Whereas SOA architectural patterns (or *reference architectures*) may be developed to explain and underpin the generic design template supporting a specific SOA, a reference model is intended to provide an even higher level of commonality, with definitions that should apply to *any* SOA at all. The value of such a reference model is as a foundational work that can and should be used to develop architectural patterns and promote effective discourse on derived works.

The goal of this reference model document is to define the essence of the service oriented architecture paradigm, and emerge with a vocabulary and a common understanding of SOA. It should provide a normative reference that remains relevant for SOA as an abstract and powerful *model*, irrespective of the various and inevitable technology evolutions that we experience in this industry and that will impact on specific SOA *implementations*.

1.2 Audience

The intended audiences of this document non-exhaustively include:

- Architects and developers designing, identifying or developing a system based on the service-oriented paradigm.
- Standards architects / analysts developing specifications that relate to or make use of the service-oriented paradigm.
- Chief Information Officers and other decision makers seeking a "consistent and common" understanding of service oriented architecture.

1.3 How to use the reference model

New readers are encouraged to read this reference model in its entirety. Concepts are presented in an order that the authors hope promote rapid understanding.

This section introduces the conventions, defines the audience and sets the stage for the rest of the document. Non-technical readers are encouraged to read this information as it provides background material necessary to understand the nature of reference models and their use.

Section 2 introduces the Reference Model for SOAs. First, the main axioms, key concepts and relationships between those concepts are introduced followed by more detailed sections on the main concepts: a *service* is defined along with *service description*. There then follows a section detailing interaction between services, followed by service policies and expectations. Finally, being the key to a service's actual use, the concept of service discoverability is introduced.

This section is provided for the benefit of multiple audiences:

- Non-technical readers may use this section to gain an explicit understanding of the core principles of SOA.
- Architects are encouraged to use this section as guidance for developing specific service oriented architectures. Section 2 and its subsections are designed to provide guidance for consistent logical divisions of components within architectures. It also helps architects adhere to the basic principles of service-oriented design.

Section 3 addresses what it might mean for an SOA-based system to be conformant with this reference model.

The glossary provides definitions of terms which are relied upon within the reference model specification but do not necessarily form part of the specification itself.

1.4 Notational Conventions

The key words *must*, *must not*, *required*, *shall*, *shall not*, *should*, *should not*, *recommended*, *may*, and *optional* in this document are to be interpreted as described in **[RFC2119]**.

References are surrounded with **[square brackets and are in bold text]**.

1.5 Relationships to Other Standards

Due to its nature, this reference model may have an implied relationship with any group that:

- Considers its' work "Service Oriented"; and/or
- Makes (publicly) an adoption statement to use this SOA Reference Model of this TC as a base or inspiration for their work when complete.

Additionally, there are a large number of standards and technologies that are related by the fact they claim to be or are "service oriented".

Any work that aligns with the functional areas of SOA such as the service, service description, advertising mechanism, service data model or service contract are likely to be directly related.

The reference model does not endorse any particular service-oriented architecture, or attest to the validity of third party reference model conformance claims.

2 The Reference Model

The reference model for service-oriented architectures describes concepts and relationships that are fundamental in describing SOA architecture patterns (i.e. SOA reference architectures) and specific SOA architectures applied to the solution of specific problems. In general, a service-oriented architecture represents a uniform means to discover and access distributed services that invoke functionality which produces desired effects with measurable preconditions and expectations. The services hide implementation details but have associated service descriptions to provide sufficient information to understand the technical and business requirements for invoking the service. The actual decision (or agreement) to invoke a service often is contingent on understanding and complying with those requirements.

While such a description of SOA gives a flavor for why it is of interest, it is not sufficient for understanding the primary SOA concepts that must be utilized in designing a SOA and effectively using an SOA. The remainder of this section introduces the main concepts and a detailed discussion of the concepts and their relationships are in the sections that follow.

2.1 Overview of model

A key concept of SOA is that of a **service**. In general, people and organizations create capabilities to solve or support the solution of problems they face in the course of their business. SOA is conceived as a way of making those capabilities visible and supporting standard means of access so the existing capabilities can be reused or new capabilities can be readily substituted to improve the solutions. A service is a means to access such capabilities.

To use a service, it is necessary to know it exists, what is accomplished if the service is invoked, how to invoke the service, and other characteristics to allow a prospective consumer to decide if the service is suitable for the current needs and if the consumer satisfies any requirements of the service provider to be permitted access. Such information constitutes the **service description**.

Services are accessed in order to achieve particular effects. However, the nature of SOAs are that there is an arm's length relationship between service providers and consumers. As a result, there is a distinction to be drawn between the public interactions with a service and the private actions of the service provider and consumer. An important reason for the scalability and security attributes of SOAs is that the distinction promotes independence between service participants. We can focus on the public aspects of using a service by examining the **conditions** of using a service and the **expectations** that arise as a result of using the service. We loosely associate the service conditions with the **service policies** and the expectations with **service contracts**.

Another key concept in SOA is that of **service interaction**. Although services are accessed in order to achieve particular desired effects, this is effected by exchanging information between service providers and consumers. Typically this is by exchanging messages using a standardized protocol; however, there are many modalities possible for using services

Finally we identify **discoverability** as a key concept of SOAs. Discoverability refers to the possibility and mechanisms by which service consumers and providers can be brought together. There are many possible mechanisms by which discoverability may be achieved; SOAs are not limited to registries or repositories of service descriptions although these are undoubtedly powerful means of achieving it. Discoverability itself is a key concept for SOAs.

2.2 The Reference Model in Detail

2.2.1 Service

A service is a mechanism to enable access to a set of capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description. A service is provided by one entity for use by others, but the eventual consumers of the service may not be known to the service provider and may demonstrate uses of the service beyond the scope originally conceived by the provider.

A service is invoked through a service interface, where the interface comprises the specifics of how to access the underlying capabilities. There are no constraints on what constitutes the underlying capability or how access is implemented by the service provider. Thus, the service could carry out its described functionality through one or more automated and/or manual processes that themselves could invoke other available services. A service is opaque in that its implementation is hidden from the service consumer except for (1) the information model exposed through the published service interface and (2) any information included as metadata to describe aspects of the service which are needed by service consumers to determine whether a given service is appropriate for the consumer's needs. The consequence of exercising a service is one or more real world effects. The effects may include

(1) information returned in response to a request for that information, including information returned as a result of prior interactions with the service,

(2) processing done in response to a request to change the state of identified entities, or

(3) some combination of (1) and (2).

Note, the user in (1) does not typically know how the information is generated, e.g. whether it is extracted from a database or generated dynamically; in (2), the user does not typically know how the state change is effected. In either case, the service consumer would need to provide input parameters defined (either required or optional) by the service and the service would return information, status indicators, or error descriptions, where both the input and output are as

[12] http://en.wikipedia.org/wiki/Turing_completeness

described by the information model exposed through the published service interface. Note that the service may be invoked without requiring information input from the consumer (other than a command to initiate action) and may accomplish its functions without providing any return or feedback to the consumer.

The description of the service concept has emphasized a distinction between a capability that represents some functionality created to address a problem or a need and the service that forms the point of access to bring that capability to bear in the context of SOA. It is assumed the capability was created and exists outside of SOA and one of the major benefits of SOA is enabling the capability to be applied to an expanded realm of relevant problems. In actual use, maintaining this distinction may not be critical (i.e. the service may be talked about in terms of being the capability) but the separation is pertinent is terms of a clear expression of the nature of SOA and the value it provides.

2.2.2 Service description

The service description represents the information needed to use a service. It may be considered part of or the complete set of the metadata associated with a service (see Section 2.2.3) but in any case, the service description overlaps and shares many common properties with service metadata. In most cases, there is no one "right" set of metadata but rather the metadata content depends on the context and the needs of the parties using the associated entity. The same holds for a service description. While there are certain elements that are likely to be part of any service description, most notably the information model, many elements such as function and policy may vary. However, the mechanisms to specify the service description should be represented through use of a standard, reference-able format that can accommodate the necessary variations and lend themselves to common processing tools (such as discovery engines) to make use of the service description.

While the concept of a SOA supports use of a service without the service consumer needing to know the details of the service implementation, the service description makes available critical information a consumer needs to decide to use a service and then to affect that use. In particular, a service consumer must know:

1. The service exists and is available;
2. The service performs a certain function or set of functions consistent with technical assumptions that underlie its functions;
3. The service operates under a specified set of constraints and policies;
4. The service will (to some extent) comply with policies as prescribed by the service consumer;
5. How to interact with the service in order to achieve the required objectives, including the format and content of information exchanged between the service and the consumer and the sequences of information exchange that may be expected.

Subsequent sections of this document will deal with these aspects of a service in details but the following subsections will describe their relation to the service description.

2.2.2.1 Service Availability

Item 1 refers to the key requirement for a service description to include sufficient data to permit a service consumer and service provider to physically exchange information. This might range from metadata such as the location of the service and what information protocols it supports and requires to whether the service is currently available or not.

2.2.2.2 Service Functionality

Item 2 relates to the need to unambiguously express the function(s) of the service and the real world effects (see section 2.4) that result from it being invoked. This portion of the description needs to be expressed in a way that is generally understandable by service consumers but able to accommodate a vocabulary that is sufficiently expressive for the domain for which the service provides its functionality. The description of functionality may include, among other possibilities, a textual description intended for human consumption or identifiers or keywords referenced to specific machine-process-able definitions. For a full description, it may be useful to indicate multiple identifiers or keywords from a number of different collections of definitions.

Part of the description of functionality may include underlying technical assumptions that determine the limits of functionality exposed by the service or of the underlying capability. For example, the amounts dispensed by an automated teller machine (ATM) are consistent with the assumption that the user is an individual rather than a business. To use the ATM, the user must not only adhere to the policies and satisfy the constraints of the associated financial institution (see sections 1.1.1.1 for how this relates to service description and section 2.4 for a detailed discussion) but the user is limited to withdrawing certain fixed amounts of cash and a certain number of transactions in a specified period of time. The financial institution, as the underlying capability, does not have these limits but the service interface it exposes to its customers does, consistent with its assumption of the needs of the intended user. If the assumption is not valid, the user may need to use another service to access the capability.

2.2.2.3 Policies Related to a Service

Items 3 and 4 relate to the service description's support for associating constraints and policies with a service and providing necessary information for prospective consumers to evaluate if a service will act in a manner consistent with the consumer's constraints and policies.

In some situations the consumer may similarly provide an indication of its constraints and policies to support a service's need to do a similar evaluation of suitability. Thus, both prospective consumers and providers are likely to use the service description (and the consumer description) to mutually establish what section 2.3.3 refers to as the *execution context*.

2.2.2.4 Service Interface

The service interface is the means referred to in Item 5 for interacting with a service. It includes the specific protocols, commands, and information exchange by which actions are initiated that result in the real world effects as specified through the service functionality portion of the service description.

The specifics of the interface are syntactically represented in a standard reference-able format. These prescribe what information needs to be provided to the service in order to exercise its functionality and/or the results of the service invocation to be returned to the service requester. This logical expression of the set of information items associated with the consumption of the service is often referred to as the service's information model. Note, specifying the particulars of the standard reference-able format is beyond the scope of the reference model but requiring that mechanisms be available to define and retrieve such definitions are fundamental to the SOA concept. Also note that the service may be invoked without requiring information input from the consumer (other than a command to initiate action) and may accomplish its functions without providing any return or feedback to the consumer.

While this discussion refers to a standard reference-able syntax for service descriptions, we do not specify how the consumer accesses the interface definition nor how the service itself is accessed. However, it is assumed that for a service to be usable, its interface must be represented in a format that allows interpretation of the interface information.

2.2.2.5 An Example of Using Information Contained in the Service Description

The following example may help to clarify the concepts related to service and service description. To access electricity generated by the local electric utility, the service interface is the wall outlet and to use the service I need to understand what kind of plug fits the outlet. The utility assumes I will plug in devices that are compatible with the voltage they are providing and my assumption is I can safely plug in devices without these being damaged. If I am a home or business user, a constraint is I must establish an account and the contract I have with the electric utility is they will meter my usage and I will pay at a rate they prescribe. If I am a visitor to someone with a contract, the utility does not have a contract with me (and I do not have to satisfy the initial account constraint) but I still must be compatible with the service interface. The utility policy may be that in the event of high use by the community, the utility may reduce voltage or institute rolling blackouts. My implied policy is I may complain to my legislative representative if this happens frequently. The resource is the utility's ability to generate and distribute electricity, and the service is my getting access to that electricity. The resource would exist if every device was required to be hardwired to the electric utility's equipment but this would result in a very different service with a very different interface.

2.2.3 Descriptions and Metadata

One of the hallmarks of a Service Oriented Architecture is the degree of documentation and description associated with it; particularly *machine processable descriptions* – otherwise known as *metadata*.

The purpose of this metadata is to facilitate integration, particularly across ownership domains. By providing public descriptions, it makes it possible for potential participants to construct applications that use services and even offer compatible services with minimal human-level contact between them.

2.2.3.1 The roles of description

An important additional benefit of metadata – as opposed to informal natural language descriptions – is that it potentially permits automation in the creation of software. Both service providers and service consumers can benefit from such automation – reducing the cost of developing such systems.

For example, metadata can be used as a basis of discovery in dynamic systems, it can assist in managing a service, validating and auditing uses of services may also be simplified by rich metadata. It can help to ensure that requirements and expectations regarding the content of any data interchanged are properly interpreted and fulfilled.

2.2.3.2 The limits of description

There are well-known theoretic limits on the effectiveness of descriptions – it is simply not possible to completely specify in a completely unambiguous manner the precise semantics of a service. (This is Gödel's incompleteness result in another guise.)

Another way of stating the above is that there will always be unstated assumptions made by the describer of a service that must be implicitly shared by readers of the description. This applies to machine processable metadata as well as to human readable documentation.

Luckily, such precision is not normally necessary either – what is required is sufficient precision to enable required functionality.

Another kind of limit of descriptions is more straightforward: describing a service (for example) does not eliminate the requirement for making a choice. For example, a service directory might have the descriptions of many services – provided by many organizations. An automatic search of that directory is therefore likely to return multiple responses to any mechanical search criteria. At some point this set of responses has to be converted into a choice of a single service in order for a service consumer (say) to perform its function. In a multi-vendor scenario, that choice must also take into account real world aspects of the service – such as who the provider of the service is and whether the service consumer can or should trust the provider. It is unlikely that such factors can be easily and securely encoded in descriptions and search criteria.

2.3 Interacting with services

Interacting with a service involves exchanging information with the service and performing actions against the service. In many cases, this is accomplished by sending and receiving messages to and from the service end-point; but there are other modes possible that do not involve explicit message sending. However, for simplicity, we often refer to message exchange as the primary mode of interaction with a service. Together the forms of information exchanged and the kinds of actions performed form the **service interface** – see section 2.2.2.

There are three key concepts that are important in understanding what it is involved in interacting with services – the **information model**, the **process model** and the **execution context**.

2.3.1 Information model

The information model of a service is a characterization of the information that may be exchanged with the service.

The scope of the information model includes the format of documents and messages, the structural relationships within those documents and also the ontologies of terms used within those documents.

We do not, however, generally include within the information model of a service the information and data that might be stored or internally manipulation by a service. That is part of the service implementation.

2.3.1.1 Structure

Knowing the representation, structure and form of information required is a key initial step in ensuring effective interactions with a service. There are several levels of such structural information; ranging from the encoding of character data, through the use of formats such as XML, SOAP and schema-based representations.

2.3.1.2 Ontology

Particularly for messages, an important aspect of the service information model is the interpretation of strings and other tokens in the data. Loosely, one might partition the interpretation of a message into structure (syntax) and ontology (semantics); although both are part of the information model.

A described information model typically has a great deal to say about the form of messages, about the types of the various components of messages and so on. However, pure type information is not sufficient to completely describe the appropriate interpretation of data. For example, within an address structure, the city name and the street name are typically given the same type – some variant of the string type. However, city names and street names are not really the same type of thing at all. Distinguishing the correct interpretation of a city name string and a street name string is not possible using type-based techniques – it requires additional information that cannot be expressed purely in terms of the structure of data.

Ontologies are formal descriptions of sets of terms in terms of the relationships between them. Most commonly, the relationships are class relationships – one term represents a concept that is a sub-class of another. However, relationships are not limited to the sub-class relationships; other aspects of concepts can also be usefully represented; such as the range of possible values given property can take and whether the property is functional or not.

The role of explicit ontologies is to provide a firm basis for selecting correct interpretations for tokens in messages. For example, in the address example above, an ontology can be used to capture the appropriate distinction between street name and city name; so much so that in many cases it is possible to automatically map the contained information from one representation to another.

More specifically, for a service to be consistent, the service should make consistent use of terms as defined in one or more ontologies. Of course, specific domain semantics are beyond the scope of this reference model; however, the reference model does address the requirement that the service interface enable providers and consumers to unambiguously identify relevant definitions for their respective domains.

2.3.2 Behavior model

The second key requirement for successful interactions with services is knowledge of the behavioral or process aspects of the service. Loosely, this can be characterized as knowledge of the actions on, responses to and temporal dependencies between actions on the service.

For example, in a News subscription service, a successful use of the service involves initially registering a subscription with the service; which will then be followed by an irregular series of one-way news items. Key to using the service is the knowledge that you must first register your preferences and then you will get messages without further prompting.

Another example is a service that supports updating a balance with a transaction. Such services are typically *idempotent*: i.e., they will not change their state should a subsequent interaction be attempted for the same transaction. The behavioral model of the account update service then consists of an initial communication – incorporating the transaction to log – followed by a response which includes the new balance.

2.3.2.1 Process Model

It is fairly common to partition the process model associated with a service into two levels: the particular sequences of operations needed to achieve single service exchanges and longer term transactions. These two levels may be nested – a long running transaction is often composed of sequences of exchange patterns.

For example, in a publish-and-subscribe service, there are individual operations dealing with registering a new subscription (say) and publishing a new notice (say). The longer view of a given service considers the total sequence of notifications associated with a given subscription. Another concept that may be featured in a process model is the transactional structure of a service (c.f. ACID analysis of processes).

Note that although the existence of a process model is fundamental to this Reference Model, its extent is not defined. In some architectures the process model will include aspects that are not strictly part of this reference model – for example we do not address the orchestration of multiple services – although orchestration and choreography may be part of the process model of a given architecture. At a minimum, the process model must cover the interactions with the service itself.

Choosing an appropriate representation of process models is a fine art; a representation system that can express sequences and dependencies is often *Turing complete*[2] – i.e., is effectively a programming language. The problem with Turing complete representations of processes is that processing such descriptions quickly becomes intractable for non-trivial process models. For example, the task of comparing two processes is a difficult exercise that is provably impossible in the general case. On the other hand, without some such expressive power it can be difficult to capture the required dependencies that are a natural part of process descriptions.

However, showing that two process models are equivalent is not the only requirement for representing process models. A more common requirement is simply to be able to identify the appropriate steps that must be followed for a successful interaction. This is analogous to following a recipe or executing a program – a task that is easily mechanizable.

2.3.2.2 Behavior

The **behavioral** model of a service is about the behavior that results in interactions with the service. Of course, a great portion of the behavior of a service may be private; however, the expected public view of a service surely includes the implied behavior of the service.

For example, in a service that represents a bank account, it is not sufficient to know that to use the service you need to exchange a given message (with appropriate authentication tokens). It is also of the essence that using the service may actually affect the bank account – withdrawing cash from it for example.

The behavior of a service is closely connected to its intended real-world effect; although not identical to it. In general, we can state that the behavior of a service (an attempt to withdraw cash from an account) results in an intended (or occasionally unintended) effect in the world: the account's balance is lower.

2.3.3 Services in context

In an implementation, services are associated with an **execution context**. Or, another way of expressing this is to consider that there is a distinction between a potential service and an actual service that is capable of being interacted with. An actualized service has an execution context that determines many of the properties of the service; including attributes such as security.

For example, suppose that it were important that a given service was always executed in an authenticated context – i.e., that the service provider and the service consumer have authenticated themselves to each other. The details of how authentication is performed are not our concern here. That authentication context is an example of a particular execution context that applies to the service.