Reference Model for Service Oriented Architectures

Working Draft 10ED, 1 October 2005

Document identifier:
wd-soa-rm-10ED

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 Reference Model for Service Oriented Architectures is an abstract framework for understanding significant entities and relationships between 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 service oriented architectures or in training and explaining SOA. A reference model is not directly tied to any standards, technologies or other concrete implementation details. 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 focuses on the field of software architecture. While the concepts and relationships described may apply to other "service" environments, this specification makes no attempt to completely account for use outside of the software domain.

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.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the SOA-RM TC web page at:

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

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

The notion of Service Oriented Architecture (SOA) has received significant attention within the software design and development community. The result of this attention is the proliferation of many conflicting definitions of SOA. Whereas SOA architectural patterns (or *reference architectures*) may be developed to explain and underpin a 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 *all* SOA.

1.1 What is a reference model

A reference model is an abstract framework for understanding significant relationships among the entities of some environment that enables the development of specific architectures using consistent standards or specifications supporting that environment. A reference model consists of a minimal set of unifying concepts, axioms and relationships within a particular problem domain, and is independent of specific standards, technologies, implementations, or other concrete details.

The purpose of a reference model is to provide a common conceptual framework that can be used consistently across and between different implementations and is of particular use in modeling specific solutions.

The goal of this reference model is to define the essence of service oriented architecture, and emerge with a vocabulary and a common understanding of SOA. It provides a normative reference that remains relevant for SOA as an abstract and powerful model, irrespective of the various and inevitable technology evolutions that will impact SOA.

1.2 Audience

The intended audiences of this document include non-exhaustively:

- Architects and developers designing, identifying or developing a system based on the service-oriented paradigm.
- Standards architects and analysts developing specifications that rely on service oriented architecture concepts.
- Decision makers seeking a "consistent and common" understanding of service oriented architecture.
- Users who need a better understanding of the concepts and benefits 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.

Section one 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 and usage of reference models.

Section two introduces the concept of SOA and identifies some of the ways that it differs from previous paradigms for distributed systems. Section two offers guidance on the basic principles of service oriented architecture. This can be used by non-technical readers to gain an explicit understanding of the core principles of SOA and by architects as guidance for developing specific service oriented architectures.

Section three introduces the Reference Model for SOA. First, the main axioms, key concepts and relationships between those concepts are introduced followed by more detailed sections on the main concepts, namely *service* is defined along with *service description*. There then follows a section detailing interaction between services, followed by service policies and expectations. Finally, the concept of service discoverability is introduced.

Section four addresses compliance 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";
- Makes (publicly) an adoption statement to use the Reference Model for SOA of this TC as a base or inspiration for their work; and
- Standards or technologies that claim to be service oriented.

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

2 Service Oriented Architecture

2.1 What is SOA?

Service Oriented Architecture (SOA) is a paradigm for organizing and using distributed capabilities that may be under the control of different ownership domains. It is natural in such a context to think of one person's needs being met by capabilities offered by someone else; or, in the world of distributed computing, one computer agent's requirements being met by a computer agent belonging to a different owner. There is not necessarily a one-to-one correlation between needs and capabilities; the granularity of needs and capabilities vary from fundamental to complex, and any given need may require the combining of numerous capabilities while any single capability may address more than one need. The perceived value of SOA is that it provides a powerful framework for matching needs and capabilities and for combining capabilities to address those needs.

Visibility, interaction, and effect are key concepts for describing the SOA paradigm. **Visibility** refers to the capacity for those with needs and those with capabilities to be able to see each other. This is typically done through generating metadata to describe such aspects as functions and technical requirements, related constraints and policies, and mechanisms for access or response. The metadata must be in a form (or can be transformed to a form) in which its syntax and semantics are widely accessible and understandable.

Whereas visibility introduces the possibilities for matching needs to capabilities (and vice versa), **interaction** is the activity of sorting through the possibilities to determine which match or combination of matches provides an adequate and feasible response to the needs and then activating that response. Typically mediated by the exchange of messages, the interaction proceeds through a series of information exchanges and invoked actions. There are many facets of interaction; but they are all grounded in a particular **execution context** – the set of technical and business elements that eventually form a path between those with needs and those with capabilities and that permit information to be exchanged, actions to be performed and provides a decision point for any policies and contracts that may be in force.

The purpose of using a capability is to realize one or more **real world effects**. At its core, an interaction is "an act" as opposed to "an object" and the result of an interaction is an effect (or a set/series of effects). We are careful to distinguish *public* actions and *private* actions; private actions are inherently unknowable by other parties. On the other hand, public actions result in changes to the *state* that is shared (at least) between those involved in the current execution context. Real world effects are, then, couched in terms of changes to this shared state.

The expected effects should be made visible as part of the capability metadata and form an important part of the decision on whether a given capability matches similarly described needs. At the interaction stage, the description of real world effects establishes the expectations of those using the capability. Note, it is not possible to describe every effect from using a capability and, in fact, a cornerstone of SOA is that one using a capability does not need to know all the details.

To this point, this description of SOA has yet to mention what is usually considered the central concept: the service. It is likely that both needs and capabilities exist outside of SOA but a major driver for SOA is the perceived improvement in bringing needs and capabilities together. **In SOA, services are the mechanism by which needs and capabilities are brought together.** SOA is not the solution of domain problems but rather a way of organizing a wider array of possibilities to generate a domain solution. By itself, SOA does not provide a solution to a difficult domain problem where a satisfactory solution does not already exist. SOA can, however, provide an organizing and delivery paradigm that enables one to get more value from use of both solutions which are locally "owned" and solutions under the control of others. It also enables one to express solutions in a way that makes it easier to modify or evolve the identified solution or to try

alternate domain solutions. SOA does not provide any domain elements of a solution that do not exist without SOA.

The concepts of visibility, interaction, and effect apply directly to services in the same manner as these were described for the general SOA paradigm. Visibility is established through the **service description** which contains the information necessary to interact with the service and describes this in such terms as the service inputs, outputs, and associated semantics. The service description also conveys what is accomplished when the service is invoked and the conditions for invoking the service. In general, entities (people and organizations) offer capabilities through services and act as **service providers**. Those with needs who make use of capabilities through their associated services are referred to as **service consumers**. The service description allows prospective consumers to decide if the service is suitable for their current needs and establish whether a consumer satisfies any requirements of the service provider to be permitted access.

Having described what is SOA, it is appropriate to note several things which are related but are not necessary attributes or restrictions.

- SOA identifies necessary aspects of interactions involving multiple ownership domains; however, it does not directly embody concepts relating to ownership.
- SOA is commonly implemented using Web services, but services can be made visible, support interaction, and generate effects through other implementations.

In most discussions of SOA, the terms “loose coupling” and “coarse-grained” are commonly applied as SOA concepts, but these terms have intentionally not been used in the current discussion because they are subjective and without useful metrics to indicate when the ideals have been accomplished. In terms of needs and capabilities, SOA will be most effective when it is bringing to bear a solution and not the “fine-grained” pieces where those pieces are unlikely to be reusable outside that particular solution. That said, granularity and coarseness are usually relative to detail for the level of the problem being addressed, e.g. one that is more strategic vs. one down to the algorithm level, and defining the optimum level is not amenable to counting the number of interfaces or the number or types of information exchanges connected to an interface.

The value of SOA is that it provides a simple scalable paradigm for organizing large networks of systems that require interoperability to realize the value inherent in the individual components. Indeed, SOA is scalable because it makes the fewest possible assumptions, including about the network and also minimizes any trust assumptions that are often implicitly made in smaller scale systems. Loose coupling and coarse-grained as concepts do not address these more fundamental aspects that are critical to SOA success.

3 The Reference Model

A service oriented architecture represents a uniform means to offer, discover and interact with capabilities to produce desired effects consistent with measurable preconditions and expectations. This section introduces the main concepts and a detailed discussion of the concepts and their relationships are in the sections that follow.

3.1 Overview of model

A key concept of SOA is that of **service**. In general, entities (people and organizations) create capabilities to solve or support a solution for the problems they face in the course of their business. SOA is a way to organize the world around this key concept of service. The noun "service" is defined in dictionaries as "The performance of work (a function) by one for another." However, service, as a concept, is a unifier for the following related ideas:

- The capability to perform work for another
- The specification of the work offered for another
- The offer to perform work for another

Therefore, this imprecise term is partitioned into separate, yet interrelated, precise concepts. These concepts are an offer, interaction and effect.

The concept of an offer follows directly from the dictionary definition of service: 'by one' and 'for another.' In general terms, an offer is a proposal. These offers are made by providers which may possess a capability that address a need. The convergence of a capability and a need results in an interaction. At its core, an interaction is "an act" as opposed to "an object." Therefore, this concept focuses the necessary interfaces and behavior to support the interaction. The final concept focuses on the consequences of interaction.

The concepts above emphasize a distinction between a capability and the ability to bring that capability to bear in the context of SOA. It is assumed that the capability was created and exists independently of SOA.

A key concept of SOA is that of a **service**. In general, people and organizations create capabilities to solve or support the solution for 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 repurposed or new capabilities can be readily substituted to improve the solutions. A service is a means to access such capabilities.

In order to use a service, it is necessary to know that it exists, what is accomplished if the service is invoked, how the service is invoked, and other characteristics. Collectively this is the service **visibility**. When given an explicit searchable form, this information allows, for example, prospective consumers to decide if the service is suitable for their current needs and establish whether a consumer satisfies any requirements of the service provider to be permitted access. This information constitutes the **service description**.

It is often the case that the consumer of a service and the provider of a service belong in different ownership domains. As a result, there is a natural distinction to be drawn between the public interactions with a service and the private actions of both the service provider and consumer. This distinction maintains and encourages independence of each service participant which, in turn, greatly enhances the scalability and security attributes of SOA. Focus can be directed to 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. Service conditions are loosely associated 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 achieved 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 SOA. 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. Registries or repositories of service descriptions are undoubtedly powerful means of achieving this, but SOA is not limited to these.

SOA identifies necessary aspects of interactions involving multiple ownership domains; however, it does not directly embody concepts relating to ownership.

3.2 The Reference Model

3.2.1 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 typically hidden from the service consumer except for (1) the data 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 invoking a service is an expectation of one or more real world effects. The effects may include:

1. information returned in response to a request,
2. processing done in response to a request to change the state of defined 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 required by the service and the service would return information, status indicators, or error descriptions, where both the input and output are as described by the data 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 service concept above 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 that the capability was created and exists outside of the SOA. One of the major benefits of SOA is enabling the capability to be applicable 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 in terms of a clear expression of the nature of SOA and the value it provides.

3.2.2 Service description

The service description represents the information needed in order to use a service. It may be considered part of or the complete set of the metadata (see Section 3.2.3) associated with a service. 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 data model, many elements such as function and policy may vary.

Best practice suggests that the service description should be represented using a standard, reference-able format. Such a format facilitates the use of common processing tools (such as discovery engines) that can capitalize on 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 that a consumer needs in order to decide whether or not to use a service. In particular, a service consumer must possess the following items of information:

1. That the service exists and is reachable;
2. That the service performs a certain function or set of functions;
3. That the service operates under a specified set of constraints and policies;
4. That the service will (to some implicit or explicit 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 the relationship of these information items to the service description.

3.2.2.1 Service Reachability

A service description should include sufficient data to permit a service consumer and service provider to exchange information. This might include metadata (such as the location of the service and what information protocols it supports and requires) and whether the service is currently reachable or not.

3.2.2.2 Service Functionality

Item 2 relates to the need to unambiguously express the function(s) of the service and the real world effects (see section 3.5) 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 Section 3.2.2.3 for how this relates to service description and Section 3.5 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 as exposed 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.

3.2.2.3 Policies Related to a Service

Items 3 and 4 from Section 2.2.2 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 3.3.3 refers to as the *execution context*.

3.2.2.4 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 should be 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 consumer. This logical expression of the set of information items associated with the consumption of the service is often referred to as the service's data model. It should be noted that the particulars of the standard reference-able format is beyond the scope of the reference model. However, requiring that mechanisms be available (in order to define and retrieve such definitions) is fundamental to the SOA concept.

While this discussion refers to a standard reference-able syntax for service descriptions, it is not specified 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 by its consumers.

3.2.2.5 An Example of Using Information Contained in the Service Description

The following example may help clarify the concepts related to service and service description.

A utility has the capacity to generate and distribute electricity (the underlying capability). A consumer accesses electricity generated (the service) via a wall outlet (service interface). In order to use the electricity (what is provided by invoking the service), a consumer needs to understand what type of plug to use; the utility presumes that the customer will only connect devices that are compatible with the voltage provided; and the consumer in turn assumes that compatible devices can be connected without damage or harm (service assumptions). A residential or business user will need to open an account with the utility in order to use the supply (service constraint) and the utility will meter usage and expects the consumer to pay for use at the rate prescribed (service policy). When the consumer and utility agree on constraints and polices (service contract), the consumer can receive electricity using the service. Another person (say, a visitor to someone else's house) may use a contracted supply without any relationship with the utility or any requirement to also satisfy the initial service constraint but would nonetheless be expected to be compatible with the service interface. In certain situations (for example, excessive demand), a utility may limit supply or institute rolling blackouts (service policy). A consumer might lodge a formal complaint if this occurred frequently (consumer's implied policy). In this example, the underlying capability would still exist and be usable even if every device were required to be hard-wired to the utility's equipment, but this would result in a very different service and service interface.

3.2.3 Descriptions and Metadata

One of the hallmarks of a Service Oriented Architecture is the degree of documentation and description associated with it; particularly *machine process-able 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. Standardizing the formats of such metadata reduces the cost and burden of producing the descriptions necessary to promote reuse and integration.

3.2.3.1 The roles of description

An important additional benefit of metadata – as opposed to informal natural language descriptions – is its potential to facilitate automated software development. 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. Metadata can assist in managing a service, validating and auditing usage of services which may also be simplified by rich metadata. It can also help ensure that requirements and expectations (regarding the content of any data interchanged) are properly interpreted and fulfilled.

3.2.3.2 The limits of description

There are well-known theoretic limits on the effectiveness of descriptions – it is simply not possible to specify, completely and unambiguously the precise semantics of a service.

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 process-able metadata as well as to human readable documentation.

Luckily, complete precision is not 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 see the required function performed. In a multi-vendor scenario, that choice must also take into account real world aspects of the service – such as whether the service consumer can identify the provider, can or should trust the provider, and whether the provider is reliable and timely in delivering the service offered. It is unlikely that such factors can be easily and securely encoded in descriptions and search criteria.

3.3 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, but there are other modes possible that do not involve explicit message transmission. However, for simplicity, we often refer to message exchange as the primary mode of interaction with a service. Together the types of information exchanged and mechanisms used constitute the **service interface** – see Section 3.2.2.4.

The key concepts that are important in understanding what it is involved in interacting with services are the **data model**, the **process model,** the **execution context** and the **expectations** about the interaction.

3.3.1 Data model

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

The scope of the data model includes the format of documents and messages, the structural relationships within those documents and also the definition of terms used within those documents. Only information and data that are potentially exchanged with a service are generally included within that service's data model.

Loosely, one might partition the interpretation of an informational block into structure (syntax) and meaning (semantics); although both are part of the data model. Particularly for information that is exchanged across an ownership boundary, an important aspect of the service data model is the interpretation of strings and other tokens in the information.

3.3.1.1 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.

A described data 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 “typed” information is not sufficient to completely describe the appropriate interpretation of data.

3.3.1.2 Ontology

The primary task of any communication infrastructure is to facilitate the exchange of information and the exchange of intent. For example, a purchase order combines two somewhat orthogonal aspects: the description of the items being purchased and the fact that one party intends to purchase those items from another party. Even in purely internal exchanges within an Enterprise Application suite, exchanges have similar aspects: this is an update to the customer profile with these changes.

With information being exchanged potentially across ownership boundaries a critical issue is the interpretation of the data. This interpretation must be consistent between the participants in the service interaction. Consistency in interpretation of information is a stronger requirement than

merely type (or structure) consistency – the tokens in the data itself must also have a shared basis.

For example, within a street address structure, there is a huge potential for variability in representing addresses. For example, an address in San Francisco, California may have variations in the way the city is represented: SF, San Francisco, San Fran, the City by the Bay are all alternate denotations of the same city. For successful exchange of address information, all the participants must have a consistent view of the meaning of the address tokens. How a city name is interpreted is a simple but classic problem in ensuring that addresses can be reliably shared.

An ontology is a formal description of terms and the relationships between them in a given context. 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, an ontology can be used to capture the alternate ways of expressing the names of cities as well as distinguishing a city name from a street name.

Ontologies also provide a point of context to facilitate the *reinterpretation* of data – for example that a 3/8" steel washer may be a potential replacement for a 1cm spacer. Such a reinterpretation is effectively represented as a particular traversal of the graph of concepts and relationships embodied in the ontology. How much automation of ontology walking is appropriate will depend on the nature of the service and the service participants.

Note that, for the most part, it is not expected that service consumers and providers would actually exchange ontologies in their interaction – the role of the ontology is a background one – it facilitates sound interactions. Hence ontology references are mostly to be found in **service descriptions**.

More specifically, and in order for a service to be consistent, the service should make consistent use of terms as defined in an ontology. Specific domain semantics are beyond the scope of this reference model; but there is a requirement that the service interface enable providers and consumers to identify unambiguously those definitions that are relevant to their respective domains.

3.3.2 Behavioral model

The second key requirement for successful interactions with services is knowledge of the actions invoked against the service and the process or temporal aspects of interacting with 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 security-controlled access to a database, the actions available to a service consumer include presenting credentials, requesting database updates and reading results of queries. The security may be based on a challenge-response protocol. For example, the initiator presents an initial token of identity, the responder presents a challenge and the initiator responds to the challenge in a way that satisfies the database. Only after the user's credentials have been verified will the actions that relate to database update and query be accepted.

The sequences of actions involved are a critical aspect of the knowledge required for successful use of the secured database.

There are other aspects of the behavior of services that are important. These include, for example, whether the service is transactional, idempotent or long running. As a particular example, a service that supports updating an account balance with a transaction is typically idempotent; i.e., the state of the account would not be affected should a subsequent interaction be attempted for the same transaction.

3.3.2.1 Action model

The **action** model of a service is about the individual actions that may be invoked against the service. Of course, a great portion of the behavior resulting from an action may be private; however, the expected public view of a service surely includes the implied effects of actions.

For example, in a service that manages a bank account's state, 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.

3.3.2.2 Process Model

The **process model** characterizes the temporal relationships between actions and other higher-order attributes of interacting with the service. 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.

Note that although the existence of a process model is necessary 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.

3.3.2.3 Higher-order attributes of processes

Beyond the straightforward mechanics of interacting with a service there are other, higher-order, attributes of services' process models that are also often important. These can include whether the service is **idempotent**, whether the service is **long-running** in nature and whether it is important to account for any **transactional** aspects of the service.

A service is idempotent if subsequent attempts to perform identical transactions are discounted. For example, it often important that a bank will only cash a cheque once – subsequent attempts to cash the same cheque should be ignored or rejected. Note that idempotency is not the same as effect-free or stateless: a service that always returns the same results is idempotent, but only by virtue of the fact that it does not change from invocation to invocation.

Idempotency is an important attribute of a service in an environment where there is a significant possibility that the interaction between the service provider and consumer may be interrupted – whether by a network issue or simply one of the parties dropping out.

A service is long-running if the activities engendered by an interaction are likely to persist beyond the immediate interaction itself. For example, a classic book selling service might be viewed as a long-running service: the activity started by the purchase of the book may take days or weeks to complete. It can be important to account for a long-running process as it has implications for the kinds of infrastructure needed – both by the service provider and by the service consumer – in order to be able to keep track of the progress of the interaction.

Often, once a business-level contract has been agreed on, it can be difficult or impossible to simply cancel the consequences of the agreement. This is particularly an issue when the agreement of several parties is necessary simultaneously. For example, booking a vacation may require a flight ticket as well as a hotel room – without either the vacation is a bust. However, the airline typically will not have a relationship with the hotel; so if it turns out that there are no hotel rooms available for the vacation the airline ticket will need to be canceled.

The process of reversing a previously completed transaction – backing out of the airline booking for example – is likely to be quite different to the process for the original transaction. Knowledge such compensatory actions is a key aspect of interacting with transactional services.

3.3.3 Actualized Services

The **execution context** of a service interaction is the set of infrastructure elements, process entities and policy assertions that are deployed as part of the instantiated service interaction. In effect, the execution context defines the point of contact between abstractions such as service descriptions which are mostly about the potential for interaction and an actually executing service.

The most basic aspect of an execution context must be consistency between what is actually happening and any descriptions of what ought to be happening. The execution context is the point of measurement between the service description and reality.

The execution context is not limited to one side of the interaction; rather it refers to the totality of the interaction – including the service provider, the service consumer and the common infrastructure needed to mediate the interaction. Thus it represents a path between the intentions of the service consumer and those of the service provider.

The execution context is a critical touchstone for many aspects of a service interaction – for example, it defines the decision point for any policy enforcement relating to the service interaction. Note that a policy decision point is not necessarily the same as an enforcement point: an execution context is not by itself something that lends itself to enforcement. On the other hand, any enforcement mechanism of a policy is likely to take into account the particulars of the actual service interaction.

The execution context also allows us to distinguish services from one another. Different instances of the same service – denoting interactions between a given service provider and different service consumers for example – are distinguished by virtue of the fact that their execution contexts are different.

Finally, the execution context is also the context in which the interpretation of data that is exchanged takes place – it is where the *symbol grounding* happens as it were. A particular string has a particular meaning in a service interaction in a particular context – the execution context.

3.4 Real World Effect

There is always a particular purpose associated with interacting with a service; conversely, a service provider (and consumer) often has a priori conditions that apply to its interactions. The service consumer is trying to achieve some result by interacting with the service, as is the service provider. At first sight, such a goal can often be expressed as “trying to get the service to do something”. This is sometimes known as the **real world effect** of using a service. For example, an airline reservation service can be used in order to book travel – the desired real world effect being, presumably, a seat on the right airplane.

The internal actions that a service providers and consumers perform as a result of participation in service interactions are, by definition, private and fundamentally unknowable.[1] By unknowable we mean both that external parties cannot see others’ private actions and, furthermore, should not have explicit knowledge of them. Instead we focus on the state that is shared between the parties

[1] A similar analysis applies to service consumers: just how a consumer of a service decides which requests to make and which actions to performs is something that the service provider cannot determine.

– the **shared state**. Actions by service providers and consumers lead to modifications of this shared state; and that in turn leads to modified **expectations** by the participants.

When an airline has confirmed a seat for a passenger on a flight this represents a fact that both the airline and the passenger share – it is part of their shared state. Thus the real world effect of booking the flight is a modification of this shared state – the existence of the booking. Flowing from the shared facts, both the passenger, the airline and interested third parties may make inferences – in particular that when the passenger arrives at the airport the airline confirms the booking and permits the passenger onto the airplane (subject of course to the passenger meeting the other requirements for traveling).

Of course, in order for the airline to know that the seat is confirmed it will likely invoke some kind of private action to record the reservation; but, by minimizing assumptions about how the airline fulfils its contracts, potential for smooth interoperation is maximized. Such minimization principles represent a key success factor for scalability.

Although there is not necessarily a one-to-one correspondence, the natural container for the conditions applying to a service is the **service policy**. Similarly, the natural container for the expectations arising from a service is the **service contract**.

3.5 Policies and Contracts

Broadly speaking, a **policy** represents some form of constraint or condition on the use, deployment or description of an owned entity. A **contract**, however, represents an agreement of some kind; often the agreement is also about the conditions of use of a service and what the expected real world effects might be. The reference model is focused primarily on the concept of policies and contracts as they apply to services.

3.5.1 Service Policy

In abstract, a policy is a statement of the obligations, constraints or other conditions of use of a given service that expresses intent on the part of a participant. More particularly, policies are a way for expressing the relationship between the **execution context** and the **data** and **behavior models** associated with the service.

Conceptually, there are three aspects of policies: the policy assertion, the policy owner (sometimes referred to as the policy subject) and policy enforcement.

For example, the assertion: “All messages are triple-DES encrypted” is an assertion constraining the forms of messages. As an assertion, it is measurable: it may be true or false depending on whether the traffic is actually encrypted or not. Note that policy assertions are often about the way the service is realized; i.e., they are about the relationship between the service and its execution context.

A policy always represents a participant’s point of view. An assertion becomes the policy of a participant when they make it their policy – this linking is normally not part of the assertion itself. For example, if the service consumer declares that “All messages are triple-DES encrypted”, then that reflects the policy of the service consumer. This policy is one that may be asserted by the service consumer independently of any agreement from the service provider.

Finally, a policy may be enforced. Techniques for the enforcement of policies depend on the nature of the policy. From a conceptual point of view, service policy enforcement amounts to ensuring that the policy assertion is consistent with the real world. This might mean preventing unauthorized actions to be performed or states to be entered into; it can also mean initiating compensatory actions when a policy violation has been detected. An unenforceable policy is not a policy; it would be better described as a wish.

Policies potentially apply to many aspects of SOA: security, privacy, manageability, Quality of Service and so on. Beyond such infrastructure-oriented policies, participants may also express business-oriented policies – such as hours of business, return policies and so on.

Policy assertions may be written down in a formal machine processable form. The importance of such a machine processable form of policy depends on the purpose and applicability of the policy. In particular, where a policy declaration might affect whether a particular service is used or not, then such policies should be expressed in machine-processable form.

Languages that permit policy assertions also range in expressivity from simple propositional assertions to modal logic rules. However, the Reference Model is neutral to how a policy is represented.

A natural point of contact between service participants and policies associated with the service is in the service description – see Section 3.2.2. It would be natural for the service description to contain references to the policies associated with the service.

3.5.1.1 Service Contract

A variant of the policy concept is the agreement or **contract**. A contract has all the same features as a policy with one key addition: the concept of agreement, contracts are policies that have been agreed to by participants governed by the policy; policies do not need agreement only enforcement.

Where a policy is associated with the point of view of individual participants, a contract represents an agreement between two or more participants. Like policies, contracts can cover a wide range of aspects of services: quality of service agreements, interface and choreography agreements and commercial agreements.

Thus, following the analysis above, a service contract is a measurable assertion that governs the requirements and expectations of two or more parties. Unlike policy enforcement, which is usually the responsibility of the policy owner, contract enforcement may involve resolving disputes between the parties to the contract. The resolution of such disputes may involve appeals to higher authorities.

Like policies, contracts may be expressed in a machine processable form. Where a contract is used to codify the results of a service interaction, it is good practice to represent it in a machine processable form. This facilitates automatic service composition, for example. Where a contract is used to describe over-arching agreements between service providers and consumers, then the priority is likely to make such contracts readable by people.

3.6 Service Discoverability

A key concept of the SOA Reference Model is the discoverability of services. Discoverability is an important aspect of bringing together the service provider and consumer: A service provider must be capable of making details of the service (notably service description and policies) available to potential consumers; and customers must be capable of finding that information.

This may (and commonly does) involve a service provider entering the service description into a service registry and the service consumer searching for an appropriate match to their needs. The SOA concept of discoverability is not restricted to any single mechanism. In some architectures there may not be a registry .

Service Discoverability requires that the service description and policy – or at least a suitable subset thereof – be available in such a manner and form that, directly or indirectly, an awareness of the existence and capabilities of the service can become known to potential consumers. The extent to which the discovery is “pushed” by the service provider, “pulled” by a potential consumer, subject to a probe or another method, will depend on many factors.

For example, a service provider may advertise and promote their service by either including it in a service directory or broadcasting it to all consumers; potential consumers may broadcast their particular service needs in the hope that a suitable service responds with a proposal or offer or a service consumer might also “probe()” an entire network to determine if suitable services exist. When the demand for a service is higher than the supply, then by advertising their needs,

potential consumers are likely to be more effective then service providers advertising offered services.

One way or another, the potential consumer must acquire a sufficient description to evaluate whether the service matches their expectations and, if so, the method for the consumer to establish a contract and invoke the service.

Specific SOA reference architectures and implementations will prescribe the mechanisms for actual service discovery, ensuring a service's presence and availability, and failure conditions and error handling.

4 Conformance Guidelines

The authors of this reference model envision that architects may wish to declare their architecture is conformant with this reference model. Conforming to a Reference Model is not generally an easily automatable task – given that the Reference Model's role is primarily to define concepts that are important to SOA rather than to give guidelines for implementing systems.

However, we do expect that any given Service Oriented Architecture will reference the concepts outlined in this specification. As such, we expect that any design for a system that adopts the SOA approach will

- Have entities that can be identified as services as defined by this Reference Model,
- Such entities will have descriptions associated with them,
- Service entities will have identifiable interaction models, including models of the information exchanged by the services and the temporal behavior of the services
- It should be possible to identify a means by which consumers of services and providers of services are able to engage; and
- That there will be identifiable aspects of service entities that correspond to the policies relating to the conditions of use of services and to the expectations that result from interacting with services.

It is not appropriate for this specification to identify *best practices* with respect to building SOA-based systems. However, the ease with which the above elements can be identified within a given SOA-based system could have significant impact on the scalability, maintainability and ease of use of the system.

5 References

5.1 Normative

[RFC2119] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, http://www.ietf.org/rfc/rfc2119.txt, IETF RFC 2119, March 1997.

5.2 Non-Normative

[W3C WSA] W3C Working Group Note "Web Services Architecture", http://www.w3.org/TR/ws-arch/ , 11 February 2004

Appendix A. Glossary

Terms that are used within this Reference Model are often also found in other specifications. In order to avoid potential ambiguity, this glossary locally scopes the definitions of those terms for the purpose of this Reference Model and thus overrides any other definitions.

Advertising (or Announcement of Availability)

A means of conveying the existence of and sharing awareness about a service to potential consumers.

Agent (requester or provider)

An entity acting on behalf and with the authority of another entity and charged to fulfill a task.

Architecture

A set of artifacts (that is: principles, guidelines, policies, models, standards and processes) and the relationships between these artifacts, that guide the selection, creation, and implementation of solutions aligned with business goals.

Software architecture is the structure or structures of an information system consisting of entities and their externally visible properties, and the relationships among them.

Authentication

The act by which an agent establishes – to an agreed level of confidence – the identity of another entity.

Capability

Response to a need that is characterized by a set of preconditions and where the solution is consistent with the expectations of real world effects that would result from making use of the capability.

(Service) Consumer

An entity which intends to make use of a service.

Contract

The syntactic, semantic and logical constraints governing the use of a service.

Data Model

A Data Model is the abstract paradigm used in the invocation and consumption of a service. It is expressed as a set of information items associated with the use of a service.

Discovery

The act of detecting and gaining understanding of the nature of a service.

Encapsulation

The act of hiding internal specifications of an entity from the user of that entity, in such a way that the internal data and methods of the entity can be changed without changing the manner in which the entity is used. What is seen by the user is only an interface, or service.

Framework

A set of assumptions, concepts, values, and practices that constitutes a way of viewing the current environment.

Interface

A named set of operations that characterize the behavior of an entity.

Mediation

The transformation, routing, validation and processing of messages.

Message

A serialized set of data that is used to convey a request or response from one party to another.

Metadata

A set of properties of a given entity which are intended to describe and/or indicate the nature and purpose of the entity and/or its relationship with others.

Negotiation

A process that seeks to establish an acceptable basis for a contract between agents for the provision of a service.

Ontology

Represents an agreement within a specific environment of the meanings to be associated with different concepts and their relations to each other.

Opaqueness

The extent to which an agent is able to interact successfully with a service without detecting how the service is implemented.

Policy

A statement of obligations, constraints or other conditions of use of a given service. When a specific set of entities accept such a policy, a contract is usually established.

Reference Model

A reference model is an abstract framework for understanding significant relationships among the entities of some environment that enables the development of specific architectures using consistent standards or specifications supporting that environment.

A reference model is based on a small number of unifying concepts. 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.

(Service) Requester or provider

An agent that interacts with a service in order to achieve a goal

Security

A set of policies and measures designed to ensure that agents in an environment can only perform actions that have been allowed. Security in a specific environment is an agreed compromise between meeting the needs of agents and maintaining the integrity of the environment.

Semantics

A conceptualization of the implied meaning of information, shared between the service consumer and the service provider, that requires words and/or symbols within a usage context.

Service

A behavior or set of behaviors offered by one entity for use by another according to a policy and in line with a service description.

Service description

A set of information describing a service, sufficient to allow a potential consumer to ascertain, where appropriate:

- the identity of (and/or information about) the service provider;

- the policies, parameters and terms of use of the service;

- the procedures and constraints governing invocation of the service,

and thus determine whether the service meets the expectations and requirements of the consumer. Acceptance of the service description by a consumer does not of itself imply a contract to use the service.

Service Oriented Architecture (SOA)

A software architecture of services, policies, practices and frameworks in which components can be reused and repurposed rapidly in order to achieve shared and new functionality. This enables rapid and economical implementation in response to new requirements thus ensuring that services respond to perceived user needs.

SOA uses the object-oriented principle of encapsulation in which entities are accessible only through interfaces and where those entities are connected by well-defined interface agreements or contracts.

Appendix B. Acknowledgments

The following individuals were members of the committee during the development of this specification:

[TODO: insert cte. Members]

Appendix C. Notices

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification, can be obtained from the OASIS Executive Director.

OASIS invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights, which may cover technology that may be required to implement this specification. Please address the information to the OASIS Executive Director.