Service Oriented Architecture Reference Model

Working Draft 07, 12 May 2005

Document identifier:

wd-soa-rm-07

Location:

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

Editors:

C. Matthew MacKenzie, Adobe Systems Incorporated, mattm@adobe.com
John Harby, Individual, jharby@gmail.com
Michael Ruiz, BAE Systems PLC, Michael.ruiz@baesystems.com
Christopher Bashioum, Mitre Corporation, cbashioum@mitre.org
Ken Laskey, Mitre Corporation, klaskey@mitre.org
Wesley McGregor, Government of Canada (PWGSC), McGregor.Wesley@tbs-sct.gc.ca
Francis McCabe, Fujitsu Limited, fgm@fla.fujitsu.com
Don Flinn, Individual, flinn@alum.mit.edu
Peter Brown, Individual, peter@justbrown.net
Vikas Deolaliker, Sonoa Systems, Inc., vikas@sonoasystems.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 system 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.

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 service-oriented architecture (SOA) paradigm has received significant attention within the software design and development industry in recent times resulting in many conflicting definitions of service-oriented architecture. 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.

This document explicitly avoids defining implementation detail, as doing so would unnecessarily constrain and date the reference model. The goal is to provide a document that can stay relevant through the various technology evolutions that we experience in this industry.

A reference model cannot be implemented, nor should it be. A reference model is a foundational work that can and should be used to develop architectural patterns and promote effective discourse on derived works.

1.1 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 relates to or makes use of the service-oriented paradigm.
- Chief Information Officers and other decision makers seeking a "consistent and common" understanding of service oriented architecture.

1.2 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 service oriented reference model. First, services are defined and service composition and description are described. A brief overview of the policy components and their

relationships is given. 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 aims to provide guidelines for conformance with the reference model and is aimed at those who wish to explicitly state that their architectures are conformant with this reference model.

Section 4 provides references to external material used in the reference model.

The appendices provide several non-normative examples and a glossary to provide clarity of terms whose use may otherwise be ambiguous.

1.3 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].

1.4 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.

165

2 The Reference Model

Figure 2-1 - SOA Architectural Model introduces the core Service Oriented Architecture (SOA) reference model and its high level components.

A service, the fundamental element of a SOA, is decomposed into four distinct aspects and two cross cutting concerns. The four distinct aspects include:

- Descriptor
- Policy
- Contract
- Data Model

The “cross cutting” concerns are:

- Semantics
- Discovery, Presence and Availability

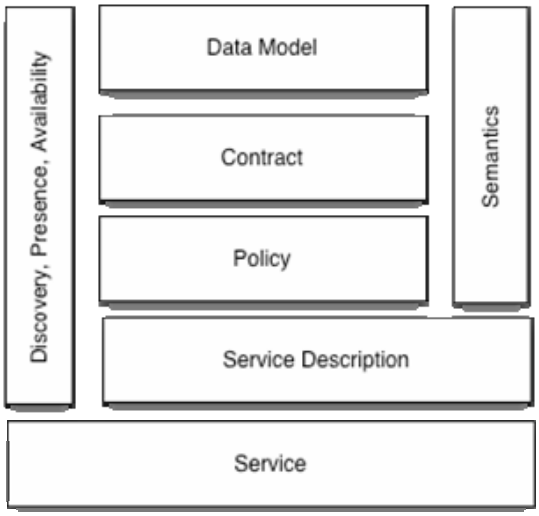
The Descriptor is comprised of metadata that articulates the interface of a service in order for Service Consumer to understand the service’s externally accessible functionality.

A Policy is a set of assertions that must be adhered to when a service is invoked.

A Contract is implied when a Service Consumer makes and invocation request to a service, in substantial alignment with the Policy declaration.

Data Model is the abstract paradigm used in the invocation and consumption of a Service. A Data Model will likely manifest itself within a concrete architecture as a set of concrete Messages.

The cross cutting concerns are defined as aspects that cross several other elements within the object model. Within the SOA-RM, these crosscutting concerns are a Semantic aspect and a Discovery aspect. Semantic agreement on what entities mean with respect to their roles in a system is necessary for service oriented architecture. Many of the components (Service Descriptions, Policies, Contracts and Data Models) need to be available for discovery by potential service consumers to determine both the suitability of a service and their ability to invoke and/or consume the service. The concept of Discovery is to gain awareness of the Presence of the elements and details of their availability.



200

201 *Figure 2-1 - SOA Architectural Model*

202

203 2.1 Services

204

205 A service is a set of functionality provided by one entity for the use of others. It is invoked
206 through a software interface but with no constraints on how the functionality is implemented by
207 the providing entity. Thus, the service could carry out its described functionality through one or
208 more automated and/or manual processes that themselves could invoke other available services.
209 A service is opaque in that its implementation is hidden from the service consumer except for (1)
210 the data model exposed through the published service interface and (2) any information included
211 as metadata to describe aspects of the service which are needed by service consumers to
212 determine whether a given service is appropriate for the consumer's needs. Thus while service
213 opacity is an essential of SOA, it is not absolute.

214

215 2.1.1 Service Composition

216

217 Consistent with the axiom of opacity, a Service Consumer cannot see anything behind the service
218 interface and does not know if one service is actually consuming and aggregating additional other
219 services. Whether a Service's functions are mapped to a set of classes in some native language
220 or another service is not important or relevant as far as invoking the service is concerned.

221

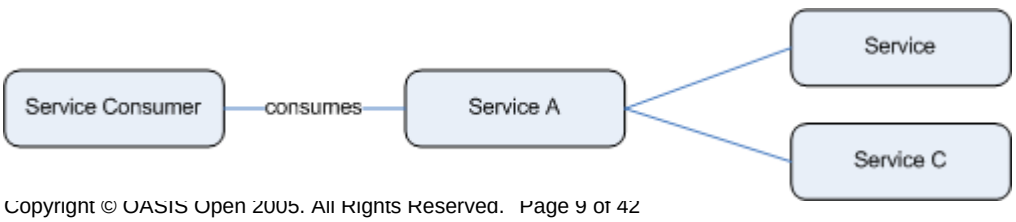


Figure 2 - Service Composition

Examining Figure 2 - Service Composition above, the service function (for service A) is described in the service description specific to that service. If completing the function depends on two or more serial or parallel paths of execution successfully completing behind the service interface (like calling services B and C) within a certain time frame, that is typically not relevant to state in the service description for service A. Ideally, the service consumer is only concerned with the service's ultimate success or failure. Mapping the functionality to success and failure is the responsibility of the service provider.

As part of hiding its implementation, when a service is invoked by multiple users in a manner such that a new service invocation is requested before a previous service invocation is completed, the mechanisms the service uses to handle the overlapping (and possibly simultaneous) invocations is not typically revealed to the requester. Indeed, the service may make use of other services providing specialized functionality to support such needs. However, there may be situations, such as quality of service requirements, where the effects of implementation choices have consequences that impact descriptive quantities included in the service metadata. Here, while the implementation details may not be specifically revealed, information derivable from these details will be available to the requester.

2.1.2 Service interface

The service interface specifies how to access the service and syntactically represents this information in a standard, reference-able format. It prescribes 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 data model. Note, that the service may be invoked without requiring input from the requester and may accomplish its functions without providing any return or feedback to the requester.

In addition to conforming to a standard, reference-able syntax, the service interface must also make consistent use of SOA semantics as defined in this reference model. This may be represented as a mapping between SOA semantics and the chosen interface syntax. Note, the specific domain semantics of the service provider and service consumer are beyond the scope of this reference model but the reference model does [DOES IT?] address the need for the service interface to enable providers and consumers to unambiguously identify relevant definitions for their respective domains. See detailed discussion of SOA semantics in section 2.3.

2.1.3 Service description

As discussed above, the concept of a SOA is based on the use of a service without the service consumer needing to know the details of the service implementation. Hidden details could include the specific logic applied, the mechanism for encoding the logic, or the physical means by which the service is hosted. However to use a service, a service consumer must know

1. The service exists and is available
2. The service performs a certain function or set of functions
3. The service operates under a specified set of assumptions, constraints, and policies
4. The service can be invoked through a specified means, including inputs that the service requires and outputs that will form the response to the invocation.

The mechanisms to establish presence and availability are discussed elsewhere in this document; items (2) through (4) form the service description. The service interface, as described above, describes the basics of the required inputs and outputs that make up the data model for item (4). The description of functionality and assumptions, constraints, and policies are less specific and more dependent on the context to which the service provider and consumer are aligned.

The most difficult of the description items is item (2), that capturing service functionality. This aspect of 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. This may include, among other possibilities, a textual description intended for human consumption or identifiers or keywords referenced to specific machine-process-able definitions. The specification of a single description vocabulary is not only beyond the scope of this reference model; it is unlikely that such a single, general-purpose vocabulary exists or can be developed.

Assumptions, constraints, and policies are particular descriptive aspects of a service that control if and under what circumstances the service is appropriate or accessible for use. While the line between each of these is vague, the common requirement is that they must be expressed in such a way as to enable corresponding instances to be processed in a consistent, logical fashion. In essence, assumptions, constraints, and policies not only provide information but also the elements of a logical framework that can be interpreted and enforced.

Assumptions in the technical sense provide conditions that underlie the derivation of the service functionality. For example, a service that calculates the pressure distribution around a body might indicate whether the solution assumes compressible or incompressible flow and whether shock conditions fall within the service capabilities. The appropriate service would be different for a submarine vs. a small passenger aircraft vs. a jet flying at supersonic speeds. This example not only highlights the need to evaluate a set of assumptions but also the need to express assumptions specific to a particular domain of discourse.

Constraints, like assumptions, can restrict how a service is to be used. While it may not be an underlying assumption, it could be a precondition to a service being accessed. For example, a constraint could be that a prospective consumer needs to prove there is a paid subscription before the service can be accessed.

Policies express a set of assertions and obligations to which service providers and/or consumers must adhere. To make use of policies, these must be expressed in a way that characteristics of the provider or consumer can be identified to evaluate whether the policy conditions are being satisfied. For example, if policy states that employee salary information can only be accessed by their direct supervisor or the group's designated HR representative, then the required conditions must be visible to and identifiable by the policy evaluation mechanism. There may also be a need to capture the results of policy evaluations and such results may be appropriately included as part of the metadata of the service or the participating entities. Metadata is discussed below and policy is more fully discussed in Policies and contracts.

2.1.3.1 The relationship between service description and service metadata

The service description may be considered part of or the complete set of the metadata associated with a service (see Appendix META for a discussion of metadata in the context of a SOA) but in any case, the service description overlaps and shares many common properties with service metadata. As noted in Appendix META, 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 assumptions and policy may vary. However, the mechanisms to specify the service description should follow a standard, reference-able format that can accommodate the necessary variations and lend themselves to common processing tools (such as discovery engines) to manipulate the service description.

Consider, for example, the descriptive elements that may apply for a data resource vs. a processing resource. Here, we will assume the resources to be distinguished as follows:

A data resource is a source of content that accepts a request and returns a value or set of values in response. The return can be an entity (such as a particular schema), an attribute of an entity (such as when the schema was last modified), or any numerical or textual value or set of values. The content can be static objects stored in some repository or dynamically generated through the use of a processing resource.

A processing resource is one that accepts a task and return a status indicating the extent to which the task was completed and information on how the state of entities changed as a result of the processing. One or more processing resources may be invoked as part of a process of submitting a query and being returned a response. From the standpoint of a user (either human or machine), it is unimportant what combination of data and processing resources are invoked as long as the request is satisfied.

Both types of resources are likely to have items such as a name, a textual description, and possibly a set of descriptors/keywords with a pointer to the vocabulary definition from which the descriptors/keywords are taken. Both resources may also identify responsible parties, including who is responsible for operations, who is responsible for design, who is responsible for implementation

The description of services to establish their discovery, presence, and availability is imperative for a service-oriented architecture. From a metadata standpoint, there is no significant difference in describing a service that may be considered integral to the SOA infrastructure as opposed to one defined by a specialized participating community. The metadata must support (1) discovery by a user looking for a service to compose a solution, (2) mutual evaluation by the service and the prospective user to decide if service authorization requirements are met by the user and usability/applicability requirements are met by the service, and (3) access/invocation after service and user have mutually satisfy their conditions for use. A common metadata set includes familiar elements such as name, description, and keywords. Access/invocation and pedigree are included per their descriptions above, and security and SLA metadata, while not fully analyzed but tentatively identified as Upper Level metadata, are included in the notional elements.

The Service metadata does include several unique elements. Two instances of Responsible Party metadata are used: one to identify the entity that is responsible for the design, development, and maintenance of the software that comprises the service; a second entity is identified who is responsible for service operation issues for NCES users. Both instances of Responsible Party will likely use the Person/Organization or Title/Position building blocks.

Service metadata also contains Version and Status elements. Both of these should reference documentation that defines the values which are applied to these elements. The version numbering may follow a format specified through NCES governance, but a general requirement to include a pointer to a defining document supports the current directed use, modifications to the directed use, and any other versioning algorithm that the development community finds useful. The Status element is intended to reflect the status as determined by the developer (e.g., current version, beta, superseded), and may be seen as a counterpart to pedigree which is an evaluation from the users. As with version numbering, NCES governance could specify authoritative status states, but the reference to a defining document support both effective governance and future contingencies.

It is assumed that there may be multiple instances of the Access/Invocation metadata bin to expose different aspects of a service. For example, the access may be different depending on the guaranteed quality of service. Recall that the Access/Invocation metadata bin included a list of pre-qualified users as a notional means of speeding access without repeating all the authorization checks previously satisfied for that access point. For a service with multiple access points, the Service metadata includes the notional capability to provide a global prequalification list that alleviates the need to provide a duplicate list with each Access/Invocation metadata instance. It is expected that any differences in the local list would override prequalification in the global list.

2.1.3.2 Application metadata

In the AoA analysis, application are considered processing resources that a service is designed to invoke. For example, a Discovery service could access several discovery engines and Application metadata would provide the basis on which one would choose a particular application and how that applications would be executed. Structural metadata detailing an application API might be used by a service developer in developing service access to the application.

The purpose of Application metadata is to support cataloguing of applications that were not originally intended for Web service access. In most ways, the information included in the Application metadata is the same as identified for a service, and most of the Service metadata discussion is applicable here. The important point is again that many applications may provide similar capabilities and may even be alternatives to be invoked by a single service. It is the purpose of the metadata to allow the user to discriminate among the alternatives and invoke the one that is best suited for the current tasking needs.

2.1.3.3 Data Source metadata

Data Source metadata provides the counterpoint to Service metadata. The metadata must support (1) discovery by a user needing data, (2) mutual evaluation by the data source and the prospective user to decide if data source authorization requirements are met by the user and user usability/applicability requirements are met by the data source, and (3) read and write, as appropriate, after the data source and user have mutually satisfy their conditions for use. The data source does not have to be a database; in fact, the user may have no specific knowledge of how the data is stored or accessed. Any information needed to evaluate the applicability of the data source should be supported via constraint descriptions and established through pedigree evaluation.

Most of the notional elements for Data Source metadata are the same as described for Service metadata, and those explanations also apply here. The one notional element added was update cycle. This may not be applicable to all data sources. For example, a database tracking truck parts is likely to be continually updated and it is impractical and of questionable use to update the data source metadata every few minutes. However, simply indicating the refresh cycle is continuous may be of use. For a data mart, the refresh cycle is more relevant because it indicates whether the contents may be stale. In this case, the policy information may be useful because it can provide a rationale as to why the data resource should be considered valid over that refresh cycle. The analysis indicates that knowing data is current is a significant concern and it is likely that this set of elements will gain better definition through operational testing.

Other metadata bins in this category:

Currently, resources tend to fall under either data resources or processing resources. This metadata bin would be expanded if additional resource type is defined.

Takeaway

The metadata described in this section is the culmination of the groundwork laid from the building blocks on. More atomic building blocks are repeatedly reused, enabling an immediate degree of interoperability even if a user would not understand unique metadata added at a higher level. For example, the Responsible Party instances reuse the structure defined by Person/Organization and thus it should be relatively easy to find a contact point to explain other concepts. The Access/Invocation metadata provides a common description of a resource access point, where the description supports a range of activities that include whether access is allowed or useful at all.

As noted, several of the metadata bins support instances that will change during the associated entity's life cycle, and such changes will be made by authorized agents other than the original metadata producer. This implies that Resource metadata, at the top of the pyramid, must also have support for update. In addition, resources are assumed to be long-lived and their function and mission space may evolve over time. A metadata system must support and provide configuration management as the building blocks included and the Upper Level metadata referenced change over time to match the new contexts.

2.2 Policies and contracts

Broadly speaking, a policy represents some form of constraint or condition on the use, deployment or description of an owned entity. Policies are inherently unilateral – any participant may have policies about issues that are important to them. A contract, however, is a policy that has been agreed to.

A contract can refer to everything from the detailed description of the service interface to the legal contract entered into when two or more parties use a service. However, the SOA RM focuses on those agreements necessary for a successful interaction with a service.

2.2.1 Service Policy

Abstractly, a policy is an assertion that expresses intent on the part of a participant.

Policies apply to many aspects of SOAs: to security, to privacy, manageability, Quality of Service and so on.

Policy assertions may be, but need not be, written down in a formal machine process-able form. Languages that permit policy assertions also range in expressivity from simple propositional assertions to modal logic rules. However, the SOA RM 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. It would be natural for the service description to contain references to the policies associated with the service.

Associated with policies is the concept of enforcement. Enforcement is the realization of the policy: an un-enforced policy is simply an abstract logical proposition. However, how a policy is enforced, or even whether a policy is enforced is not a relevant part of the reference model.

A policy always represents a participant's point of view. For example, a provider of a service may have a policy that all users of the service must be authenticated prior to their access to certain functions. This policy is one that may be enforced by the service provider independently of any agreement from potential users of the service. Similarly, someone's agent may embody a privacy policy independently of any services the agent interacts with.

2.2.2 Service Contract

Where a policy represents an assertion from the point of view of a participant, 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. However, the concept of a service contract within the SOA RM applies primarily to the requirements for the successful use and provision of services.

A contract may be, but need not be, expressed in a machine process-able form. It seems significantly likely that an executed contract will not be in a machine process-able form; especially for commercial agreements. However, languages that can express policies, especially the more powerful variants can often also be used to express machine process-able contracts.

Each contract may be associated with a life-cycle. This life-cycle has three main phases: a negotiation phase, an active phase and a completion phase.

While it is possible that a specific negotiation phase precedes an agreement to a contract, often it is more implicit. For example, merely attempting to interact with a service may represent an agreement to follow the prescribed procedures for using the service.

Often a contract specifies policies that are assumed to be in force during the active phase of the contract. As such, those policies are subject to enforcement in a similar way to unilateral policies.

Enforcement of an agreement will depend on the nature of the agreement: violating an infrastructure-level agreement is likely to lead to errors and unexpected results. Violating a commercial agreement is likely to lead to loss of service or other legal remedies.

While there may be many kinds of contract, we envisage three main kinds of contract that may apply in service oriented architectures: the contracts that represent the valid use and provision of services, the contracts that represent the permitted uses of services and the contracts that result from using services.

For example, the service description may contain descriptions of the interfaces of a service – the kinds of data entities expected and the names of the operations supported – and may also contain choreographic descriptions of the order of interactions. Such descriptions may range from simple identifiers implying a mutually understood protocol to a complete description of the vocabularies, expected behaviors and so on.

However, a valid use of a service is not equivalent to a permitted use of the service. For example, one may present a syntactically correct request to a service for withdrawing money from an account. If that request is not accompanied by a suitable authentication, then that request is typically denied – it is not permitted. Many security considerations and quality of service considerations lie in this realm of agreement.

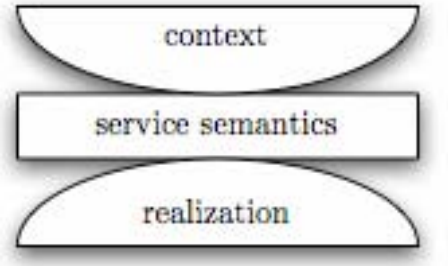
Often the purpose of interacting with a service is to effect a further agreement. For example, one use of a book-selling service is to cause a book to be purchased and delivered.

This kind of contract is an important aspect of the rationale for deploying Service

Oriented Architectures; however, such contracts are beyond the scope of this SOA RM.

2.3 Semantics

A service represents an action boundary between the infrastructure that the service is deployed over and the business context in which it is deployed. Addressing the semantics of this boundary in the appropriate manner is one of the key challenges to developing large scale reliable systems.



The semantics of a service are the shared expectations about the service. In many environments, this cannot be represented in a monolithic fashion: the semantics of services have a natural layering.

2.3.1 The layers of a semantics of service

Fundamentally, we expect that all services deployed in a SOA have an intended purpose. That purpose is the linchpin by which we measure the expectations for a service and is the basis of its semantics. The purpose of a service is the highest-level semantic characterization of the service.

The requirements for reliably and mechanistically interacting with a service represent a baseline for the semantics of a service. This includes any metadata required to contact the service; but also includes such aspects of a service as message transport, data encoding and so forth.

The requirements and expectations for the content of any data interchanged. This corresponds to the data model of SOAs

The requirements and expectations for the appropriate sequences of interactions. This may include dependencies relating to the stateful requirements on the entities interacting via the service.

The requirements and expectations about the intended effects of any interactions – or sequences of interactions.

The policies and contracts that may be relevant to the service interaction. Policies and agreements may apply to all levels of the semantics of a service.

In principle, the semantics of a service reflects many aspects of its establishment – from the format and structure of any data communicated between the participants of a service interaction to on the participants to the expected effects of successfully interacting with the service.

2.3.2 Metadata

One of the hallmarks of a Service Oriented Architecture is the degree of documentation associated with it. The purpose of this metadata is to facilitate integration, particularly across ownership domains. By providing descriptions, the task of designing client applications that make use of a service is considerably enhanced.

In this spirit, we might also expect that the different semantic aspects of a service outlined above may also be documented. Such documentation may be in machine process-able form – in which case it is commonly referred to as metadata – or it may be in informal written form – in which case it is commonly referred to as documentation.

If documented in metadata, a service's semantics has many possible uses: it 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.

However, it is not essential to the concept of SOAs that the semantics of a service be so completely described.

2.3.3 Vocabulary

For successful interactions, the various entities must have a shared understanding of the content of interaction as well as the expected behaviors. This includes the meaning of the symbols and strings used in the communication – the shared vocabulary.

In some cases, the shared vocabulary can be as simple as a shared data model schema. However, in the context of the Internet, with applications spanning ownership domain boundaries, we are often forced to deal explicitly with meaning – because we cannot rely on the same understanding of terms when different systems are integrated.

2.3.4 Context

Since words and symbols used in a particular discourse may have multiple possible interpretations, and since different participants may have different terms for the same concepts, providing a basis for selecting the correct interpretations of words and symbols is a key to building reliable systems. Context metadata surrounding a particular discourse helps to establish correct interpretation of actions and data between the participants involved in that discourse.

2.3.5 Autonomy

Autonomy is inherently a relative concept -- one is autonomous from control by something. In the case of a SOA-style system, we expect that a very common situation is one where the providers of services and the consumers of services will often belong in different ownership domains; as such, they will inevitably have certain rights and freedoms not normally applicable to closed systems. In particular, providers of services can refuse service, and consumers of services may arbitrarily abandon connections.

In this context, it is advisable that SOA architectures be designed from the perspective that service providers and consumers are autonomous from each other. Such a constraint for service participants leads to a reliability benefit in SOA architectures – they will be inherently more robust and reliable than closed architectures.

2.3.6 Data/Information Model

The goal of SOA Data/Information Model is to specify an abstract interface and data model for exchange of data among SOA entities. Entities in SOA need a standard way to (de)serialize data, extract and/or construct metadata, and infer service semantics.

At the highest level data in SOA can be classified as private and public. Private data includes the data used by a service. The data model of this data is private to the service implementation. This data model does not provide a mapping or bridge to private data. In an exchange this type of data is carried as payload inside of an exchange data unit.

Public data includes data that embodies the state, property and parameters of an SOA. Public data should be available at standard interfaces and in standard formats.

The data model should also define standard mechanisms for services to extract metadata that may be serialized with data.

2.4 Discovery, Presence and Availability

2.4.1 Discovery

Discovery, in the context of Service Oriented Architecture, is the act of detecting, identifying, understanding and selecting a service within the constraints and boundary conditions of a service fabric. It must immediately be pointed out that it is the service description that is discovered not

the service itself. A service can only be consumed once it has been located and the interface proffered by the service description is enjoined within the consuming entity.

2.4.2 Structured vs. Unstructured Discovery

By and large, unstructured service descriptions do not lend themselves to be understood by a large audience. It is reasonable then for sagacious consumption that service descriptions are in the form discussed in section 2.1.2 in order to render them consumable resulting in larger client uptake. However there is no reason other than the rationale presented here and above that a service description be well structured. It is up to the provider to decide a service description format that best suits his/her needs and those of the intended consumers.

2.4.3 Service Fabric Constraints

Should a service participate in a fabric where the fabric dictates the structure of service descriptions, the provider must then tailor his service description to the accepted format adopted by the fabric management entity. Although service description constraints probably exert restrictions onto the service provider, the “common model” approach allows for a baseline of known semantics and quicker application incorporation.

2.4.4 Discovery Methods

In general, there are two primary methods by which an entity can be informed about the existence of another and they are:

Discovery by broadcast, which is autonomously receiving information about an entity or,

Discovery by detection, which is seeking information about other entities on one’s own volition either intentionally or accidentally.

The following diagram illustrates the detection models.

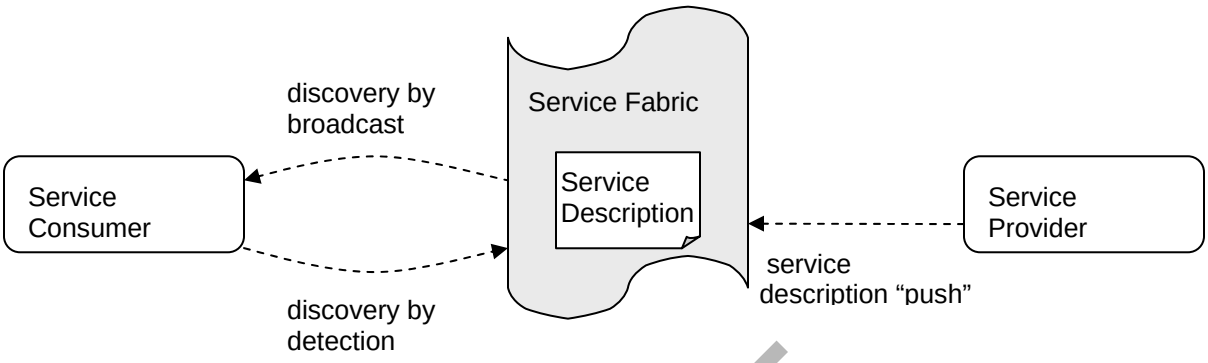


Figure 2-2 - Service Description detection methods

The broadcast method for information (in our case service) discovery is used in a number of technologies and on the Internet today with some success; however there are some known limitations and issues related to this method. For example, if an entity issues an information broadcast, specifically a service description “push”, and the other entity (service consumer) is not in a state capable of receiving it, the information is not captured, cannot be acted upon, and is considered lost.

As previously mentioned, discovery by detection has two variants, accidental and intentional. Accidental detection is haphazard at best and reward less at worst. A consumer looking for a suitable service could search previously known locations and if unsuccessful, could then search other random locations ad infinitum. As well, a consumer could send out a broadcast query fabric wide (if supported by the fabric) and then select from all replies within a certain time frame the service that best meets its need. But, how does the consumer know all replies have been received and hence the choice made the correct one?

Intentional discovery by detection is typically through a known medium. This method of discovery has seen support from the standards bodies and significant investment by the industry. Technologies such as registries and standardized search engines provide for well-defined query semantics simplifying and removing the burden from the service consumer and placing it onto a

well-known entity or agent (which may also be a service unto itself). There are several competing standards that satisfy the requirements for intentional discovery by detection each with known issues and limitations.

2.4.5 Identification, Understanding and Matching

Once a service description has been located by whatever means, it needs to be identified and the contents ultimately understood. For example, should two service descriptions resulting from a service query have the same name, there must exist a way to uniquely identify them within the fabric in order to classify them appropriately and consume them intelligently within the context of the consumer and provider. Unique identifiers can be easily constructed based on UUIDs or URIs or some other method designed by the fabric management entity (should one exist) but it is ultimately up to the fabric owner(s) to decide if a identification method is necessary, then agree upon its design and enforcement policy.

As part of the syntactic coupling required by service consumption, service resource (and specifically service parameter) matching is an integral part of the overall semantic compatibility model. Understanding of intent is as important as the understanding of the resource requirements. For example, in digging a hole in the ground, it is equally important to understand what a hole in the ground is, as having the shovel by which to dig it.

The SOA Reference Model illustrates where the semantic component has bearing and influence. Understanding of the service description via a normalized service description template (or some such design artifact) assists in the understanding of the service. If the service description can be parsed and dropped into some object that is easily understood by a human or by a machine within the consumer's context, then the service description has achieved its syntactic and semantic goals.

2.4.6 Presence and Availability

By definition, availability is the ability of an entity to be utilized or consumed within the context of its environment. Previously we have distinguished between the service description and the service itself, and based on this and the definition of availability, service descriptions and services must be discussed separately.

The availability of a service description is straightforward. A service description is either present in some form on the fabric or it is not. In other words, a service description can be discovered or not. In the case where a service description is discoverable but in a form that cannot be understood by a consumer, it should be considered non-existent to that consumer.

A service however has significantly more latitude as to its availability. A service description may indicate, for example, certain hours of operation and hence the service need only be operative during those hours. A service can also be in a failure state unable to respond say due to a hardware fault, but its service description is still available implying the service itself is available when in fact it is not. It is completely up to a derived architecture to specify the operational service model for unavailable services and the runtime availability of service descriptions relating to failed or state based services.

Furthermore for services, one can extend service availability to the “flexible execution” model whereby a service is always available but only executes or instantiates when a “message” is actually received by the service endpoint to consume it. This model, used by dynamic and agile computing technologies, is becoming more and more prevalent as the industry moves to virtualization of the computing enterprise. Of course, the resources need to be available for service invocations should an enterprise utilize this model, but as with all virtualization techniques, there is a point at which the resource base is exhausted and “requests” must be refused or stored for later execution.

3 Conformance Guidelines

The authors of this reference model envision that architects may wish to declare their architecture is conformant with this reference model. In order to be conformant to this reference model, a mapping must be made from each core element of this reference model to components of the conformant architecture.

The following guidelines must be followed for an architecture to be conformant with the SOA Reference Model:

- All services shall be opaque [see Services and Service Composition]
- Every service shall have precisely one canonical service description [see The Reference Model]
- Every service description shall contain at least the following elements [see Service interface]:
    - Data Model [see Data/Information Model]
    - Policy [see Service Policy]
    - Contract [see Service Contract]
- There shall exist a mechanism to convey awareness of a service to all consumers [see Discovery, Presence and Availability]
- Every service shall advertise their service description via this mechanism [see Discovery, Presence and Availability]

4 References

4.1 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.

4.2 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.

(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.

838 Encapsulation

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

842

843 Framework

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

846

847 Interface

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

849

850 Mediation

851 The transformation, routing, validation and processing of messages.

852

853 Message

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

855

856 Metadata

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

859

860 Negotiation

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

863

864 Ontology

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

867

868 Opaqueness

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

871

872 Policy

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

875

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

915 rapid and economical implementation in response to new requirements thus ensuring that
916 services respond to perceived user needs.

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

920

Appendix B. Use Cases and Examples (Non-Normative)

1 Introduction

This section is non-normative. Employing use cases for increasingly complex scenarios, it explores the requirements for developing Service Oriented Architecture specifications using the SOA-RM.

Three scenarios of an SOA were considered - a simple, an intermediate and a complex, multi-service example. In the simple case there is only one service, which receives and satisfies a request. The intermediate case expands the simple case to use multiple services in one entity. In the complex case, the scenario encompasses multiple services located in different entities, both interior and exterior to the prime service, which directly receives the request from a consumer. As we move towards more complexity, each service implicitly incorporates all the constraints, requirements and solutions of the simpler services.

The reference model must be sufficient to guide writers of an SOA specification that will satisfy these scenarios. A given specification does not have to satisfy all the scenarios and the reference model must be flexible enough to support specifications that cover only one or two of the scenarios as well as specifications that cover all three scenarios.

2 Use-Cases

These use cases will be used to test the adequacy and completeness of the reference model for developing specifications for a Service Oriented Architecture for different classes of real-world instances.

2.1 Simple SOA

This use-case describes the simplest type of SOA; a single service, which supports transaction based security, e.g. SSL and a simple policy. No other QoS is required for the simple case. The scenario is a short-lived, atomic transaction. Consequently, it can be considered to be an ACID transaction, i.e. one that is Atomic, Consistent, Isolated and Durable. As long as a transaction can be completed as an ACID transaction it greatly simplifies the resulting Service Oriented Architecture. There is no need for complex compensation as the pending transaction can be frozen and simply rolled back in the case of failure. Further, the single service eliminates the need for correlating the activity of multiple services. Of course, the service may have a potentially large number of applications to complete its work, but that is out of scope for an SOA specification.

Figure 1 shows the flow for the simple SOA case. When the service receives a request, it accesses its policy to determine if the request can be satisfied using the simple scenario. If this is true it retrieves its security requirements from the policy. Then the service checks whether the request meets its security requirements. Other situations for refusing the request may exist, for instance, where the provider is unable to handle a request in the time that the consumer requires.

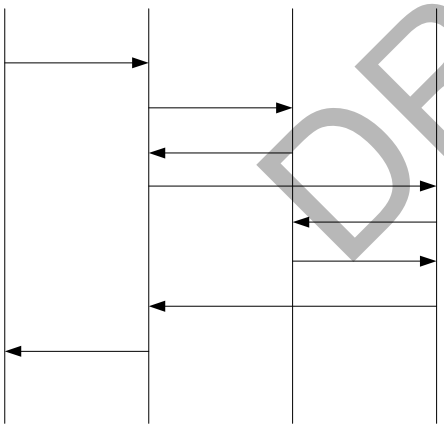


Figure 2-1

If the consumer requirements are met, the service processes the request and returns the results; otherwise the service performs its fault activity.

Requester Service Policy Security

Since the root service may receive multiple requests, it must have some means of identifying the different requests. The identification, which is sent to the requester in the reply, must be understandable to the requester. The service should have some means to advertise its service so the requester can discover it. Note that once the requester discovers a service for a particular activity it may preserve this information and have no further need for the discovery service unless circumstances change.

Note that the consumer may not wish to disclose certain information in an initial service request until it can be certain that its request will be acceptable to the provider, by virtue of the fact that it conforms to the provider's policy. Thus, there may be multiple requestor messages to complete a single request.

When a fault occurs, see figure 1.1, the service performs a rollback and reports the fault to the requester. Since the transaction is short-lived and contained within one service, the simple scenario must be designed to return to its pre-existing state without any complex activities.

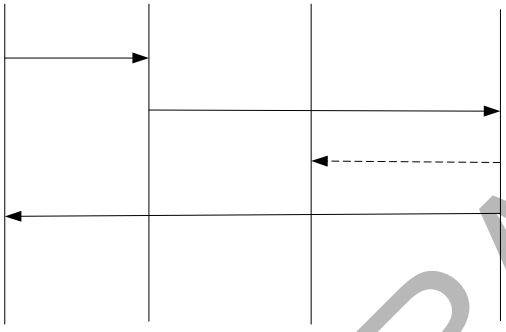


Figure 2-2

2.2 Intermediate SOA

An intermediate SOA scenario differs from a simple SOA in that it is composed of multiple services, which are located in the same domain and under the control of one entity.

The intermediate server provider satisfies requests by using multiple services behind a single service façade, the root service, exposed to the requester. If the service request conforms to the provider's policy for requests, the provider accepts the request. Otherwise, it returns a fault.

As opposed to the simple SOA scenario, the root service uses additional services to complete an activity. The root service controls the activity of the secondary services since they are all part of a single entity. Since there are multiple services, there is a need for coordination of the services.

(Coordination has the meaning here of controlling the flow of the process and is not meant to infer any existing technology.) Some of the services in this scenario may be long-lived and thus do not have the ACID properties of the simple use-case. In addition, some of the services may require intermediate information from other services to complete their activity. In the later case the root service should request the intermediate information from the supplying service and deliver it to the service or services that need the information.

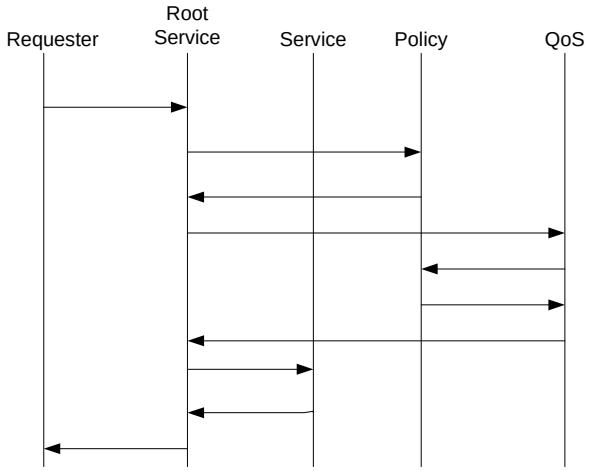


Figure 2-3

An SOA specification aimed at the intermediate scenario must be able to handle additional Quality of Service requirements beyond that of the simple service. These would include management of the secondary services and long running transactions.

In the situation where the service cannot complete its activity it must supply a compensating activity and report the fault to the root service. In turn the root service must be able to compensate for the fault or transmit the fault to all the secondary services supporting the activity and report the failure to the requestor. Each of the secondary services must perform their compensation activity.

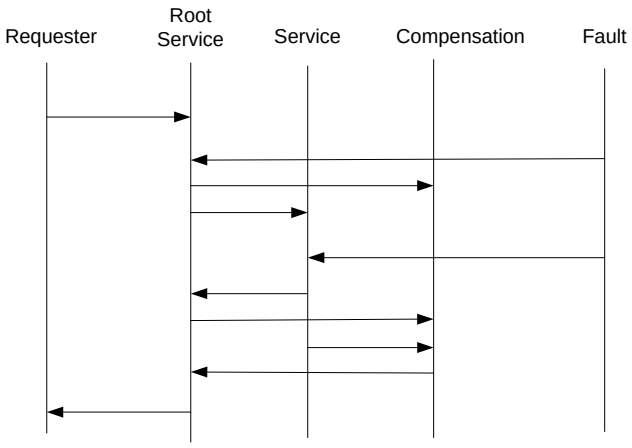


Figure 2-4

2.3 Complex SOA

A complex SOA scenario is one in which a requester submits a service request to a service provider, which requires child services to complete their requested activity. As opposed to the intermediate scenario, the child services used by the complex service may be located in the root service entity as well as other locations, which are controlled by entities distinct from the root entity. The secondary layer of services may use other services, again from other entities and which are independent of the secondary layer of entities forming a tree of arbitrary depth.

In this scenario, there may be short run services, which may be treated as atomic transactions and long running services, which may require hours or days to complete their tasks.

A complex SOA architecture has the following characteristics and capabilities:

- A hierarchy of services, i.e. the primary service must be able to control and use secondary services, which may in turn control and use tertiary services, etc.
- The ability of the services in each layer must be able to support a two phase commit and/or report and compensate if they fail or are informed of a failure elsewhere in the system.
- A child service may have more than one parent.
- A child service may have no direct knowledge beyond their parent service(s).

The primary or root service must be able to interpret incoming requests and determine:

- What tasks are required to satisfy the request
- What additional services are required to complete all the tasks

Once the primary service determines the workflow for the activity, it then has to:

- Discover and alert each of the required child services to prepare to accomplish their tasks
- Pass any required intermediate results to services that require the intermediate information. This information may pass through multiple layers of the hierarchy
- Once all the services have reported they are prepared the root service instructs them to complete their activity.

All services must possess the ability to:

- Advertise their services so that others may use them
- Compensate for an abort caused by any faults that may occur in any of the services including their own
- Understand and be able to act on instructions from their parent service(s).

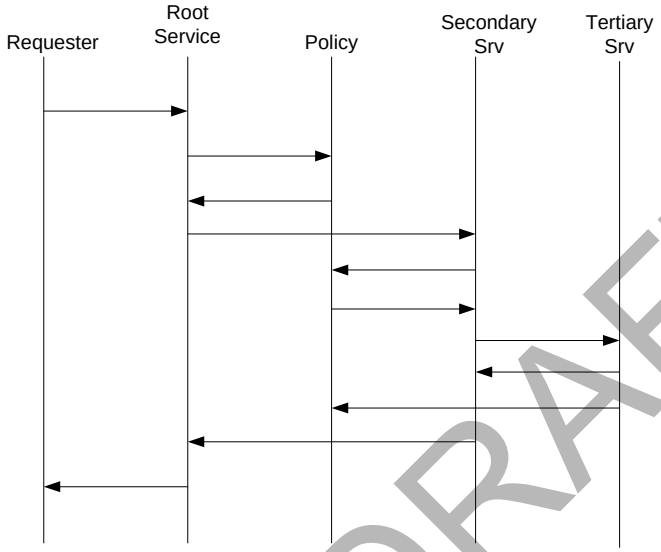


Figure 2-5

Figure 2-5 shows the activities for a complex service hierarchy. In this scenario the primary service depends on a number of layers of services to complete its activity. Lower levels of service may depend on further lower levels of services to complete their activities. Each service must satisfy the requirements of a simple service and possibly an intermediate service.

The relationship between different layers of services may be peer-to-peer or parent-child. Figure 2-6 shows an example of the hierarchal structure of the complex scenario. The colored ellipses represent the different entities. The blue ellipse represents the primary entity. The primary entity contains the root service for the activity, which has one or more children contained within its entity. The yellow ellipse represents another entity, E2, whose root entity has a parent child relationship with the primary root service. The purple ellipse, E3, represents an entity that has a

peer-to-peer relationship with the primary root. Entities E2 and E3 may have further peer-to-peer or parent-child relations with other entities.

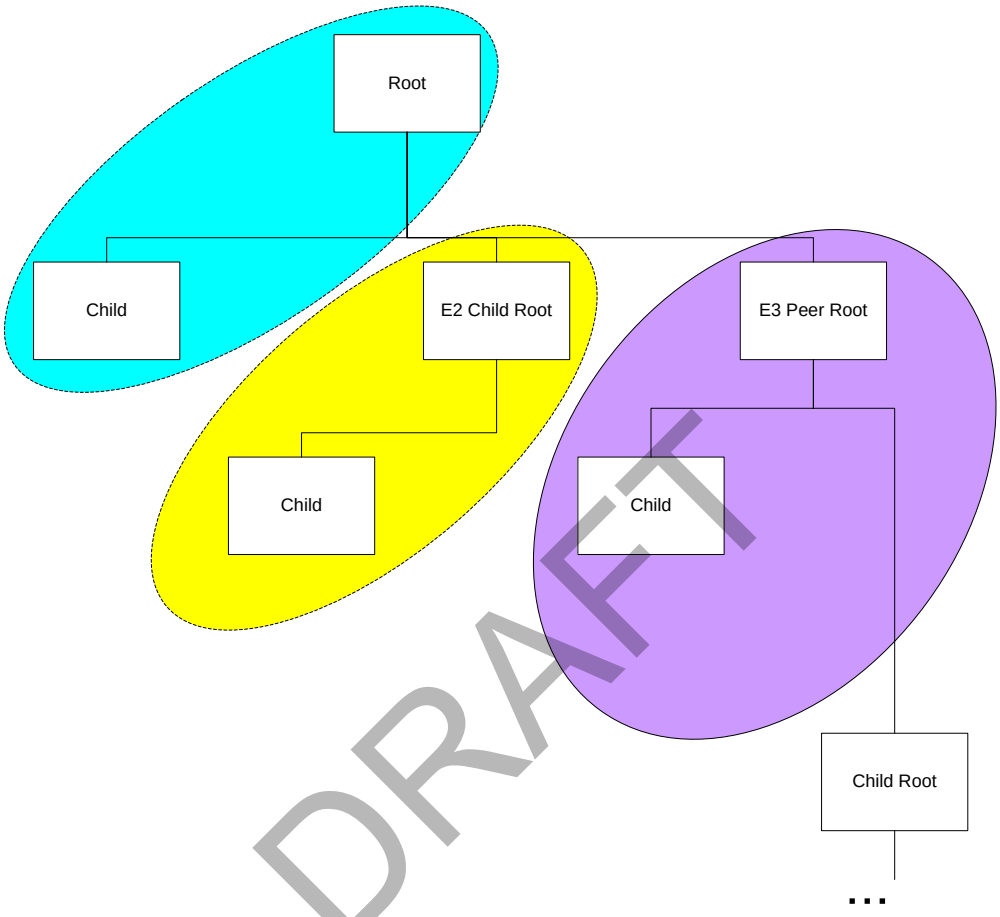


Figure 2-6

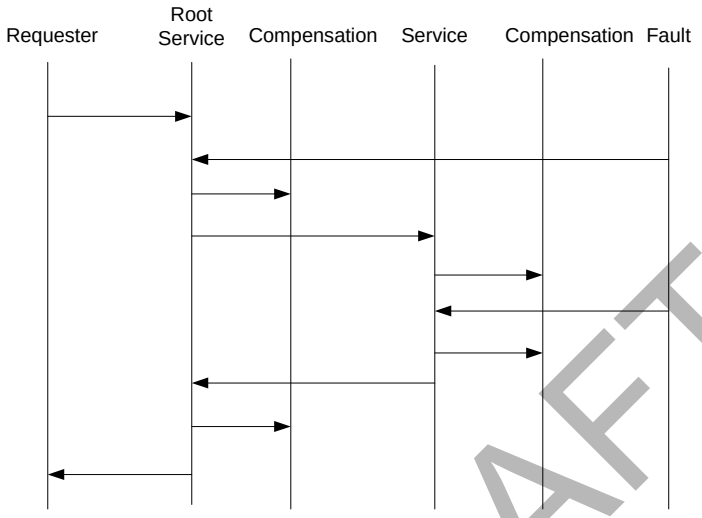
In the peer-to-peer case, each set of services is under control of a different entity and may have policy controlled by that entity. This will require policy and control negotiation between the different entities. In the parent-child case the parent entity dictates the policy and control structure of the child entities. Note that as in Figure 2-6 there may be a mixture of the two types of relationship to satisfy of a given SOA process.

1083 Any service in the hierarchy may produce a non-recoverable fault, which must be passed up the
1084 hierarchy. When it reaches the root service it is the duty of the root service to compensate for the
1085 fault or report the failure to the requestor.

1086

1087 Figure 2-7 shows the fault scenario for the complex use-case. The major difference from the
1088 intermediate case is that each of the disparate entities has its own compensation and the
1089 disparate entities relay the results of their compensation to the root service. The root service is
1090 responsible for send the fault related to the activity to other secondary services and to the
1091 requester. Each entity root service is responsible for sending the fault to its children.

1092



1093

1094 *Figure 2-7*

Appendix C. Metadata in the context of a SOA

Metadata is often described as a critical element that will support and enable a service-oriented architecture. To accomplish the many functions for which it has been associated, metadata for a SOA must go beyond being the data model in a database or the information included before a table in a data file to identify the variables represented by the values in the rows and columns. For example, metadata has been discussed in terms of the following capabilities:

Consumers must be able to search for resources without knowing the details, such as specific APIs, of the resource beforehand. This implies that the description of the resource must be expressed in a universally accessible format and, though it will be associated with the resource, the description will be external to the resource so it can be accessed without reading or otherwise invoking the resource itself.

The external description must contain sufficient detail so the consumer can decide if the resource will satisfy the current need.

If the resource is appropriate, the consumer must be able to access the resource content or invoke the resource processing without knowing the APIs or other details of the resource.

If the consumer attempts to access the resource, sufficient information must be available about the consumer so that the provider or an agent acting for the provider can determine if the access is authorized.

The producer and consumer must share a common format for the description and must also agree on how to interpret the description content. This may be accomplished by indicating a common vocabulary or distinct vocabularies for which services exist to mediate a translation.

To accomplish this, the traditional definition of metadata must be expanded. In the SOA context, we will define metadata to be *a subset of the data related to an entity that provides some critical descriptive information which is useful in some context for identifying, using, or otherwise interacting with the entity*. It provides a set of descriptive properties which serves one or more of the following functions

uniquely characterizes an entity and for which values associated with the descriptive properties allow a user (human or machine) to discriminate between one entity and another,

describes how the entity and its contents can be accessed (both procedurally and the terms of access) in either a read or write mode or executed if the entity comprises processing instructions,

contains pointers to information not explicitly part of a given metadata set but which is required as processing or control inputs by other applications or services.

Metadata often includes what the entity is, where it is located, and how to make use of it. It may describe entity properties such as format, structure/organization, context, business rules, or any other chosen elements of its integral or associated data or capabilities. It may include the calling argument to methods, invocation of services, or similar executable commands that act on the content of an instance of the entity, including accessing it from its native storage format.

Examples of metadata

Example 1 – metadata for a book

Consider the ways in which metadata for a book may be defined and used for different contexts.

For a librarian, the Library of Congress classification number is likely an important metadata element.

Conversely, for a bookseller, the classification number is not likely to be as important but the current sales price would be (while this price may not be of interest to the librarian).

The text in the book is unlikely to be identified as metadata, but specific quotes from the book may be metadata for someone advertising the book.

Example 2 – getting the weather

Consider a user looking for meteorological data. Metadata associated with a data resource that could support this includes

general document metadata with the name of the data resource and the geographic locations from where it can be accessed; metadata specific to the function of the data resource, such as the date, time, and location where the data was collected,

access control restrictions which must be satisfied (or possibly licensing terms if it is a commercial source) and a pointer to the service interface (e.g. WSDL) to retrieve the data,

a pointer to pedigree information describing the quality of the data as evaluated based on how the data was collected and processed and the accuracy of the measurements.

The request for the meteorological data may generate a log file detailing the services invoked and resources used to satisfy the request, and the log file could be archived using a network storage service. Associated with the stored log would be metadata containing a log ID, the date of the request, and the identity of the requester. Note, in this example, the log file itself is not considered metadata but information describing the log file is. A pointer to the log metadata would be returned with the requested data so the requester would both know how the request was fulfilled and be able to point to the log as a repeatable means to satisfy a similar request in the future.

As noted in both the book example in the Introduction and the weather example in the previous section, what constitutes the appropriate metadata set depends on the context of the user and the current needs to be satisfied. Thus, it is less important to have defined the perfect metadata set than to ensure that the combined metadata available can provide or support access to the critical information at the critical time.

Appendix D. Acknowledgments

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

[TODO: insert cte. Members]

Appendix E. 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.