Reference Architecture for Service Oriented Architecture Version 1.0

Public Review Draft 1

23 April 2008

Specification URIs:
This Version:
http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/soa-ra-pr-01.html
http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/soa-ra-pr-01.pdf (Authoritative)
http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/soa-ra-pr-01.doc

Previous Version:
N/A

Latest Version:
http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/soa-ra.html
http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/soa-ra.pdf (Authoritative)
http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/soa-ra.doc

Latest Approved Version:
N/A

Technical Committee:
OASIS Service Oriented Architecture Reference Model TC

Chair(s):
Francis G. McCabe

Editor(s):
Jeff A. Estefan, Jet Propulsion Laboratory, jeffrey.a.estefan@jpl.nasa.gov
Ken Laskey, MITRE Corporation, klaskey@mitre.org
Francis G. McCabe, Individual, frankmccabe@mac.com
Danny Thornton, danny.thornton@soamodeling.org

Related work:

This specification is related to:

- OASIS Reference Model for Service Oriented Architecture

Abstract:

This document specifies the OASIS Reference Architecture for Service Oriented Architecture. It follows from the concepts and relationships defined in the OASIS Reference Model for Service Oriented Architecture. While it remains abstract in nature, the current document describes one possible template upon which a SOA concrete architecture can be built.

Our focus in this architecture is on an approach to integrating business with the information technology needed to support it. The issues involved with integration are always present, but, we find, are thrown into clear focus when business integration involves crossing ownership boundaries.

This architecture follows the recommended practice of describing architecture in terms of models, views, and viewpoints, as prescribed in ANSI[1]/IEEE[2] 1471 Std. This Reference Architecture is principally targeted at Enterprise Architects; however, Business and IT Architects as well as CIOs and other senior executives involved in strategic business and IT planning should also find the architectural views and models described herein to be of value.

The Reference Architecture has three main views: the Business via Service view which lays the foundation for conducting business in the context of Service Oriented Architecture; the Realizing Services view which addresses the requirements for constructing a Service Oriented Architecture; and the Owning Service Oriented Architecture view which focuses on the governance and management of SOA-based systems.

Status:

This document was last revised or approved by the SOA Reference Model TC on the above date. The level of approval is also listed above. Check the "Latest Version" or "Latest Approved Version" location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee's email list. Others should send comments to the Technical Committee by using the "Send A Comment" button on the Technical Committee's web page at http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm.

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 Technical Committee web page (http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm.

The non-normative errata page for this specification is located at http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm.

[1] American National Standards Institute

[2] Institute of Electrical and Electronics Engineers

Notices

Table of Contents

Table of Figures

1 Introduction

Service Oriented Architecture is an architectural paradigm that has gained significant attention within the information technology (IT) and business communities. The OASIS Reference Model for SOA provides a common language for understanding the important features of SOA but does not address the issues involved in constructing, using or owning a SOA-based system. This document focuses on these aspects of SOA.

The intended audiences of this document include non-exhaustively:

- Architects will gain a better understanding when planning and designing enterprise systems of the principles that underlie Service Oriented Architecture.
- Standards architects and analysts will be able to better position specific specifications in relation to each other in order to support the goals of SOA.
- Decision makers will be better informed as to the technology and resource implications of commissioning and living with a SOA-based system; in particular, the implications following from multiple ownership domains.
- Users will gain a better understanding of what is involved in participating in a SOA-based system.

1.1 What is a Reference Architecture?

A reference architecture models the abstract architectural elements in the domain independent of the technologies, protocols, and products that are used to implement the domain. It differs from a reference model in that a reference model describes the important concepts and relationships in the domain focusing on what distinguishes the elements of the domain; a reference architecture elaborates further on the model to show a more complete picture that includes showing what is involved in realizing the modeled entities.

It is possible to define reference architectures at many levels of detail or abstraction, and for many different purposes. In fact, the reference architecture for one domain may represent a further specialization of another reference architecture, with additional requirements over those for which the more general reference architecture was defined.

A reference architecture need not be a concrete architecture; i.e., depending on the requirements being addressed by the reference architecture, it may not be necessary to completely specify all the technologies, components and their relationships in sufficient detail to enable direct implementation. Such a concrete architecture may be valuable and necessary to ensure a successful implementation; however, the detail necessary in concrete architectures may force technology choices that are not forced by the requirements per se, but by the technology choices available at the time.

1.1.1 What is this Reference Architecture?

This Reference Architecture is an abstract realization of SOA, focusing on the elements and their relationships needed to enable SOA-based systems to be used, realized and owned; while avoiding reliance on specific concrete technologies.

When designing systems that are intended to be used across ownership boundaries over extended periods of time it is necessary to address not only how the system is to be constructed, but also how it integrates with the life of users of the system and what is involved in owning such a system. In effect, we take a total cost of ownership stance on the architecture of SOA-based systems.

While requirements are addressed more fully in Section 2, the key assumptions that we make in this Reference Architecture is that SOA-based systems involve:

- resources that are distributed across ownership boundaries[3];
- people and systems interacting with each other, also across ownership boundaries;
- security, management and governance is similarly distributed across ownership boundaries; and
- interaction between people and systems is primarily through the exchange of messages with reliability that is appropriate for the intended uses and purposes.

Below, we talk about such an environment as a SOA ecosystem. Informally, our goal in this Reference Architecture is to show how Service Oriented Architecture fits into the life of users and stakeholders in a SOA ecosystem, how SOA-based systems may be realized effectively, and what is involved in owning such a SOA-based system. We believe that this approach will serve two purposes: ensuring that the true value of a SOA meeting the stated requirements can be realized using appropriate technology, and permitting the audience to focus on the important issues without becoming over-burdened with the details of a particular implementation technology.

1.1.2 Relationship to the Reference Model

The primary contribution of the Reference Model is that it identifies the key characteristics of SOA, and it defines many of the important concepts needed to understand what SOA is and what makes it important. This Reference Architecture takes the Reference Model as its starting point in particular in relation to the vocabulary of important terms and concepts.

The Reference Architecture goes a step further than the Reference Model in that we try to show how we might actually have SOA-based systems. As noted above, SOA-based systems are better thought of as ecosystems rather than stand-alone software products. Consequently, how they are used and managed is at least as important architecturally as how they are constructed.

In terms of approach, the primary difference between the Reference Model and this Reference Architecture is that the former focuses entirely on the distinguishing features of SOA; whereas this document introduces concepts and architectural elements as needed in order to fulfill the core requirement of realizing SOA-based systems.

1.1.3 Relationship to other Reference Architectures

It is fully recognized that other SOA reference architectures have emerged in the industry, both from the analyst community and the vendor/solution provider community. Some of these reference architectures are at a sufficient level of abstraction away from specific implementation technologies while others are based on a solution or technology stack. Still others use emerging middleware technologies such as the Enterprise Service Bus (ESB) as the architectural foundation.

As with the Reference Model for SOA, the Reference Architecture for SOA is primarily focused on large-scale distributed IT systems where the participants may be legally separate entities. While it is quite possible for many aspects of the Reference Architecture to be realized on quite different platforms, we do not dwell on such opportunities.

1.1.4 Expectations set by this Reference Architecture

This Reference Architecture is not a complete blueprint for realizing SOA-based systems. Nor is it a technology map identifying all the technologies needed to realize SOA-based systems. It does identify many of the key aspects and components that will be present in any well designed SOA-based system.

In order to actually use, construct and manage SOA-based systems many additional design decisions and technology choices will need to be made. For example, we identify in this Reference Architecture a

[3] Even in contexts that apparently have no ownership boundaries, such as within a single organization, the reality is that different groups and departments often behave as though they had ownership boundaries between them. This reflects good organizational practice; as well as reflecting the real motivations and desires of the people running those organizations.

mode of interaction between service participants based on some form of message communication. The particular style of message communication, the transport technologies and the message encoding technologies are all important issues that are beyond the scope of this document. Similarly, the particular governance models used in a given application will need to be elaborated on and make concrete – for example, the exact committees and their jurisdictions would have to be set.

We believe that our approach will serve two purposes: ensuring that the true value of the SOA approach can be realized on any appropriate technology, and permitting our audience to focus on the important issues without becoming over-burdened with the details.

The primary contribution of this Reference Architecture is to make clear which technology and design choices are needed and what their purpose is. For example, we identify the role of participants and their relationships in terms of social structures. The specific organizations involved; how roles are designed and how the service interaction mechanisms determine the rights and responsibilities of the participants is also beyond our scope: we identify the need for the determination but not the specifics.

1.2 Service Oriented Architecture – An Ecosystems perspective

Many systems cannot be understood by a simple decomposition into parts and subsystems. There are too many interactions between the parts. For example, a biological ecosystem is a self-sustaining association of plants, animals, and the physical environment in which they live. Understanding an ecosystem often requires a holistic perspective rather than one focusing on the system's individual parts.

From a holistic perspective, a SOA-based system is a network of independent services, machines, the people who operate, affect, use, and govern those services as well as the suppliers of equipment and personnel to these people and services. This includes any entity, animate or inanimate, that may affect or be affected by the system. With a system that large, it is clear that nobody is really "in control" or "in charge" of the whole ecosystem; although there are definite stakeholders involved, each of whom has some control and influence over the community.

Instead of visualizing a SOA as a single complex machine, it is perhaps more productive to think of it as an ecosystem: a space where people, machines and services inhabit in order to further both their own objectives and the objectives of the larger community. In certain situations this may be a difficult psychological step for owners of so-called enterprise systems to take: after all, such owners may rightly believe that since they own the system they should also have complete control of it.

This view of SOA as ecosystem has been a consistent guide to the development of this architecture.

Taking an ecosystems perspective often means taking a step back: for example, instead of specifying an application hierarchy, we model the system as a network of peer-like entities; instead of specifying a hierarchy of control, we specify rules for the interactions between participants.

The three key principles that inform our approach to a SOA ecosystem are:

- a SOA is a *medium* for *exchange of value* between independently acting *participants*;
- participants (and stakeholders in general) have legitimate claims to *ownership* of resources that are made available via the SOA; and
- the behavior and performance of the participants is subject to *rules of engagement* which are captured in a series of policies and contracts.

1.3 Viewpoints, Views and Models

1.3.1 ANSI/IEEE Std 1471-2000::ISO/IEC 42010-2007

This Reference Architecture follows the ANSI[4]/IEEE[5] Std 1471-2000 and ISO[6]/IEC[7] 42010-2007 standard. Recommended Practice for Architectural Description of Software-Intensive Systems

---

[4] American National Standards Institute

[5] Institute of Electrical and Electronics Engineers

[ANSI/IEEE Std 1471, ISO/IEC 42010]. An architectural description conforming to the ANSI/IEEE 1471-2000::ISO/IEC 42010-2007 recommended practice is described by a clause that includes the following six (6) elements:

1. Architectural description identification, version, and overview information
2. Identification of the system stakeholders and their concerns judged to be relevant to the architecture
3. Specifications of each viewpoint that has been selected to organize the representation of the architecture and the rationale for those selections
4. One or more architectural views
5. A record of all known inconsistencies among the architectural description's required constituents
6. A rationale for selection of the architecture (in particular, showing how the architecture supports the identified stakeholders' concerns).

The ANSI/IEEE 1471-2000::ISO/IEC 42010-2007 defines the following terms:

Architecture

The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution.

Architectural Description

A collection of products that document the architecture.

System

A collection of components organized to accomplish a specific function or set of functions.

System Stakeholder

A system stakeholder is an individual, team, or organization (or classes thereof) with interests in, or concerns relative to, a system.

A stakeholder's concern should not be confused with a formal requirement. A concern is an area or topic of interest. Within that concern, system stakeholders may have many different requirements. In other words, something that is of interest or importance is not the same as something that is obligatory or of necessity **[TOGAF v8.1]**.

When describing architectures, it is important to identify stakeholder concerns and associate them with viewpoints to insure that those concerns will be addressed in some manner by the models that comprise the views on the architecture. The ANSI/IEEE 1471-2000::ISO/IEC 42010-2007 defines views and viewpoints as follows:

View

A representation of the whole system from the perspective of a related set of concerns.

Viewpoint

A specification of the conventions for constructing and using a view. A pattern or template which to develop individual views by establishing the purposes and audience for a view and the techniques for its creation and analysis.

---

[6] International Organization for Standardization

[7] International Electrotechnical Commission

In other words, a view is what the stakeholders see whereas the viewpoint defines the perspective from which the view is taken.

It is important to note that viewpoints are independent of a particular system. In this way, the architect can select a set of candidate viewpoints first, or create a set of candidate viewpoints, and then use those viewpoints to construct specific views that will be used to organize the architectural description. A view, on the other hand, is specific to a particular system. Therefore, the practice of creating an architectural description involves first selecting the viewpoints and then using those viewpoints to construct specific views for a particular system or subsystem. Note that ANSI/IEEE 1471-2000::ISO/IEC 42010-2007 requires that each view corresponds to exactly one viewpoint. This helps maintain consistency among architectural views; a normative requirement of the standard.

A view is comprised of one or more architectural models, where model is defined as:

Model

An abstraction or representation of some aspect of a thing (in this case, a system)

Each architectural model is developed using the methods established by its associated architectural viewpoint. An architectural model may participate in more than one view.

1.3.2 UML Modeling Notation

To help visualize structural and behavioral architectural concepts, it is useful to depict them using an open standard visual modeling language. Although many architecture description languages exist in practice, we have adopted the Unified Modeling Language™ 2 (UML® 2) **[UML 2]** as the primary viewpoint modeling language. It should be noted that while UML 2 is used in this Reference Architecture, formalization and recommendation of a UML Profile for SOA is beyond the scope of this specification. Every attempt is made to utilize normative UML unless otherwise noted.

Figure 1 illustrates an annotated example of a UML class diagram that is used to represent a visual model depiction of the Resources Model in the Business via Services View (Section 3.2). The figure caption describes the UML semantics of this diagram.

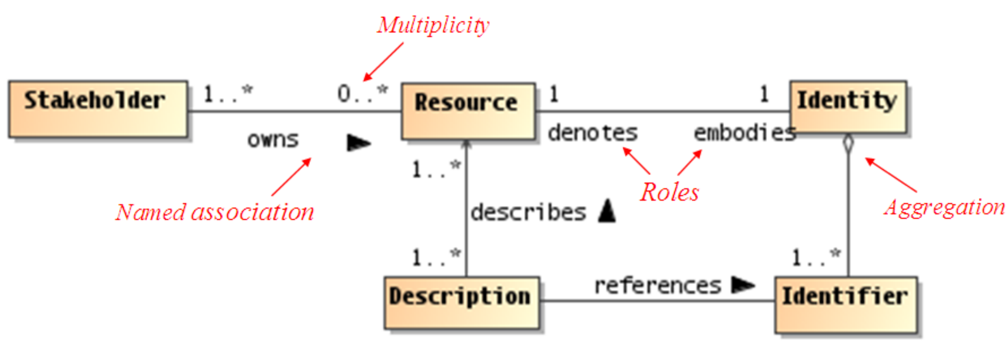


Figure 1 Example UML class diagram—Resources model.

Lines connecting boxes (classifiers) represent associations between things. An association has two roles (one in each direction). A role can have multiplicity, for example, one or more ("1..*") **Stakeholders** own zero or more ("0..*) **Resources**. The role from classifier A to B is labeled closest to B, and vice versa, for example, the role between **Resource** to **Identity** can be read a **Resource** embodies **Identity**, and **Identity** denotes a **Resource**.

Mostly, we use named associations, which is typically denoted with a verb or verb phrase followed by an arrowhead. A named association reads from classifier A to B, for example, one or more **Stakeholders** owns zero or more **Resources**. Named associations are a very effective way to model relationships between concepts.

An open diamond (at the end of an association line) denotes an aggregation, which is a part-of relationship, for example, **Identifiers** are part of **Identity** (or conversely, **Identity** is made up of **Identifiers**).

A stronger form of aggregation is known as composition, which involves using a filled-in diamond at the end of an association line (not shown in above diagram). For example, if the association between **Identity** and **Identifier** were a composition rather than an aggregation as shown, deleting **Identity** would also delete any owned **Identifiers**. There is also an element of exclusive ownership in a composition relationship between classifiers, but this usually refers to specific instances of the owned classes (objects).

This is by no means a complete description of the semantics of all diagram elements that comprise a UML class diagram, but rather is intended to serve as an illustrative example for the reader. It should be noted that this Reference Architecture utilizes additional class diagram elements as well as other UML diagram types such as sequence diagrams and component diagrams. The reader who is unfamiliar with the UML is encouraged to review one or more of the many useful online resources and book publications available describing UML (see, for example, http://www.uml.org/).

1.4 Viewpoints of this Reference Architecture

This Reference Architecture is partitioned into three views that conform to three primary viewpoints, reflecting the main division of concerns noted above: the Business via Services viewpoint focuses on how people conduct their business using SOA-based systems; the Realizing Service Oriented Architecture viewpoint focuses on the salient aspects of building a SOA, and the Owning Service Oriented Architectures viewpoint focuses on those aspects that relate to owning, managing and controlling a SOA.

The viewpoint specifications for each of the primary viewpoints of this Reference Architecture are summarized in Table 1. Additional detail on each of the three viewpoints is further elaborated in the following subsections. For this Reference Architecture, a one-to-one correspondence between viewpoints and views is assumed.

Viewpoint Element	**Viewpoint**		
	Business via Services	*Realizing Service Oriented Architectures*	*Owning Service Oriented Architectures*
Main concepts	Captures what SOA means for people using it to conduct business.	Deals with the requirements for constructing a SOA.	Addresses issues involved in owning and managing a SOA.
Stakeholders	People (using SOA), Decision Makers, Enterprise Architects, Standards Architects and Analysts.	Standards Architects, Enterprise Architects, Business Analysts, Decision Makers, Standards Architects and Analysts.	Service Providers, Service Consumers, Decision Makers.
Concerns	Conduct business safely[8] and effectively.	Effective construction of SOA-based systems.	Processes for engaging in a SOA are effective, equitable, and assured.
Modeling Techniques	UML class diagrams	UML class and sequence diagrams, component and composite structure diagrams	UML class diagrams

Table 1 Viewpoint specifications for the OASIS Reference

[8] Safety is defined by **[LEVESON]** as “the freedom from accidents or losses”.

1.4.1 Business via Services Viewpoint

The Business via Services viewpoint is intended to capture what using a SOA-based system means for people using it to conduct their business. We do not limit the applicability of SOA-based systems to commercial and enterprise systems. We use the term **business** to include any activity of interest to a user; especially activities shared by multiple users.

From this viewpoint, we are concerned with how SOA integrates with and supports the service model from the perspective of the people who perform their tasks and achieve their goals as mediated by Service Oriented Architectures. The Business via Services viewpoint also sets the context and background for the other viewpoints in the Reference Architecture.

The stakeholders who have key roles in or concerns addressed by this viewpoint are decision makers and *people.* The primary concern for people is to ensure that they can use a SOA to conduct their business in a safe and effective way. For decision makers, their primary concern revolves around the relationships between people and organizations using systems that the decision makers are responsible for.

Given the public nature of the Internet, and the intended use of SOA to allow people to access and provide services that cross ownership boundaries, it is necessary to be able to be somewhat explicit about those boundaries and what it means to cross an ownership boundary.

1.4.2 Realizing Service Oriented Architectures Viewpoint

The Realizing Service Oriented Architectures Viewpoint focuses on the infrastructural elements that are needed to support the construction of SOA-based systems. From this viewpoint we are concerned with the application of well-understood technologies available to system architects to realize the vision of a SOA that may cross ownership boundaries. In particular, we are aware of the importance and relevance of other standard specifications that may be used to facilitate the building of a SOA.

The stakeholders are essentially anyone involved in designing, constructing and deploying a SOA-based system.

1.4.3 Owning Service Oriented Architectures Viewpoint

The Owning Service Oriented Architectures Viewpoint addresses the issues involved in owning a SOA as opposed to using one or building one. Many of these issues are not easily addressed by automation; instead, they often involve people-oriented processes such as governance bodies.

Owning a SOA-based system involves being able to manage an evolving system. In our view, SOA-based systems are more like ecosystems than conventional applications; the challenges of owning and managing SOA-based systems are the challenges of managing an ecosystem. Thus, in this view, we are concerned with how systems are managed effectively, how decisions are made and promulgated to the required end points, and how to ensure that people may use the system effectively and that malicious people cannot easily corrupt it for their own gain.

1.5 Terminology

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]**.

Terms such as this “Reference Architecture” refer to this document, and “the Reference Model” refer to the OASIS Reference Model for Service Oriented Architecture”. **[SOA-RM].**

1.6 References

1.6.1 Normative References

[ANSI/IEEE 1471] *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*, American National Standards Institute/Institute for Electrical and Electronics Engineers, September 21, 2000.

[ISO/IEC 42010] International Organization for Standardization and International Electrotechnical Commission, *System and software engineering — Recommended practice for architectural description of software-intensive systems*, July 15, 2007.

[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.

[SOA-RM] C. M. MacKenzie, K. Laskey, F. McCabe, P. F. Brown, and R. Metz, (editors), “Reference Model for Service Oriented Architecture 1.0, OASIS Open, October 12, 2006.

[UML 2] *Unified Modeling Language: Superstructure*, Ver. 2.1.1, OMG Adopted Specification, OMG document formal/2007-02-05, Object Management Group, Needham, MA, February 5, 2007.

[WSA] David Booth, et al., "Web Services Architecture", W3C Working Group Note, World Wide Web Consortium (W3C) (Massachusetts Institute of Technology, European Research Consortium for Informatics and Mathematics, Keio University), February, 2004.

[WA] Tim Berners Lee, *Design Issues*, W3C, 1996. http://www.w3.org/DesignIssues/Axioms.html

1.6.2 Non-Normative References

[BLOOMBERG/SCHMELZER] Jason Bloomberg and Ronald Schmelzer, *Service Orient or Be Doomed!*, John Wiley & Sons: Hoboken, NJ, 2006.

[COX] D. E. Cox and H. Kreger, “Management of the service-oriented architecture life cycle,” "IBM Systems Journal" "'44'", No. 4, 709-726, 2005

[ITU-T Rec. X.700 | ISO/IEC 10746-3:1996(E)] Information processing systems—Open Systems Interconnection—Basic Reference Model—Part 4: Management Framework", International Telecommunication Union, International Organization for Standardization and International Electrotechnical Commission, Geneva, Switzerland, 1989.

[NEWCOMER/LOMOW] Eric Newcomer and Greg Lomow, *Understanding SOA with Web Services*, Addison-Wesley: Upper Saddle River, NJ, 2005.

[OECD] Organization for Economic Cooperation and Development, Directorate for Financial, Fiscal and Enterprise Affairs, OECD Principles of Corporate Governance, SG/CG(99) 5 and 219, April 1999.

[TOGAF v8.1] *The Open Group Architecture Framework (TOGAF) 8.1 Enterprise Edition*, The Open Group, Doc Number: G051, December 19, 2003.

[WEILL] Harvard Business School Press, IT Governance: How Top Performers Manage IT Decision Rights for Superior Results, Peter Weill and Jeanne W. Ross, 2004

[DAMIANOU] Nicodemos C. Damianou , Thesis - A Policy Framework for Management of Distributed Systems, University of London, Department of Computing, 2002.

[LEVESON] Nancy G. Leveson, *Safeware: System Safety and Computers*, Addison-Wesley Professional, Addison-Wesley Publishing Company, Inc.: Boston, pg. 181, 1995.

[WA] Architecture of the World Wide Web, W3C, 2004. http://www.w3.org/TR/webarch.

[STEEL/NAGAPPAN/LAI] Christopher Steel and Ramesh Nagappan and Ray Lai, *core Security Patterns:Best Practices and Strategies for J2EE, Web Services and Identity Management*, Prentice Hall: 2005

[ISO/IEC 27002] International Organization for Standardization and International Electrotechnical Commission, *Information technology –- Security techniques – Code of practice for information security management*, 2007

[RFC 4880] Network Working Group, *OpenPGP Message Format*, 2007

[WS-BPEL] Web Services Business Process Execution Language Version 2.0 *http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html*

[WS-CDL] Web Services Choreography Description Language Version 1.0, *http://www.w3.org/TR/ws-cdl-10/*

2 Architectural Goals and Principles

In this section, we identify both the goals of the architecture and the architectural principles that underlie our approach to the architecture.

In order to be clearer in setting the goals of this Reference Architecture, we have used a form of critical factors analysis to identify the key goals, critical success factors and requirements of this architecture. A CFA is a structured way of arriving at the requirements for a project, especially the non-functional requirements; as such, it forms a natural complement to other requirements capture techniques such as use-case analysis. The Critical Factors Analysis (CFA) requirement technique and the diagram notation is summarized in Appendix B.

2.1 Goals of this Reference Architecture

Note that not all of the requirements are mapped to solutions within the scope of this Reference Architecture. Indeed, this document can be seen as generating a series of more explicit requirements for the realizing technology.

The overall requirements are illustrated in Figure 2.

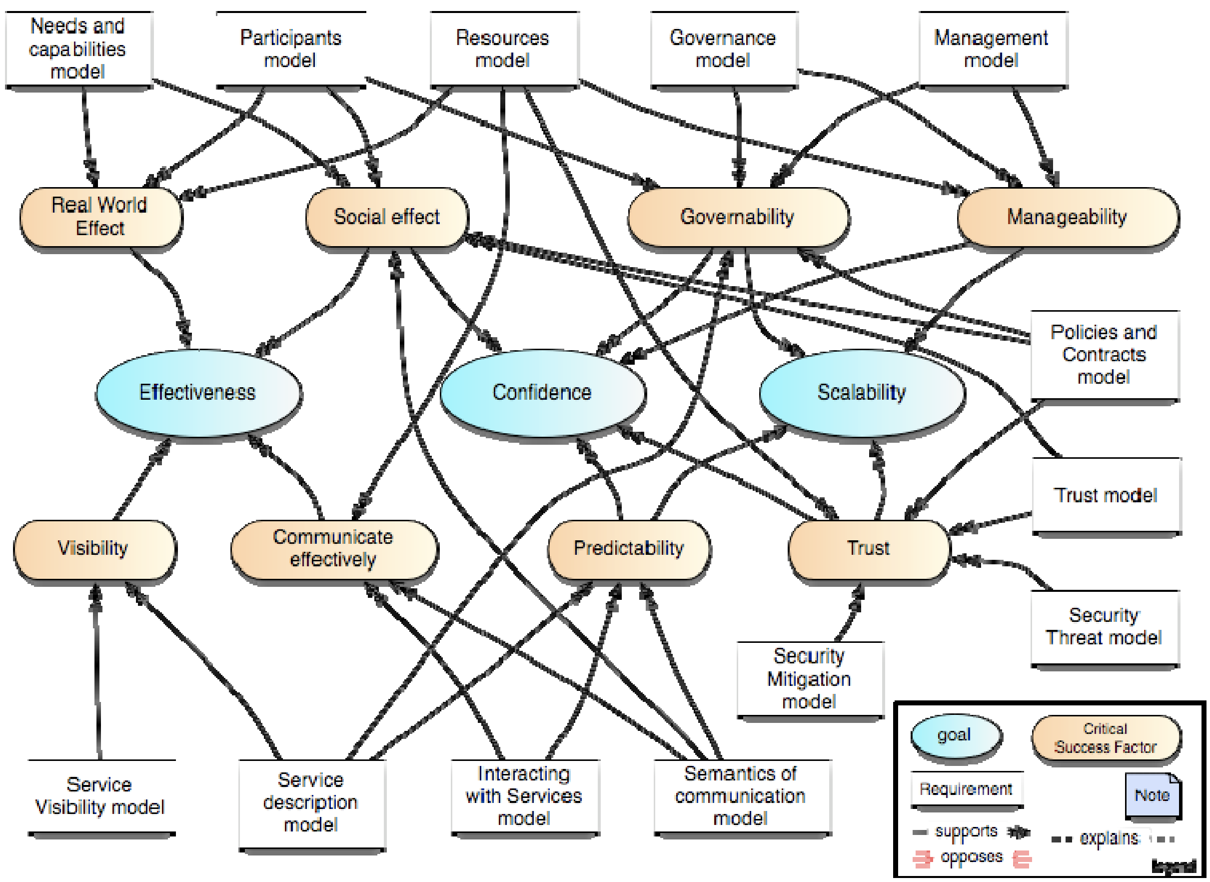


Figure 2 Critical Factors Analysis of the Reference Architecture

There are three principal goals of this Reference Architecture:

1. that it shows how SOA-based systems can effectively enable participants with needs to interact with services with appropriate capabilities;
2. that participants can have a clearly understood level of confidence as they interact using SOA-based systems; and

3. SOA-based systems can be scaled to large systems as needed.

2.1.1 Effectiveness

A primary purpose of this architecture is to show what is involved in SOA-based systems to ensure that participants can use the facilities of the system to get their needs met. Of course, not all participants' needs can be met by interacting electronically; but those that can, can be met using the framework of a SOA-based system.

The critical factors that determine effectiveness are visibility between the participants, that they can communicate effectively, and that actual real world effects and social effects can be realized. In addition, it is critical that the overall system is manageable and governable.

2.1.1.1 Real World Effect

It is of the essence that participants can use a SOA-based system to realize actual effects in the world. This implies that the capabilities that are accessed as a result of service interaction are 'wired-up' so to speak, with the real world.

We identify three models that address how service interactions can result in real world effects: a needs and capabilities model, a participants model and a resources model.

2.1.1.2 Social effect

Many, if not most, effects that are desired in the use of SOA-based systems are actually social effects more than physical effects. For example, opening a bank account is primarily about the relationship between a customer and a bank – the effect of the opened account is a change in the relationship between the customer and the bank.

The models that are important in addressing this critical factor are similar to the more general real world effect: the participants model, the needs and capabilities model and the resources model. In addition, the semantics of communication model directly supports the objective of realizing the appropriate social effect.

2.1.1.3 Visibility

Ensuring that participants can see each other is clearly also a critical factor in ensuring effectiveness of interaction. Enabling visibility requires addressing the visibility of services and the correct descriptions of services and related artifacts.

2.1.1.4 Communicate effectively

In order for there to be effective uses of capabilities and meeting of needs, it is critical that participants can not only see each other but can also interact with each other. The models that address this are the Interacting with Services model, the Resources model and the Semantics of Communication model.

2.1.2 Confidence

SOA-based systems should enable service providers and consumers to conduct their business with the appropriate level of confidence in the interaction. Confidence is especially important in situations that are high-risk; this includes situations involving multiple ownership domains as well as situations involving the use of sensitive resources.

In addition to ensuring that social effects are properly captured, other critical factors that are important for ensuring confidence are trust, predictability, manageability and proper governance.

2.1.2.1 Manageability and Governability

Given that a large-scale SOA-based system may be populated with many services, and used by large numbers of people; managing SOA-based systems properly is a critical factor for engendering confidence in them. This involves both managing the services themselves and managing the relationships between people and the SOA-based systems they are utilizing; the latter being more commonly identified with governance.

The governance of SOA-based systems requires an ability for decision makers to be able to set policies about participants, services, and their relationships. It requires an ability to ensure that policies are effectively described and enforced. It also requires an effective means of measuring the historical and current performances of services and participants.

The scope of management of SOA-based systems is constrained by the existence of multiple ownership domains. Management may include setting policies such as technology choices but may not, in some cases, include setting policies about the services that are offered.

2.1.2.2 Trust

Trust itself is clearly a critical factor in ensuring confidence. Trust itself can be analyzed in terms of trust in infrastructure facilities (otherwise known as reliability), trust in the relationships and effects that are realized by interactions with services, and trust in the integrity and confidentiality of those interactions particularly with respect to external factors (otherwise known as security).

The threat model in Section 5.2.5 captures what is meant by trust; the security models capture how external entities might attempt to corrupt that trust and how SOA-based systems can mitigate against those risks.

Note that there is a distinction between trust in a SOA-based system and trust in the capabilities accessed via the SOA-based system. The former focuses on the role of SOA-based systems as a *medium* for conducting business, the latter on the trustworthiness of participants in such systems. This architecture focuses on the former, while trying to encourage the latter.

2.1.2.3 Predictability

A factor that engenders confidence in any system is predictability. By predictability, we principally mean that the expectations of participants of SOA-based systems can be tied to the actual performance of those systems (what you see is what you get).

The primary means of ensuring predictability is effective descriptions: service descriptions document services, the interacting with services model addresses expectations relating to how services are used and the semantics of communications model addresses how meaning and intent can be exchanged between participants.

2.1.3 Scalability

The third goal of this Reference Architecture is scalability. In architectural terms, we determine scalability in terms of the smooth growth of complexity of systems as the number and complexity of services and interactions between participants increases. Another measure of scalability is the ease with which interactions can cross ownership boundaries.

The critical factors that determine scalability, particularly in the context of multiple domains of ownership are predictability, trust, governability and manageability. This is in addition to more traditional measures of scalability such as performance of message exchange.

2.2 Principles of this Reference Architecture

The following principles serve as core tenets that guide the evolution of this Reference Architecture. The ordered numbering of these principles does not imply priority order.

Principle 1:	**Technology Neutrality**
Statement:	Technology neutrality refers to independence from particular technologies.
Rationale:	We view technology independence as important for three main reasons: technology specific approach risks confusing issues that are technology specific with those that are integrally involved with realizing SOA-based systems; and we believe that the principles that underlie SOA-based systems have the potential to outlive any specific technologies that are used to deliver them. Finally, a great proportion of this architecture is inherently

concerned with people, their relationships to services on SOA-based systems and to each other.

Implications: This Reference Architecture must be technology neutral, meaning that we assume that technology will continue to evolve, and that over the lifetime of this architecture that multiple, potentially competing technologies will co-exist. Another immediate implication of technology independence is that greater effort on the part of architects and other decision makers to construct systems based on this architecture is needed.

Principle 2: Parsimony

Statement: Parsimony refers to economy of design, avoiding complexity where possible and minimizing the number of components and relationships needed.

Rationale: The hallmark of good design is parsimony, or “less is better.” It promotes better understandability or comprehension of a domain of discourse by avoiding gratuitous complexity, while being sufficiently rich to meet requirements.

Implications: Occam’s (or Ockham’s) Razor applies, which states that the explanation of any phenomenon should make as few assumptions as possible, eliminating those that make no difference in the observable predictions of the explanatory hypothesis or theory. With respect to this Reference Architecture, this is made apparent by avoiding the elaboration of certain details which though that may be required for any particular solution, are likely to vary substantially from application to application. The complement of a parsimonious design is a feature-rich design. Parsimoniously designed systems tend to have fewer features. This, in turn, means that people attempting to use such a system may have to work harder to ensure that their application requirements have been met.

Principle 3: Separation of Concerns

Statement: Separation of Concerns refers to the ability to cleanly delineate architectural models in such a way that an individual stakeholder or a set of stakeholders that share common concerns only see those models that directly address their respective areas of interest. This principle could just as easily be referred to as the Separation of Stakeholder Concerns principle, but the focus here is predominantly on loose coupling of models.

Rationale: As SOA-based systems become more mainstream, and as they start to become increasingly complex, it will be extremely important for the architecture to be able to scale. Trying to maintain a single, monolithic architecture that incorporates all models to address all possible system stakeholders and their associated concerns will not only rapidly become unmanageable with rising system complexity, but it will become unusable as well.

Implications: This is a core tenet that drives this Reference Architecture to adopt the notion of architectural viewpoints and corresponding views. A *viewpoint* provides the formalization of the groupings of models representing one set of concerns relative to an architecture, while a *view* is the actual representation of a particular system. The ability to leverage an industry standard that formalizes this notion of architectural viewpoints and views helps us better ground these concepts for not only the developers of this Reference Architecture but also for its readers. Fortunately, such a standard exists in the ANSI/IEEE 1471-2000 Std. IEEE Recommended Practice for Architectural Description of Software-Intensive Systems **[ANSI/IEEE Std 1471-2000]**; and it is this standard that serves as the basis for the structure and organization of this Reference Architecture.

Principle 4: Applicability

Statement: Applicability refers to that which is relevant. Here, an architecture is sought that is relevant to as many facets and applications of SOA-based systems as possible; even those yet unforeseen.

Rationale: An architecture that is not relevant to its domain of discourse will not be adopted and thus likely to languish.

Implications: This Reference Architecture needs to be relevant to the problem of matching needs and capabilities under disparate domains of ownership; to the concepts of “Intranet SOA”

(SOA within the enterprise) as well as “Internet SOA” (SOA outside the enterprise); to the concept of “Extranet SOA” (SOA within the extended enterprise, i.e., SOA with suppliers and trading partners); and finally, to “net-centric SOA” or “Internet-ready SOA.”

3 Business via Services View

No man is an island
No man is an island entire of itself; every man
is a piece of the continent, a part of the main;
if a clod be washed away by the sea, Europe
is the less, as well as if a promontory were, as
well as any manner of thy friends or of thine
own were; any man's death diminishes me,
because I am involved in mankind.
And therefore never send to know for whom
the bell tolls; it tolls for thee.
John Donne

The *Business via Services View* focuses on what a SOA-based system means for people using it to conduct their business.[9] The mode of business in a SOA-based system is characterized in terms of providing services and consuming services to realize mutually desirable real world effects.

The people and organizations involved in a SOA-based system form a community; which may be a single enterprise or a large peer-to-peer network of enterprises and individuals. Many of the activities that people engage in are themselves defined by the relationships between people and by the organizations that they belong to.

Thus, our tasks in this view are to model the people involved—the participants and other stakeholders—their goals and activities and the relevant relationships between people as they affect the utility and safety of actions that are performed.

The models in this view include the Stakeholders and Participants Model, the Needs and Capabilities Model, the Resources Model, and the Social Structure Model.

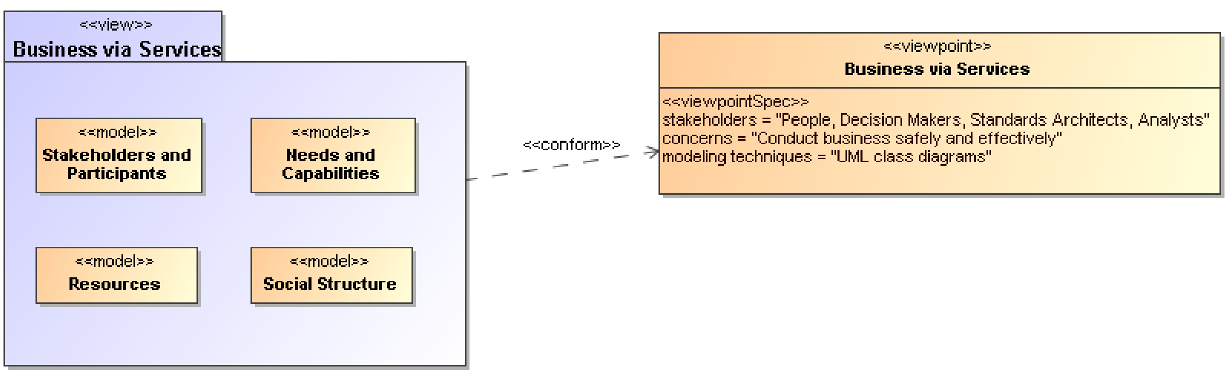


Figure 3 Model elements described in the Business via Services view

3.1 Stakeholders and Participants Model

A SOA-based system is deployed in the context of human and non-human entities capable of action. In this section we focus on the relationship between these ultimate actors and the services that they use and deploy.

[9] By *business* we mean to include any activity entered into whose goal is to satisfy some need or desire of the participant.

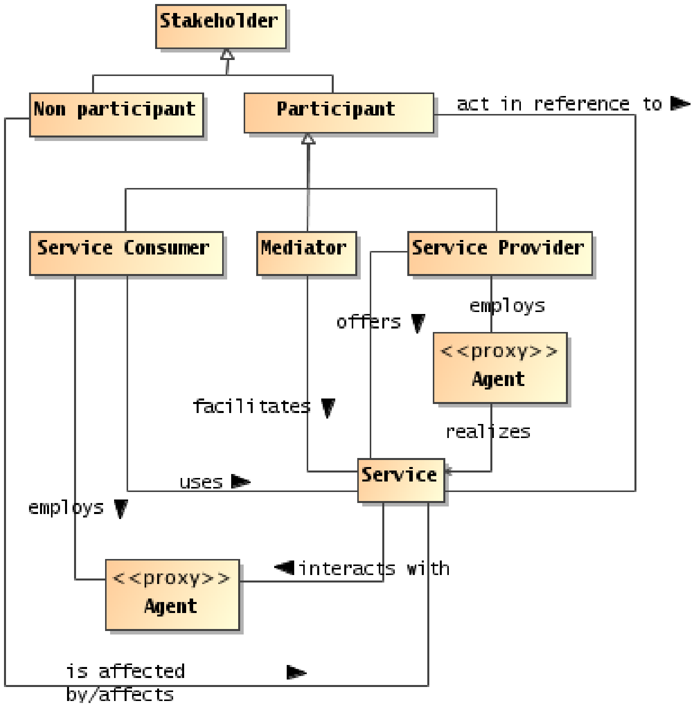


Figure 4 Service Participants

Stakeholder

> A stakeholder is an individual entity, human or non-human, or organization of entities that has an interest in the states of services and/or the outcomes of service interactions.

Stakeholders do not necessarily participate in service interactions. For example, a government may have an interest in the outcomes of commercial services deployed in a SOA-based system without actively participating in the interactions (e.g., the government may collect tax from one or more participants without being part of the interaction itself).

Participant

> A participant is a stakeholder that has the capability to act in the context of a SOA-based system.

A participant is a stakeholder whose interests lie in the successful use of and fulfillment of services. However, human participants always require *representation* in an electronic system – they require agents. Note that we admit non-human agents that have no identifiable representative as an extreme case: the normal situation is where participants are either human or organizations.

It is convenient to classify service participants into service providers and service consumers. The reason for this is twofold: an extremely common mode of interaction is where a provider participant offers some functionality as a service and a consumer participant uses that service to achieve one of his or her goals. Secondly, it helps to illustrate the dominant situation where the participants in an interaction are not truly symmetric: they each have different objectives and often have different capabilities. However, it should be noted that there are patterns of interactions where it is not clear that the distinction between service provider and consumer are valid.

Service Provider

> A service provider is a participant that offers a service that permits some capability to be used by other participants.

In normal parlance, the service provider commonly refers to either the ultimate owner of the capability that is offered or at least an agent acting as proxy for the owner. For example, an individual may own a business capability but will enter into an agreement with another individual (the proxy) to provide SOA access to that business -- so that the owner can focus on running the business itself.

Note that several kinds of stakeholders may be involved in provisioning a service. These include but are not limited to the provider of the capability, an enabler that exposes it as a service, a mediator that translates and/or manages the relationship between service consumers and the service, a host that offers support for the service, a government that permits the service and/or collects taxes based on service interactions.

Service Consumer

> A service consumer is a participant that interacts with a service in order to access a capability to address a need.

It is a common understanding that service consumers typically initiate service interactions. Again, this is not necessarily true in all situations (for example, in publish-and-subscribe scenarios, a service consumer may initiate an initial subscription, but thereafter, the interactions are initiated by publishers). As with service providers, several stakeholders may be involved in a service interaction supporting the consumer.

Service mediator

> A service mediator is a participant that facilitates the offering or use of services in some way.

There are many kinds of mediator, for example a registry is a kind of mediator that permits providers and consumers to find each other. Another example might be a filter service that enhances another service by encrypting and decrypting messages. Yet another example of a mediator is a proxy broker that actively stands for one or other party in an interaction.

Agent

> An agent is any entity that is capable of acting on behalf of a person or organization.

In order for people to be able to offer, consume and otherwise participate in services, they require the use of an agent capable of directly interacting with electronic communications – a service agent. Common examples are software applications that make use of services, hardware devices that embody an agent with a particular mission, and enterprise systems that offer services.

We do not attempt to characterize service agents in terms of their internal architecture, computational requirements or platforms here.

Non-participant stakeholder

> A non-participant is any stakeholder who may be affected by the use or provisioning of services or who has an interest in the outcome of service interactions but does not directly participate in and may not be aware of the interactions.

There are two main classes of such non-participatory stakeholders: third parties who are affected by someone's use or provisioning of a service, and regulatory agencies who wish to control the outcome of service interactions in some way (such as by taxation).

3.2 Resources Model

In many instances it is important to be able to model the assets that stakeholders may have access to. The Reference Architecture itself has many instances of such resources; for example service descriptions, services themselves and the capabilities that underlie services are all resources.

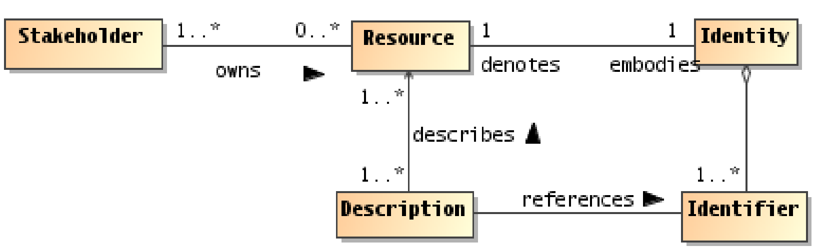


Figure 5 Resources model

Our model of resources is very simple, but is the foundation for modeling many of the things that a SOA-based system deals in such as information, services, capabilities, descriptions, policies and contracts.

Resource

> A resource is any entity of some perceived value, where the value may be in the function it performs or something intrinsic in its nature. may vary over time.

A resource has identity and it has an owner. A resource may have more than one identifier, but any well-formed identifier should unambiguously resolve to the intended resource.

An important class of resource is the class of capabilities that underlie services. For example, a light bulb is a resource that when activated gives off light; a book is a resource that when read allows one to gain knowledge from its content. Other examples of resources are services themselves, descriptions of entities (a kind of meta-resource), IT infrastructure elements used to deliver services, contracts and policies, and so on.

Identity

> Identity is the collection of individual characteristics by which a thing or person is recognized or known. In this architecture, we further restrict this to the collection of identifiers by which a person or thing is known.

Identity is an important, if abstract, concept. For example, in ensuring that a user is authenticated, the role of the authentication process is to validate the identity of the person that is attempting to gain access to a resource.

Identifier

> An identifier is any block of data – such as a string – that is associated with a particular identity.

It is good practice to use globally unique identifiers; for example globally unique IRIs. However, the primary requirement of an identifier is that it can be used to uniquely disambiguate the indicated resource from other resources.

This definition of resource is a simplification and elaboration of the concept that underlies the Web Architecture **[WA]**. Being more abstract, we do not require that the identity of a resource be in any particular form (although in practice, many resource identifiers are URIs), nor do we require resources to have representations. However, we do require resources to have owners.

3.2.1 Ownership Model

Understanding what it means to own something it important when we use an SOA-based system to exchange value. Ownership is also important in understanding the various kinds of obligations participants may enter into. Fundamentally, we view ownership as a relationship between a stakeholder and a resource, where the owner has certain rights over the resource (note not necessarily absolute rights).

Ownership

> Ownership is a relationship between an entity, a resource and a set of rights and responsibilities. When an entity owns a resource, the entity has the right to exercise the rights over the resource and may transfer ownership to another entity.
>
> In addition, owning a resource brings with it a set of responsibilities. The nature of these responsibilities will vary with the resource and the nature of the ownership; but typically, if the use of a resource harms someone, then the owner of the resource will be held responsible.

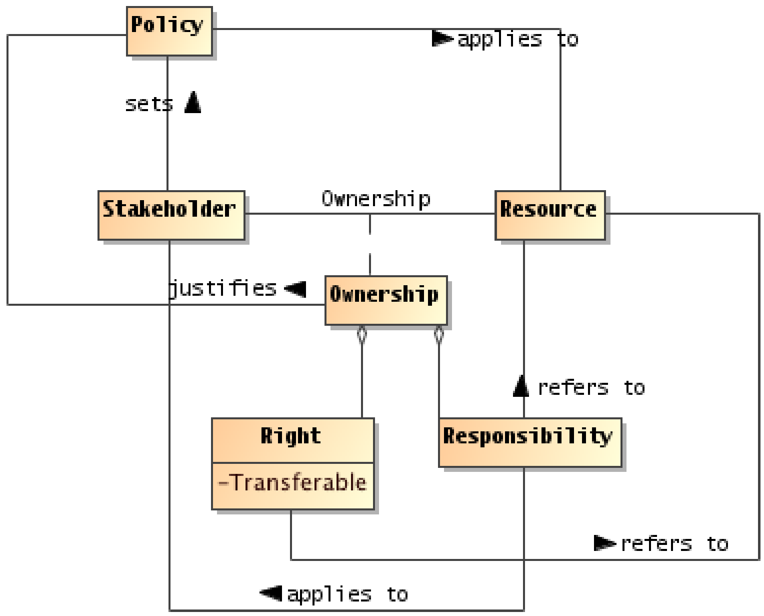


Figure 6 Resource Ownership Model

To own a resource implies taking responsibility for creating, maintaining, and if it is to be available to others, provisioning the resource. One who owns a resource may delegate any of these functions to others, but still has the responsibility to see the function is done. There may also be joint ownership of a resource, where the responsibility is shared.

Ownership is rarely absolute, rarely involves complete control over the resource. In reality, ownership is normally constrained to a particular set of rights. For example, one stakeholder may own the rights to deploy a capability as a service, another may own the rights to the profits that result from using the capability, and yet another may own the rights to use the service! However, a crucial property that distinguishes ownership from merely renting is the right to transfer ownership to another person or organization.

3.3 Needs and Capabilities Model

The motivation for participants interacting is the satisfaction of needs. From a consumer perspective, the motivation for interacting with a service is to satisfy a business objective, which in turn, is often related to the role they represent in the social structure; for the provider, the need is to gain satisfaction, monetary or otherwise, for other participants' use of the service.

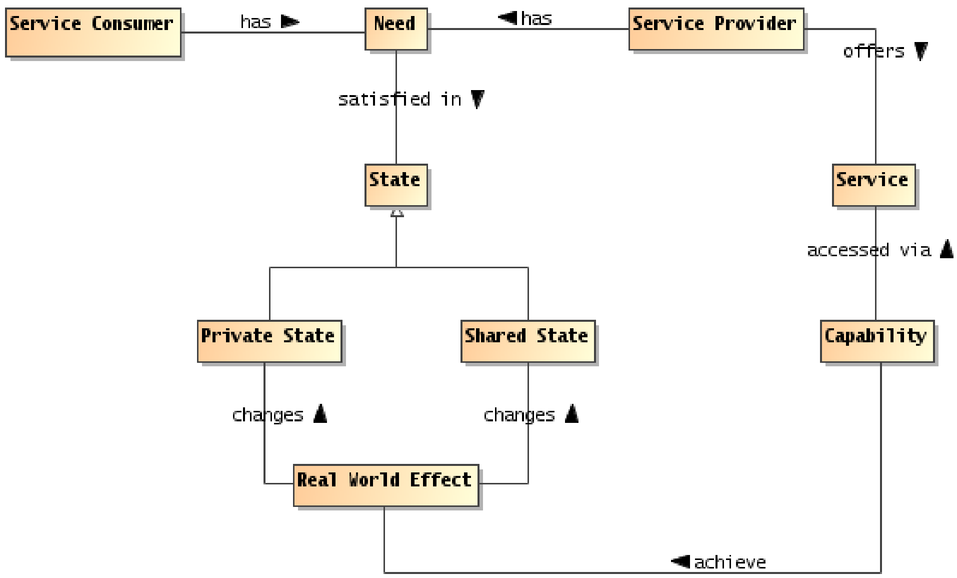


Figure 7 Needs and Capabilities

Capability

> A capability is a resource that may be used by a service provider to achieve a real world effect on behalf of a service consumer.

The model in Figure 7 show that there is an inherent indirection between needs and having them satisfied. Both needs and the effects of using capabilities are expressed in terms of state: a need is expressed as a condition on the desired state and the Real World Effect of using capabilities is a change in the state of the world.

As noted in the Reference Model, the Real World Effect is couched in terms of changes to the state that is shared by the participants in the service; in particular the public aspects of that state. In this Reference Architecture we further refine this notion in terms of changes in the social facts that are mandated by social structures – see Section 3.4.

By making a capability available for use, via the Service, the owners aim to address their needs as well as the needs of other participants who use the service. The extent to which a capability is exposed via a service (or via multiple services) is controlled by the owner of the capability.

Need

> A need is a measurable requirement that a service participant is actively seeking to satisfy.

A need may or may not be publicly measurable; the needs that this Reference Architecture finds in scope are those that are publicly measurable. However, the satisfaction of a participant's need can only be determined by that participant.

A need is characterized by a proposition – see Section 3.8. However, the extent to which a need is captured in a formal way is likely to be very different in each situation.

3.4 Social Structure Model

The actions undertaken by participants, whether mediated by services or in some other way, are normally performed in the context of a social context which defines the meaning of the actions themselves. We can formalize that context as a **social structure**: the embodiment of a particular social context.

The social structure model is important to defining and understanding the implications of crossing ownership boundaries; it is the foundation for an understanding of security in SOA and also provides the context for determining how SOA-based systems can be effectively managed and governed.

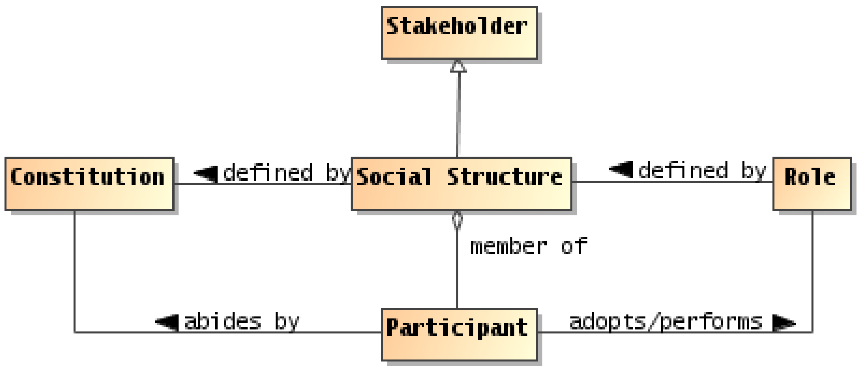


Figure 8 Social Structure

Social Structure

> A social structure (sometimes identified as social institutions) embodies some of the cultural aspects that characterize the relationships and actions among a group of participants.

In the Reference Architecture, we are concerned primarily with social structures that reflect the anticipated participants in SOA-based systems; these are often embodied in legal and quasi-legal frameworks; i.e., they have some rules that are commonly understood.

For example, a corporation is a common kind of social structure, as is a fishing club. At the other extreme, the legal frameworks of entire countries and regions also count as social structures.

It is not necessarily the case that the social structures involved in a service interaction are explicitly identified by the participants. For example, when a customer buys a book over the Internet, the social structure that defines the validity of the transaction is often the legal framework of the region associated with the book vendor. This legal jurisdiction qualification is typically buried in the fine print of the service description.

Constitution

> A constitution is an agreement shared by a group of participants that defines a social structure.

The primary purpose of the constitution is to define the roles of participants in the institution, and how to establish the regulations that define the legal actions. The regulations of the social structure effectively define how those assertions and commitments that are relevant to the social structure are created.

A constitution may be explicitly written down or it may be only partially written.

For example, a company's constitution is normally called the "Articles of Incorporation". A company's articles define the officers of the company, their rights and responsibilities and the purpose of the company. It will often also declare what the rules are for resolving conflicts.

A constitution is an agreement. It is abided to by the participants in the social structure. In some cases, this is based on an explicit agreement, in other cases participants behave as though they agree to the constitution without a formal agreement. For example, when a new employee joins a company, he or she is often required to sign an employment contract. That contract defines key aspects of the relationship between the new employee and the company. In other situations the act of agreement is less formal and less clearly established.

3.4.1 Shared State and social facts

Most of the actions performed by people and most of the important aspects of a person's state are inherently social in nature. The social context of an action is what gives it much of its meaning. We call actions in society social actions and those facts that are understood in a society social facts. It is often the case that social actions give rise to social facts.

Compared to facts about the natural world, social facts are inherently abstract: they only have meaning in the context of a social structure.

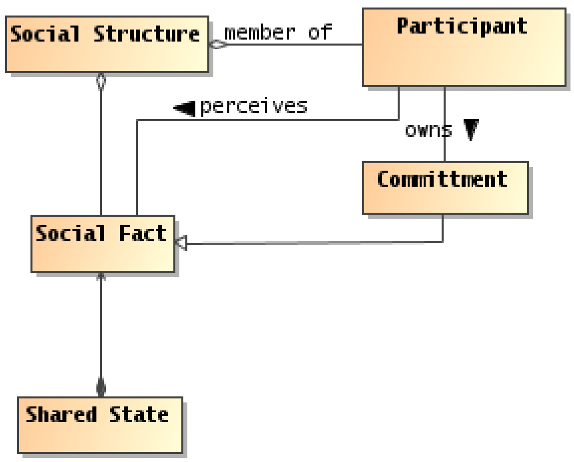


Figure 9 Shared State and Social Facts

Shared State

> The set of facts and commitments that manifest themselves to service participants as a result of interacting with a service.

Note that a participant has only a partial view of the shared state in a system. Furthermore, the participant will have internal state that is not accessible to other participants directly. However, elements of the shared state are in principle accessible to participants even if a given participant does not have access to all elements at any given time.

Social Fact

> A social fact is an element of the state of a social structure that is sanctioned by that social structure. For example, the existence of a valid purchase order with a particular customer has a meaning that is defined primarily by the company itself.

Social facts typically require some kind of ritual to establish: the action itself is physical, its interpretation is social. For example, the existence of an agreed contract typically requires both parties to sign papers and to exchange those papers. If the signatures are not performed correctly, or if the parties are not properly empowered to perform the ritual, then it is as though nothing happened.

In the case of agreements reached by electronic means, this involves the exchange of electronic messages; often with special tokens being exchanged in place of a hand-written signature.

For example, the hiring of a new employee is an action that is defined by the hiring company (and not, for example, by the president of another company). For a hiring to be valid, it is often the case that specific business processes must be followed, with key actions to be performed only by suitably authorized personnel (such as the manager of the hiring budget).

Commitment

> A commitment is a social fact about the future: in the future some fact will be true and a participant has the current responsibility of ensuring that that fact will indeed be true. A commitment to deliver some good is a classic example of a fact about the future.

Other important classes of social facts include the policies adopted by an organization, any agreements that it is holding for participants, and the assignment of participants to roles within the organization. The social facts that are understood in the context of a social structure define the shared state that is referenced in Figure 17.

Facts have the property of being verifiable (technically, a social fact can be verified to determine if it is satisfied in the social context). If, as a result of interacting with a service, a buyer incurs the obligation of paying for some good or service, this obligation (and the discharge of it) is measurable (perhaps by further interactions with the same or other services).

3.5 Acting in a Social Context

3.5.1 Actions, Real World Effect and Events

The most important concept in any model of actions and effects is that of **action** itself:

Action

Action is the application of intent by a participant (or agent) to achieve a real world effect.

This concept is simultaneously one of the fulcrums of the Service Oriented Architecture and a touch point for many other aspects of the architecture: such as policies, service descriptions, management, security and so on.

An action may have preconditions where a precondition is something that needs to be in place before an action can occur, e.g. confirmation of a precursor action. One important class of such preconditions are the conditions associated with security: authentication and authorization of the participants attempting actions.

Figure 10 shows a model of how actions are associated with agents that perform actions, the results of performing actions and how actions are associated with intention.

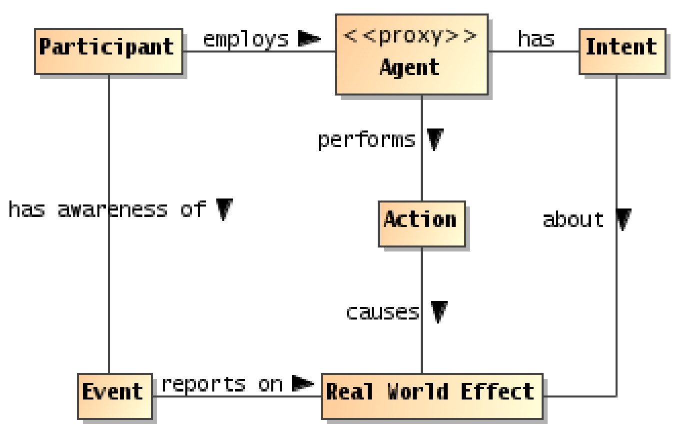


Figure 10 Actions, Real World Effect and Events Model

Real World Effect

A Real World Effect is the changes in the state of the world as a result of a participant performing an action in response to a service interaction.

The result of performing an action is, in the expected case, something changes in the world. This is the Real World Effect of performing the action. Many, if not most, instances of Real World Effect involve acting in the context of a social structure; i.e., the effect desired is the establishment of one of more social facts.

Changes in the world can be *reported* by means of events:

Event

An event is an occurrence that at least one participant has an interest in being aware of.

In the case of this Reference Architecture, a key class of events is that which reflects the effects of actions that have been performed – i.e., we are especially interested in events that report on Real World Effects of actions.

In effect, an event is the corollary to action: in a public arena, joint actions result in changes to the world; these changes are manifested as events that participants in the arena have an awareness of.

A key feature of action that distinguishes it from mere force or accident is that someone or something intended the action to occur. Intent represents an agent's relationship to one or more of its goals:

Intent

> Intent is the relationship between an agent and its goals that signifies a commitment by the agent to achieve that goal.

An agent's intent in performing an action is to further one or more of the agent's goals.

3.5.2 Social Actions

In the context of SOA, actions are primarily social in nature — one participant is asking another to do something — and goal oriented — the purpose of interacting with a service is to satisfy a need by attempting to ensure that a remote entity applies its capabilities to the need.

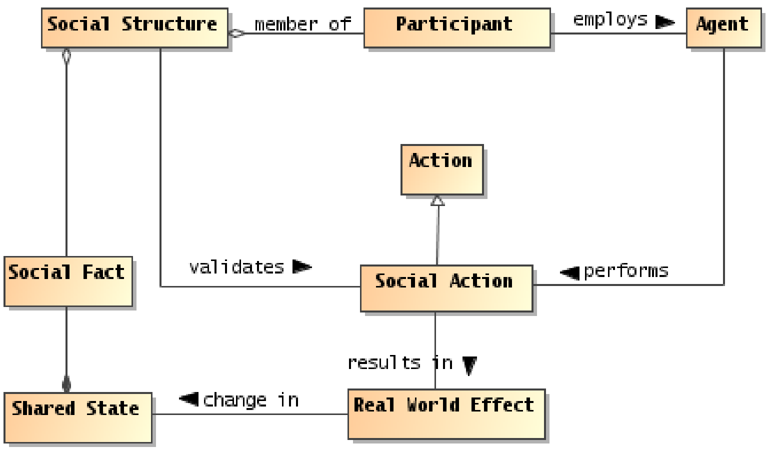


Figure 11 Acting within Social Structures

Social Action

> A social action is an action which is defined primarily by the effect it has on the relationship between participants and state of a social structure by establishing one or more new social facts. A social action consists of a physical action together with an appropriate authority.

Social actions are actions that are performed in order to achieve some result within a social structure.

Social actions are always contextualized by a social structure: the organization gives meaning to the action, and often defines the requirements for an action to be recognized as having an effect within the organization.

3.5.3 Interaction as Joint Action

When participants interact with services they are conducting actions that are inherently collaborative and joint in nature: there is no dance without a partner.

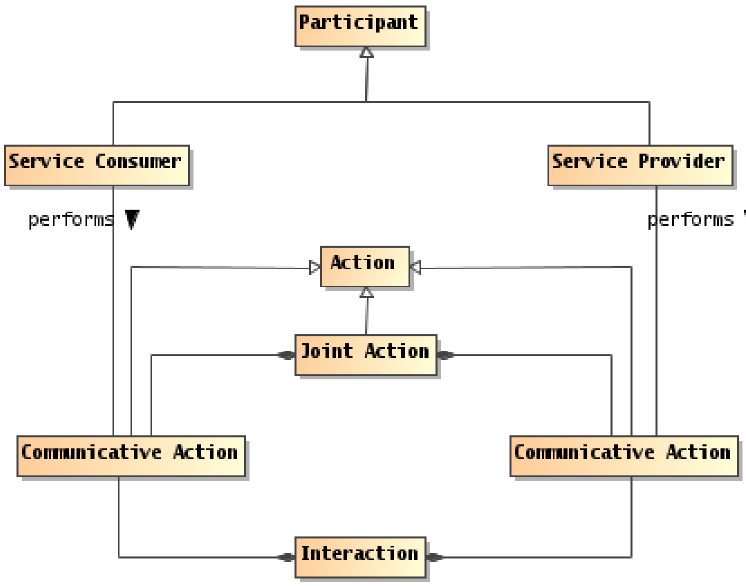


Figure 12 Service Interaction as Joint Action

Every action that Is part of an interaction between a service consumer and a service is inherently a *joint action* – involving both participants. Just as action is the foundation of an individual's actions in the context of SOA-based systems, interactions are characterized by joint actions:

Joint Action

> A joint action is an action involving the efforts of two or more participants to achieve a real world effect.

Joint actions are actions that inherently require two or more participants in order to properly relate the activities to the participants' intentions. Typically, a joint action involves two participants in communicative actions – one participant speaking and the other listening.

Joint actions are the foundation for understanding interaction between participants in a SOA-based system. It is not possible for there to be interaction between service providers and consumers without the participants engaging in a series of joint actions – typically joint communicative actions.

3.5.4 Semantics of Communication Model

Interaction is a form of communication. In this Reference Architecture, we use *messages* as the medium of interaction between service participants. Messages are exchanged that represent actions, and messages are exchanged that represent the reporting of events. In this model, we outline one way that this can be modeled effectively – in terms of shared vocabularies, shared semantics and shared understanding of communicated intent.

Since service consumers and providers are not directly acting against each other, they must do so indirectly – primarily by means of some form of communication. Speaking to someone is an action; if the speech conveys a request or a pronouncement of some kind, the former actions are used as vehicles to convey the true actions. Thus in Figure 13, we see **Action** appear twice – once in modeling the communicative actions needed to support interaction and once as the intended or conveyed action.

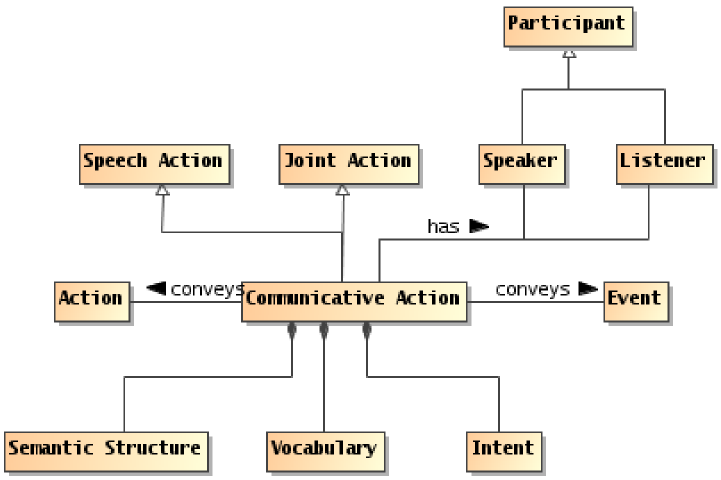


Figure 13 Semantics of Communication Model

Communicative Action

> Communicative actions are joint actions where service participants communicate with each other. A Communicative Action has a speaker and a Listener; each of whom must perform their part for the communicative action to occur.

Semantic Structure

> A communicative action has an aspect which conveys the meaning of the content being communicated. Typically, a semantic structure takes the form of a proposition which is either true, false or intended to be true or false.

The concept of semantic structure is quite abstract. However, in many cases involving machines, the semantic structure will be conveyed as some form of highly regular tree structure, with a well defined method for interpreting the structure. For example, an invoice will often follow pre-established standards for communicating invoices.

Intent

> The purpose of the communicative action is its **intent**. The intent, together with the semantic structure convey either an action – such as a request from a service consumer to the service – or an event – which typically reports on the results of previous communicative acts.

Vocabulary

> In order for there to be any communication, there must be sufficient shared understanding of the elements of interaction and of terms used in communication. A shared vocabulary may range from a simple understanding of particular strings as commands to a sophisticated collection of terms which are formalized in shared ontologies.

Note that while it is often easier to visualize the semantics of communication in terms that reflect human experience; it is not required for interactions between service consumers and providers to particularly look like human speech – it may be highly stylized in form, it may have particular forms and it may involve particular terms not found in human interaction.

However, any communication requires the core elements outlined in this model: some form of shared vocabulary, a shared basis for understanding communications, and a shared basis for establishing the intentions of participants.

3.5.5 Transactions and Exchanges Model

An important class of joint action is the **business transaction**, or **contract exchange**.

Business Transaction

> A business transaction is a joint action engaged in by two or more participants in which the real world effect is an increase in apparent value to the participants.

A classic business transaction is buying some good or service, but there is a huge variety of kinds of possible business transactions.

Key to the concept of business transaction is the contract or agreement to exchange. The form of the contract can vary from a simple handshake to an elaborately drawn contract with lawyers giving advice from all sides.

A completed transaction establishes a set of social facts relating to the exchange; typically to the changes of ownerships of the resources being exchanged.

Business Agreement

> A business agreement is an agreement entered into by two or more partners that constrains their future behaviors and permitted states. A business agreement is typically associated with business transactions: the transaction is guided by the agreement and an agreement can be the result of a transaction.

Business transactions often have a well defined life-cycle: a negotiation phase in which the terms of the transaction are discussed, an agreement action which establishes the commitment to the transaction, an action phase in which the agreed-upon items are exchanged (they may need to be manufactured before they can be exchanged), and a termination phase in which there may be long-term commitments by both parties but no particular actions required (e.g., if the exchanged goods are found to be defective, then there is likely a commitment to repair or replace them).

From an architectural perspective, the business transaction often represents the top-most mode of interpretation of service interactions. When participants interact in a service, they exchange information and perform actions that have an effect in the world. These exchanges can be interpreted as realizing part of, and in support of, business transactions.

Business Process

> A business process is a description of the tasks, participants' roles and information needed to fulfill a business objective.

Business processes are often used to describe the actions and interactions that form business transactions. This is most clear when the business process defines an activity involving parties external to the organization; however, even within an enterprise, a business process typically involves multiple participants and stakeholders.

In the context of transactions mediated and supported by electronic means, business processes are often required to be defined well enough to permit automation. The forms of such definitions are often referred to as choreographies:

Process Choreography

> The description of the possible interactions that may take place between two or more participants to fulfill an objective.

A choreography is, in effect, a description of what the forms of permitted joint actions are when trying to achieve a particular result. Joint actions are by nature formed out of the individual actions of the participants; a choreography can be used to describe those interlocking actions that make up the joint action itself.

3.6 Roles in Social Structures

One of the primary benefits of formalizing the relationships between people in terms of groups, corporations, legal entities and so on, is that it allows greater efficiencies in the operation of society. However, corporations, governments and even society, are abstractions: a government is not a person that can perform actions -- only people can actually do things.

For example, a fishing club is an abstraction that is important to its members. A club, however, is an abstraction that has no physical ability to act in the world. On the other hand, a person who is appropriately empowered by the fishing club can act. For example, when that person writes a check and mails it to the telephone company, that action counts as though the fishing club has paid its bills.

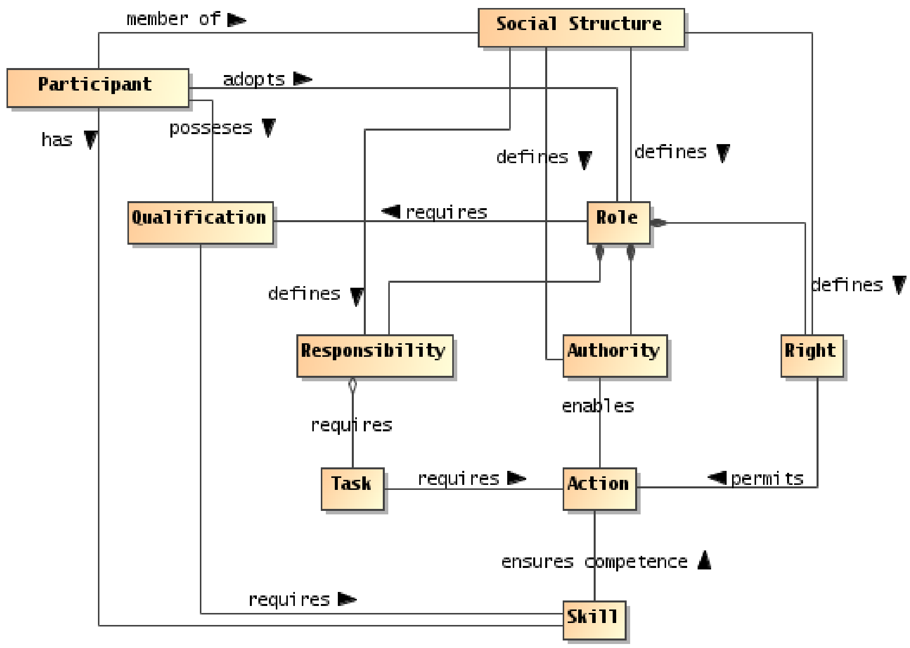


Figure 14 Roles, Rights and Responsibilities Model

Participants' actions within a social structure are often defined by the roles that they adopt.

Role

> A role is an identified relationship between a participant and a social structure that defines the rights, responsibilities, qualifications, and authorities of that participant within the context of the social structure.

For many scenarios, the roles of participants are easily identified: for example, a buyer uses the service offered by the seller to achieve a purchase. However, in particular in situations involving delegation, the role of a participant may be considerably more complex.

A participant may adopt one or more roles; and have zero or more skills and qualifications. For example, a participant adopting the role of secretary of a standards group is obliged to ensure that all the minutes of the various meetings are properly recorded; and members of certain standards groups are obliged to declare any pre-existing IP claims that may be relevant to the work of the groups.

Note that, while many roles are clearly identified, with appropriate names and definitions of the responsibilities, it is also entirely possible to separately bestow rights, responsibilities and so on; usually in a temporary fashion. For example, when a CEO delegates the responsibility of ensuring that the company accounts are correct to the CTO, this does not imply that the CTO is adopting the full role of CFO.

In order for a person to act on behalf of some other person or on behalf of some legal entity, it is required that they have the power to do so and the authority to do so.

Rights, authorities, responsibilities and roles form the foundation for the security architecture of the Reference Architecture. Rights and responsibilities have similar structure to permissive and obligation policies; except that the focus is from the perspective of the constrained participant rather than the constrained actions.

Right

> A right is a predetermined permission that permits an agent to perform some action or adopt a stance in relation to the social structure and other agents. For example, in most circumstances, sellers have a right to refuse service to potential customers; but may only do so based on certain criteria.

Authority

> The right to act as agent on behalf of an organization or another person. Usually, this is constrained in terms of the kinds of actions that are authorized, and in terms of the necessary skills and qualifications of the persons invoking the authority.

An entity may authorize or be assigned another entity to act as its agent. Often the actions that are so authorized are restricted in some sense. In the case of human organizations, the only way that they can act is via an agent.

Responsibility

> A responsibility is an obligation on a role player to perform some action or to adopt a stance in relation to other role players.

Skill

> A skill is a competence or capability to achieve some real world effect. Skills are typically associated with roles in terms of requirements: a given role description may require that the role player has a certain skill.

Qualification

> A qualification is a public determination by an issuing authority that a stakeholder has achieved some state. The issuing authority may require some successful actions on the part of the stakeholder (such as demonstrating some skills). The qualification may have constraints attached to it; for example, the certification may be time limited.

There is a distinction between a skill – which is capability that a participant may have to act – and a publicly accepted right to act. For example, someone may have the skills to fly an airplane but not have a pilot's license. Conversely, someone may have a pilot license, but because of some temporary cause be incapable of flying a plane (they may be ill for example).

Qualifications are often used as constraints on roles: any entity adopting a role within an organization (or other social structure) must have certain qualifications.

3.7 Governance and Social Structures

Given that SOA mediates an important aspect of people's relationships, it follows that there are commitments entered into by participants that require enforcement by the community and that the SOA itself must reflect the requirements of the community itself.

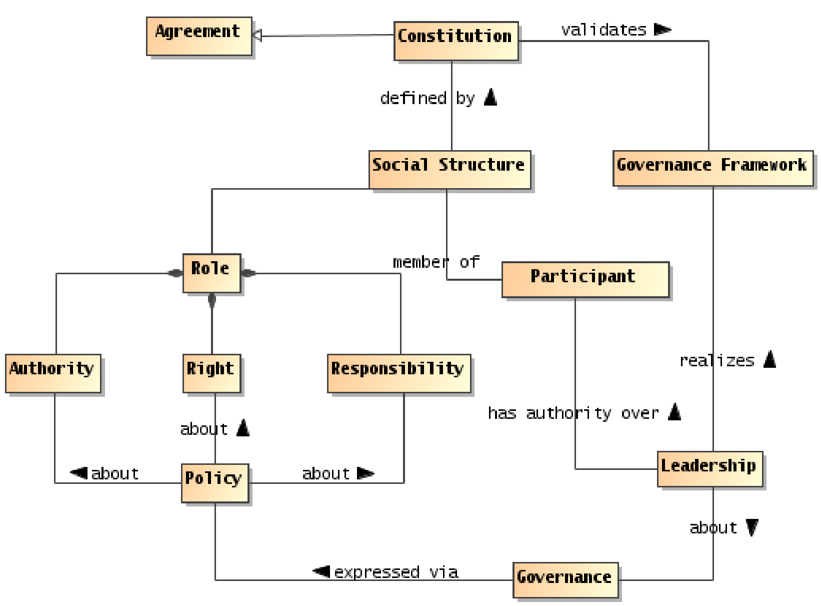


Figure 15 Social Structures and Governance

Both of these are aspects of the governance of Service Oriented Architecture.

The key elements of our model that relate to governance are the constitution of the social structure, the policies of the social structure, authority in a social structure, and the associated mechanisms of enforcement.

With few exceptions, social structures are embedded in other social structures. One result of this is that the institution's constitution is often viewable as a social fact in one or more outer social structures. For example, the Articles of Incorporation of a company is considered a legal document that supports the legal fact of existence of the company — by the legal jurisdiction of the company.

The main exception to this is, of course, the agreement that defines the constitution of a country. Notably, for most people who are born into the country, its constitution is one that they often do not explicitly agree to. However, it is universal for people who are naturalizing their citizenship to be required to explicitly agree to the constitution of their new country.

3.8 Proposition Model

The Reference Architecture makes use of descriptions of entities and states in the world. For example, we talk about a need being satisfied in Section 3.3, a policy being enforced in Section 4.4 a service description in Section 4.1.

In order to be able to relate a description with the entity that it being described we need the description to be verifiable relative to the entity. The proposition model identifies the key components that can support the verifiability of descriptions.

Proposition

> A proposition is an expression, normally in a language that has a well-defined written form, that expresses some property of the world from the perspective of a stakeholder.

In principle, the truth of a proposition must be verifiable – using a decision procedure – by examining the world and checking that the proposition and the world are consistent with each other.[10]

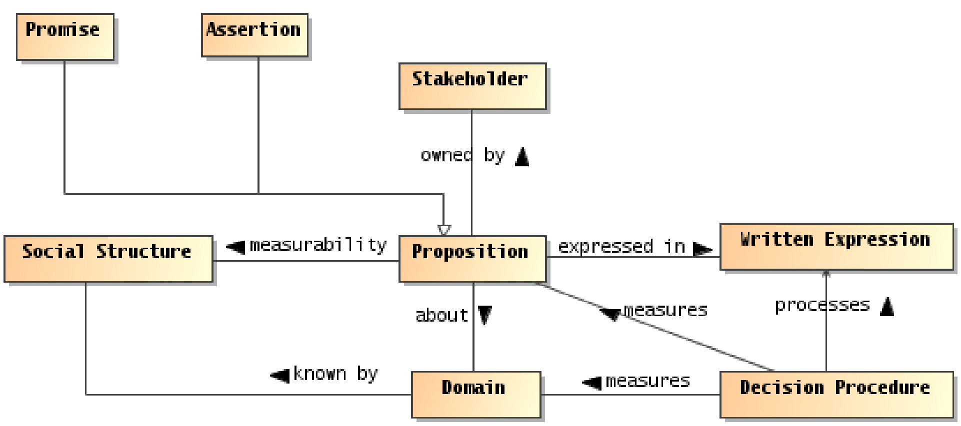


Figure 16 Propositions

Decision Procedure

> A process for determining whether an expression is true, or is satisfied, in the world.

Decision procedures are algorithms, programs that can measure the world against a formula, expression or description and answer the question whether the world corresponds to the description. If the truth of a proposition is indeterminable, then a decision procedure does not exist, and the logic is undecidable.

[10] We exclude here the special case of proposition known as a tautology. Tautologies are important in the study of logic; the kinds of propositions that we are primarily interested in are those which pertain to the world; and as such are only *contingently* true.

When we say ‘world’, we are not restricted to the physical world. The criterion is an ability to discover facts about it. In our case governmental, commercial and social structures that form the backdrop for SOA-based systems are important examples of modeled worlds.

Note that not all description languages have a decision procedure. However, for the uses to which we put the concept of proposition: policies, service descriptions, and so on, we require that the descriptive language have a decision procedure.

Propositions, as used in reference to needs, policies and contracts can be further analyzed in terms of facts that are about the world as it is, will be, or should be. The latter are particularly of concern in policies and contracts and other propositions concerning the relationships between people.

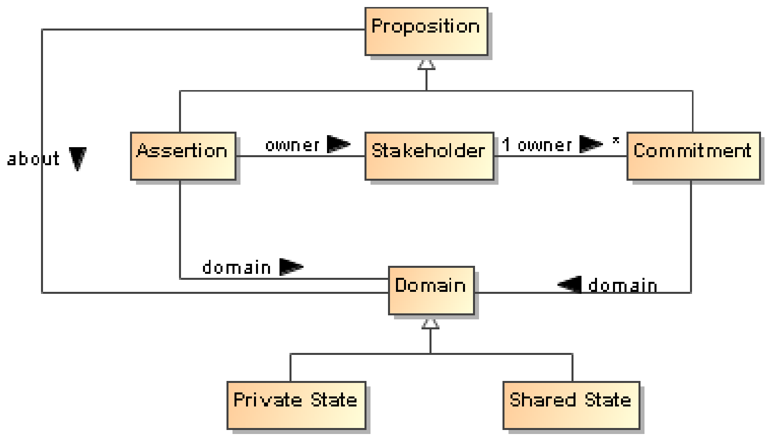


Figure 17 Assertions and Promises

Assertion

An assertion is a proposition that is held to be true by a stakeholder. It is essentially a claim about the state of the world.

Promise

A promise is a proposition regarding the future state of the world by a stakeholder. In particular, it represents a commitment by the stakeholder to ensure the truth of the proposition.

For example, an airline may report its record in on-time departures for its various flights. This is a claim made by the airline which is, in principle, verifiable. The same airline may promise that some percentage of its flights depart within 5 minutes of their scheduled departure. The truth of this promise depends on the effectiveness of the airline in meeting its commitments.

Another way of contrasting assertions and promises is to see what happens when the propositions fail: a stakeholder that makes a false assertion about the world might be classified as a liar; a stakeholder that makes a false promise is said to break its promises.

4 Realizing Service Oriented Architectures View

Make everything as simple as possible but no simpler.
Albert Einstein

The *Realizing Service Oriented Architectures View* focuses on the infrastructure elements that are needed in order to support the discovery and interaction with services. The key questions asked are "What are services, what support is needed and how are they realized?"

The models in this view include the Service Description Model, the Service Visibility Model, the Interacting with Services Model, the Realization of Policies Model, and the Policies and Contracts Model.

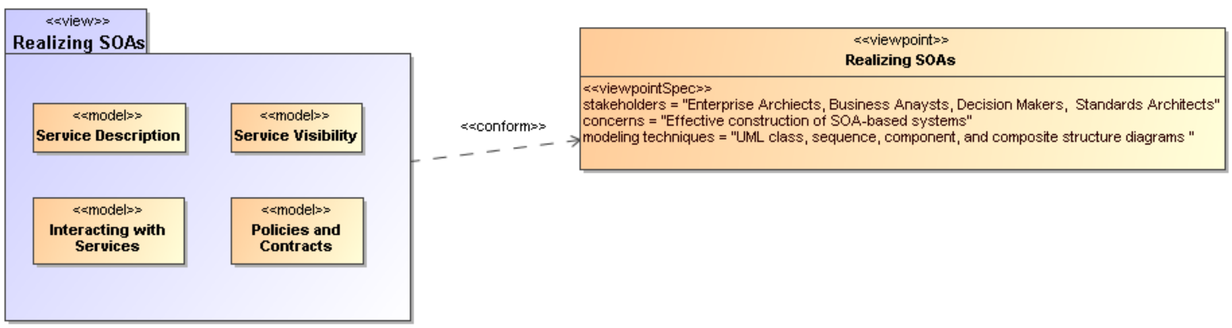


Figure 18 Model Elements Described in the Realizing a Service Oriented Architecture View

4.1 Service Description Model

A service description is an artifact, usually document-based, that defines or references the information needed to use, deploy, manage and otherwise control a service. This includes not only the information and behavior models associated with a service to define the service interface but also includes information needed to decide whether the service is appropriate for the current needs of the service consumer. Thus, the service description will also include information such as service reachability, service functionality, and the policies and contracts associated with a service.

A service description artifact may be a single document or it may be an interlinked set of documents. For the purposes of this model, differences in representation are to be ignored, but the implications of a "web of documents" is discussed later in this section.

There are several points to note regarding the following discussion of service description:

- SOA-RM states that one of the hallmarks of SOA is the large amount of associated description. The model presented below focuses on the description of services but it is equally important to consider the descriptions of the consumer, other participants, and needed resources other than services.
- Descriptions are inherently incomplete but may be determined as *sufficient* when it is possible for the participants to access and use the described services based only on the descriptions provided. This means that, at one end of the spectrum, a description along the lines of "*That service on that machine*" may be sufficient for the intended audience. On the other extreme, a service description with a machine-process-able description of the semantics of its operations and real world effect may be required for services accessed via automated service discovery and planning systems.
- Descriptions will change over time as, for example, the ingredients and nutrition information for food labeling continues to evolve. A requirement for transparency of transactions may require additional description for those associated contexts.
- Description always proceeds from a basis of what is considered "common knowledge". This may be social conventions that are commonly expected or possibly codified in law. It is impossible to describe everything and it can be expected that a mechanism as far reaching as SOA will also connect entities where there is inconsistent "common" knowledge.

- Descriptions will become the collection point of information related to a service or any other resource, but it will not necessarily be the originating point or the motivation for generating this information. In particular, given a SOA service as the access to an underlying capability, the service may point to some of the capability's previously generated description, e.g. a service providing access to a data store may reference update records that indicate the freshness of the data. As another example, it is more maintainable for description to reference the information maintained by an individual who is designated a Responsible Party (see Section 3.2.1) than to require the update of every instance where the individual is so designated.
- Descriptions of the provider and consumer are the essential building blocks for establishing the execution context of an interaction.

These points emphasize that descriptions are assembled with respect to some context and there is no one "right" description for all contexts and for all time. Several descriptions for the same subject may exist at the same time, and this emphasizes the importance of the description referencing source material maintained by that material's owner rather than having multiple copies that become out of synch and inconsistent.

It may also prove useful for a description assembled for one context to cross-reference description assembled for another context as a way of referencing ancillary information without overburdening any single description. Rather than a single artifact, description can be thought of as a web of documents that enhance the total available description.

This Reference Architecture uses the term service description for consistency with the concept defined in SOA-RM. Some of the current SOA literature speaks to the idea of a "service contract" as effectively the equivalent, although the details of what comprises the service description/contract may vary. The term service description is preferred because policies are an element of description for any resource and the agreement on policies between service participants may be thought of as a contract. Saying service contract for the service description implies just one side of the interaction is governing and misses the point that a single set of policies identified by a service description may lead to numerous contracts, i.e. service level agreements, leveraging the same description. Indeed, these agreements establish the execution context of the service interaction and are not a fundamental attribute of the service itself.

4.1.1 The Model for Service Description

Figure 20 shows Service Description modeled as a subclass of the general Description class, where Description is a subclass of the Resource class as defined in section 3.2. In addition, each Resource is assumed to have a description. The following section discusses the relationships among elements of general description and the subsequent sections focus on service description itself. Note, other descriptions, such as those of participants, are important to SOA but are not individually elaborated in this document.

4.1.1.1 Model Elements Common to General Description

The general Description class is composed of a number of elements that are expected to be common among all specialized descriptions supporting a service oriented architecture.

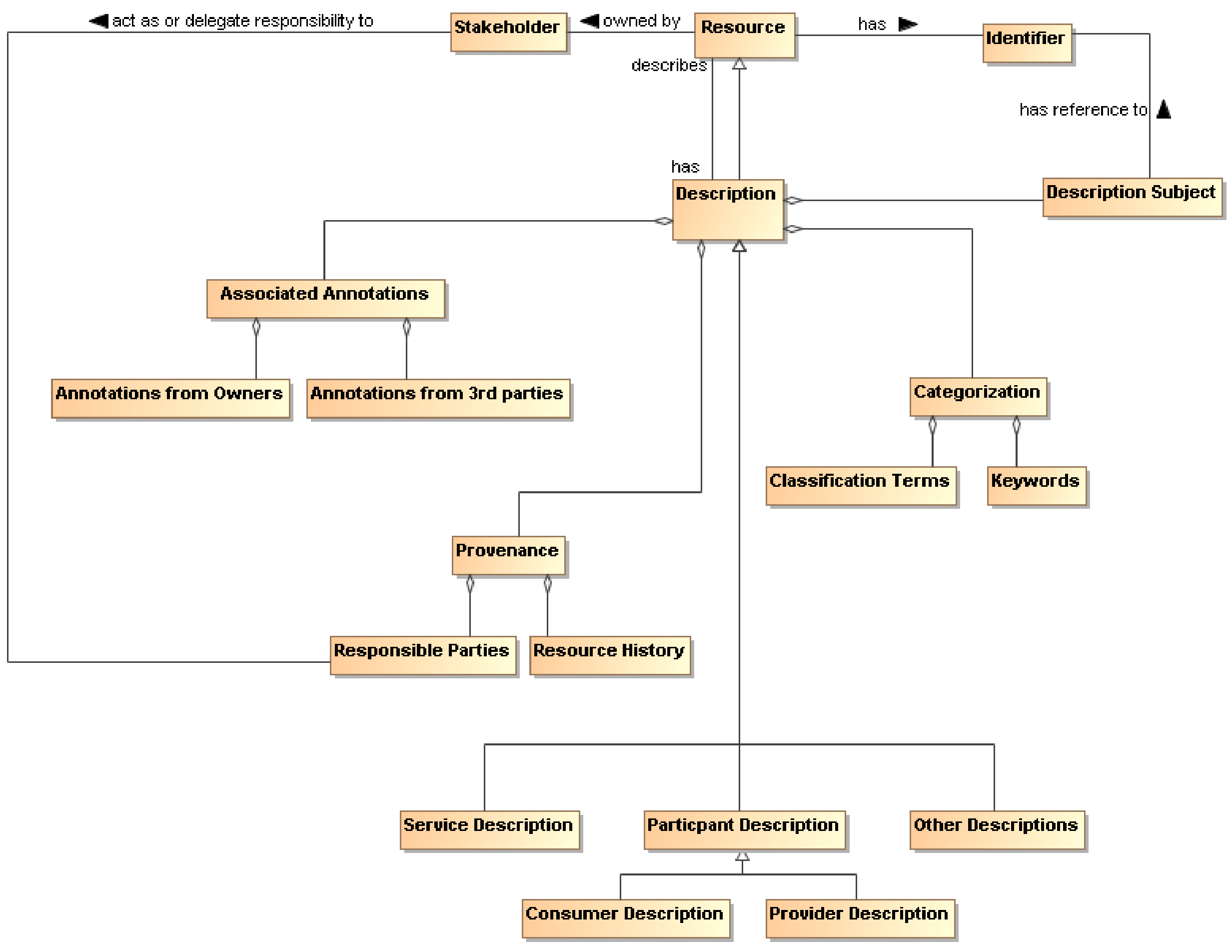


Figure 19 General Description Model

4.1.1.1.1 Description Subject

The subject of a description is a Resource. The value assigned to the Description Subject class may be of any form that provides understanding of what constitutes the Resource, but it is often in human-readable text. The Description Subject MUST also reference the Resource Identifier of the resource it describes so it can unambiguously identify the subject of each description instance.

As a Resource, Description also has an identifier with a unique value for each description instance. The description instance provides vital information needed to both establish visibility of the resource and to support its use in the execution context for the subsequent interaction. The identifier of the description instance allows the description itself to be referenced for discussion, access, or reuse of its content. While some subset of the description instance may be entered in a registry to support mediated discovery of the description subject, the entire description instance will provide the more complete description needed to initiate and continue interaction with the subject.

4.1.1.1.2 Provenance

While the Resource Identifier provides the means to know which subject and subject description are being considered, Provenance as related to the Description class provides information that reflects on the quality or usability of the subject. Provenance specifically identifies the entity (human, defined role, organization, ...) that assumes responsibility for the resource being described and tracks historic information that establishes a context for understanding what the resource provides and how it has changed over time. Responsibilities may be directly assumed by the Shareholder who owns a Resource or the Owner may designate Responsible Parties for the various aspects of maintaining the resource and provisioning it for use by others. There may be more than one entity identified under Responsible Parties;

for example, one entity may be responsible for code maintenance while another is responsible for provisioning of the executable code. The historical aspects may also have multiple entries, such as when and how data was collected and when and how it was subsequently processed, and as with other elements of description, may provide links to other assets maintained by the Resource owner.

4.1.1.1.3 Keywords and Classification Terms

A traditional element of description has been to associate the resource being described with predefined keywords or classification taxonomies that derive from referenceable formal definitions and vocabularies. This Reference Architecture does not prescribe which vocabularies or taxonomies may be referenced, nor does it limit the number of keywords or classifications that may be associated with the resource. It does, however, state that a normative definition SHOULD be referenced, whether that be a representation in a formal ontology language, a pointer to an online dictionary, or any other accessible source. See Section 4.1.2.1 for further discussion on associating semantics with assigned values.

4.1.1.1.4 Associated Annotations

The general description instance may also reference associated documentation that is in addition to that considered necessary in this model. For example, the owner of a service may have documentation on best practices for using the service. Alternately, a third party may certify a service based on their own criteria and certification process; this may be vital information to other prospective consumers if they were willing to accept the certification in lieu of having to perform another certification themselves. Note, while the examples of Associated Documentation presented here are related to services, the concept applies equally to description of other entities.

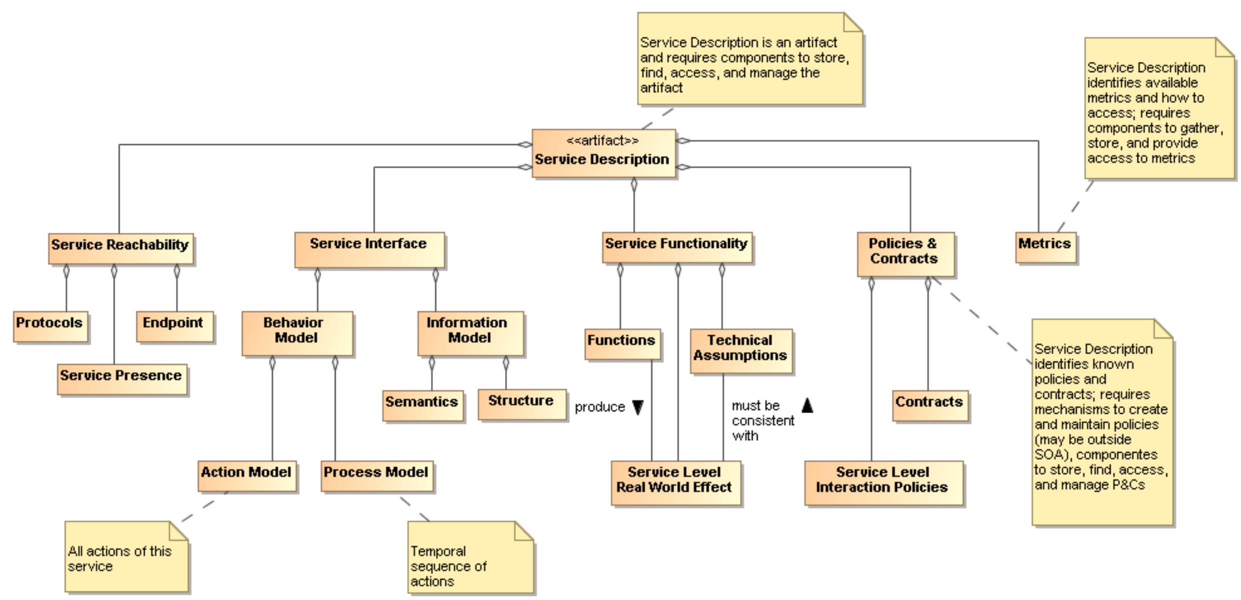


Figure 20 Service Description Model

4.1.1.2 Model Elements Specific to Service Description

The major elements for the Service Description subclass follow directly from the areas discussed in the Reference Model. Here, we discuss the detail shown in *Figure 20* and the purpose served by each element of service description.

4.1.1.2.1 Service Interface

As noted in the Reference Model, the service interface is the means for interacting with a service. For this reference architecture and as shown in Section 4.3 the service interface will support an exchange of messages, where

- the message conforms to a referenceable message exchange pattern (MEP),
- the message payload conforms to the structure and semantics of the indicated information model,
- the messages are used to invoke actions against the service, where the actions are specified in the action model and any required sequencing of actions is specified in the process model.

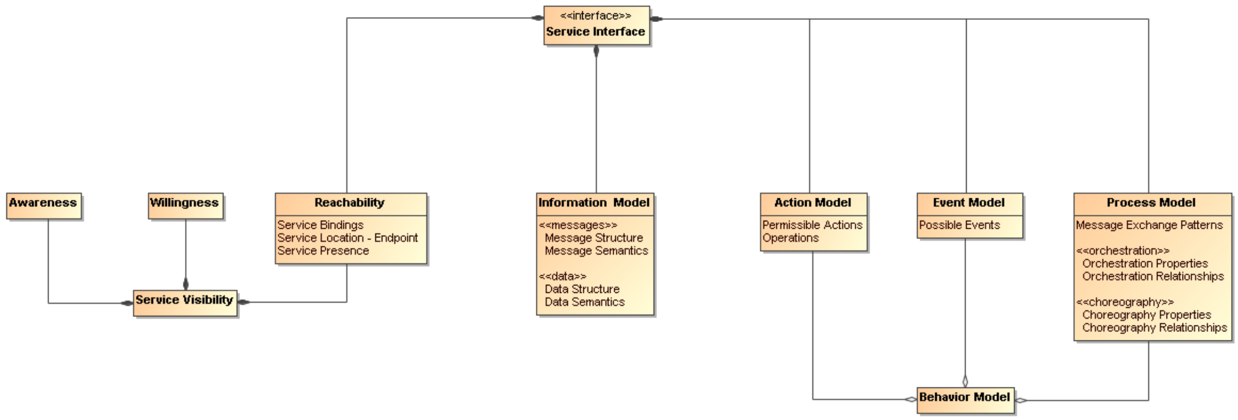


Figure 21 Service Interface Model

These aspects of messages are discussed in more detail in Section 4.3

4.1.1.2.2 Service Reachability

Service reachability, as modeled in Section 4.2.3 enables service participants to locate and interact with one another. To support service reachability, the service description should indicate the endpoints to which a service consumer can direct messages to invoke actions and the protocol to be used for message exchange using that endpoint.

In the present context, an endpoint is a referenceable entity, processor, or resource against which one can perform an action.[11] As applied in general to an action, the endpoint is the conceptual location where one applies an action; with respect to service description, it is the actual address where a message is sent.

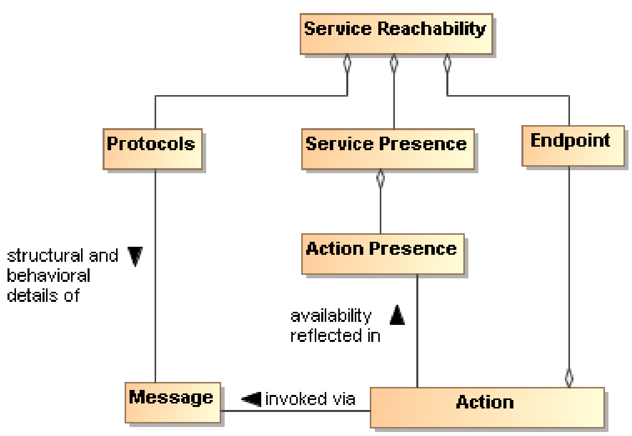


Figure 22 Service Reachability model

In addition, the service description should provide information on service presence or on a means of establishing this presence. Presence for either an action or a service may include a static representation

[11] This definition of endpoint is consistent with WS-Addressing (http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/) but generalized for any action, not exclusively those implemented as Web Services.

of availability or there may be a dynamic means to assess the current availability. The relationship between service presence and the presence of the individual actions that can be invoked is discussed under Establishing Reachability in Section 4.2.3.3.

4.1.1.2.3 Service Functionality

While the service interface and service reachability are concerned with the mechanics of using a service, service functionality and performance metrics (discussed in the next section) describe what can be expected when interacting with a service. Service Functionality, shown in *Figure 20* as part of the overall Service Description model, is an unambiguous expression of service function(s) and the real world effects of invoking the function. The Functions likely represent business activities in some domain that produce the desired Real World Effects.

The Service Functionality may also be constrained by Technical Assumptions that underlie the effects that can result. Technical assumptions are defined as domain specific restrictions and may express underlying physical limitations, such as flow speeds must be below sonic velocity or disk access that cannot be faster than the maximum for its host drive. Technical assumptions are likely related to the underlying capability accessed by the service. In any case, the Real World Effects must be consistent with the Technical Assumptions.

Elements of Service Functionality may be expressed as natural language text, reference to an existing taxonomy of functions, or reference to a more formal knowledge capture providing richer description and context.

4.1.1.2.4 Policies and Contracts, Metrics, and Compliance Records

Policies prescribe the conditions and constraints for interacting with a service and impact the willingness to continue visibility with the other participants. Whereas technical assumptions are statements of "physical" fact, policies are subjective assertions made by the service provider (sometimes as passed on from higher authorities).

The service description provides a central location for identifying what policies have been asserted by the service provider. The specific representation of the policy, e.g. in some formal policy language, is likely done outside of the service description and the service description would reference the normative definition of the policy.

Policies may also be asserted by other service participants, as illustrated by the model shown in Figure 23. Policies that are generally applicable to any interaction with the service are likely to be asserted by the service provider and included in the Policies and Contracts section of the service description. Conversely, policies that are asserted by specific consumers or consumer communities would likely be identified as part of a description's Annotations from 3rd parties (see section 4.1.1.1.4) because these would be specific to those parties and not a general aspect of the service being described.

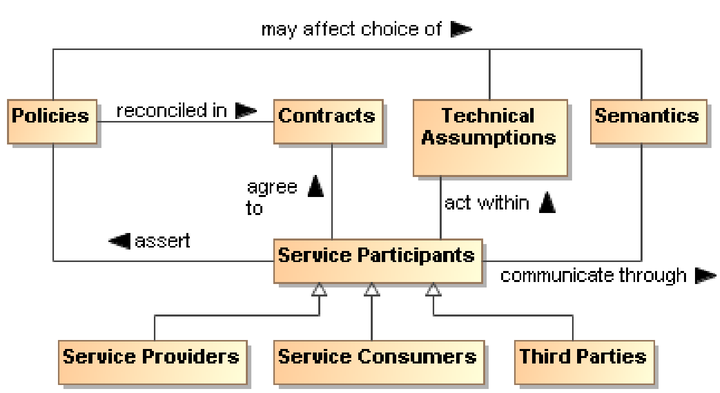


Figure 23 Model for Policies and Contracts as related to Service Participants

As noted in the model in Figure 23 the policies asserted may affect the allowable Technical Assumptions that can be embodied in services or their underlying capabilities and may affect the semantics that can be used. For example of the former, there may be a policy that specifies the surge capacity to be accommodated by a server, and a service that designs for a smaller capacity would not be appropriate to use. For the latter, a policy may require that only services using a community-sponsored vocabulary can be used.

Contracts are agreements among the service participants. The contract may reconcile inconsistent policies asserted by the participants or may specify details of the interaction. Service level agreements (SLAs) are one commonly used category of contracts.

References to contracts under which the service can be used may also be included in the service description. As with policies, the specific representation of the contract, e.g. in some formal contract language, is likely done outside of the service description and the service description would reference the normative definition of the contract. Policies and contracts are discussed further in Section 4.4.

The definition and later enforcement of policies and contracts are predicated on the existence of metrics; the relationships among the relevant concepts are shown in the model in Figure 24. Performance Metrics identify quantities that characterize the speed and quality of realizing the real world effects produced via the SOA service; in addition, policies and contracts may depend on nonperformance metrics, such as whether a license is in place to use the service. Some of these metrics reflect the underlying capability, e.g. a SOA service cannot respond in two seconds if the underlying capability is expected to take five seconds to do its processing; some metrics reflect the implementation of the SOA service, e.g. what level of caching is present to minimize data access requests across the network.

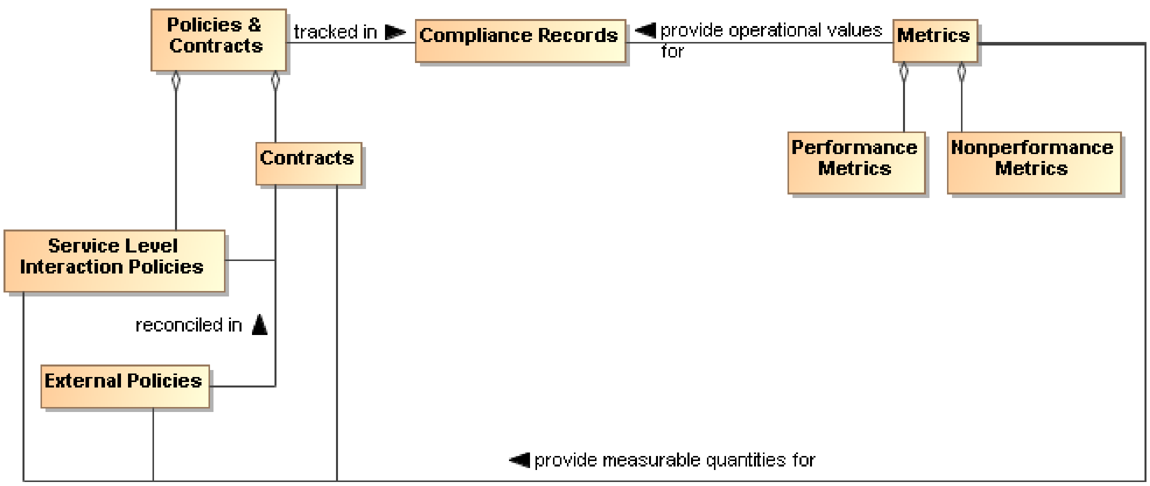


Figure 24 Model relating Policies and Contracts, Metrics, and Compliance Records

As with many quantities, the actual performance metrics are not themselves defined by this Service Description because it is not known *a priori* which metrics are being collected by the services, the SOA infrastructure, or other resources that participate in the SOA interactions. However, the service description SHOULD provide a placeholder (possibly through a link to an externally compiled list) for identifying which metrics are available and how these can be accessed.

The use of metrics to evaluate compliance is discussed in Section 4.4. The results of compliance evaluation SHOULD be maintained in compliance records and the means to access the compliance records SHOULD be included in the Policies and Contracts portion of the service description.

Note, even though policies are from the perspective of a single participant, policy compliance can be measured and policies may be enforceable even if there is not contractual agreement with other participants. This should be reflected in the policy, contract, and compliance record information maintained in the service description.

4.1.2 Use Of Service Description

4.1.2.1 Assigning Values to Description Instances

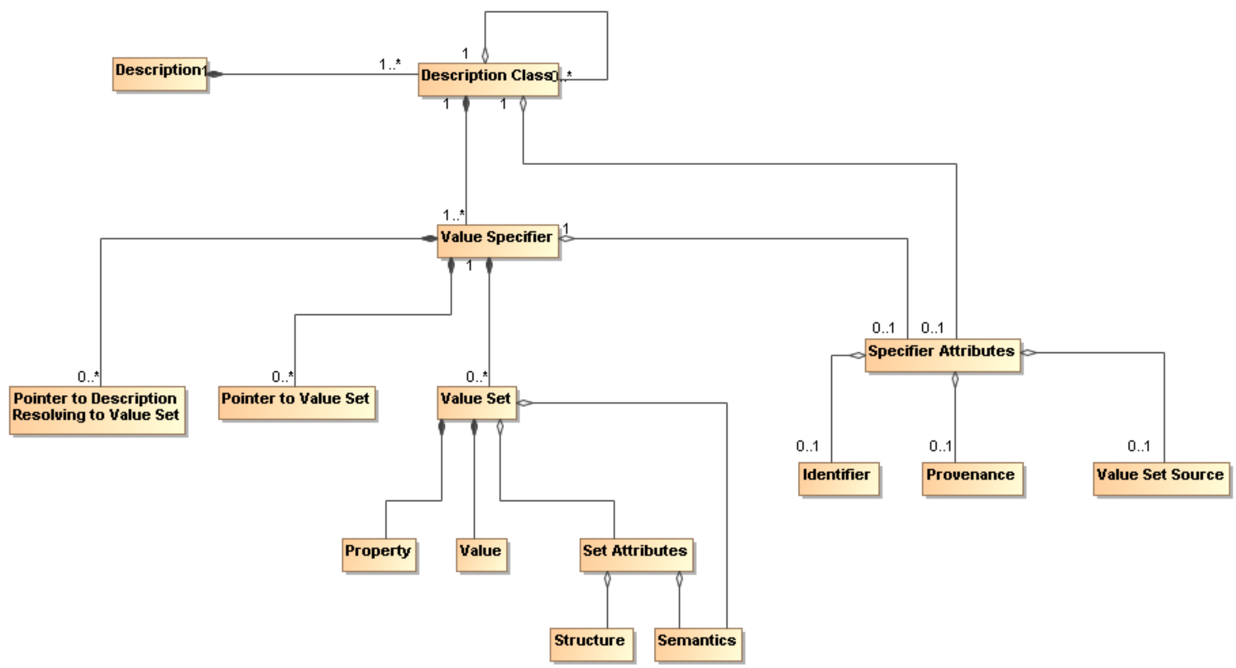


Figure 25 Representation of a Description Class

shows the template for a general description but individual description instances depend on the ability to associate meaningful values with the identified elements. Figure 25 shows a model for a collection of information that provides for value assignment and traceability for both the value meaning and the source of a value. The model is not meant to replace existing or future schema or other structures that have or will be defined for specific implementations, but it is meant as guidance for the information such structures need to capture to generate sufficient description. It is expected that tools will be developed to assist the user in populating description and autofilling many of these fields, and in that context, this model provides guidance to the tool developers.

For the model in Figure 25, each class is represented by a value specifier or is made up by components that will eventually resolve to a value specifier. For example, Description has several components, one of which is Categorization, which would be represented by a value specifier.

A value specifier consists of

- a collection of value sets with associated property-value pairs, pointers to such value sets, or pointers to descriptions that eventually resolve to value sets that describe the component; and
- attributes that qualify the value specifier and the value sets it contains.

The qualifying attributes for the value specifier include

- an optional identifier that would allow the value set to be defined, accessed, and reused elsewhere;
- provenance information that identifies the party (individual, role, or organization) that has responsibility for assigning the value sets to any description component;
- an optional source of the value set, if appropriate and meaningful, e.g. if a particular data source is mandated.

If the value specifier is contained within a higher-level component, (such as Service Description containing Service Functionality), the component may inherit values for the attributes from its container.

Note, provenance as a qualifying attribute of a value specifier is different from provenance as part of an instance of Description. Provenance for a service identifies those who own and are responsible for the service, as described in Section 3.2.1. Provenance for a value specifier identifies who is responsible for choosing and assigning values to the value sets that comprise the value specifier. It is assumed that granularity at the value specifier level is sufficient and provenance is not required for each value set.

The value set also has attributes that define its structure and semantics.

- The semantics of the value set property should be associated with a semantic model conveying the meaning of the property within the context for use, where the semantic model could vary from a free text definition to a formal ontology.
- For numeric values, the structure would provide the numeric format of the value and the “semantics” would be conveyed by a dimensional unit with an identifier to an authoritative source defining the dimensional unit and preferred mechanisms for its conversion to other dimensional units of like type.
- For nonnumeric values, the structure would provide the data structure for the value representation and the semantics would be an associated semantic model.
- For pointers, architectural guidelines would define the preferred addressing scheme.

The value specifier may indicate a default semantic model for its component value sets and the individual value sets may provide an override.

The property-value pair construct is introduced for the value set to emphasize the need to identify unambiguously both what is being specified and what is a consistent associated value. The further qualifying of Structure and Semantics in the Set Attributes allows for flexibility in defining the form of the associated values.

4.1.2.2 Service Description in support of Service Interaction

If we assume we have awareness, i.e. access to relevant descriptions, the service participants must still establish willingness and presence to ensure full visibility (See Section 4.2) and to interact with the service. Service description provides necessary information for many aspects of preparing for and carrying through with interaction.

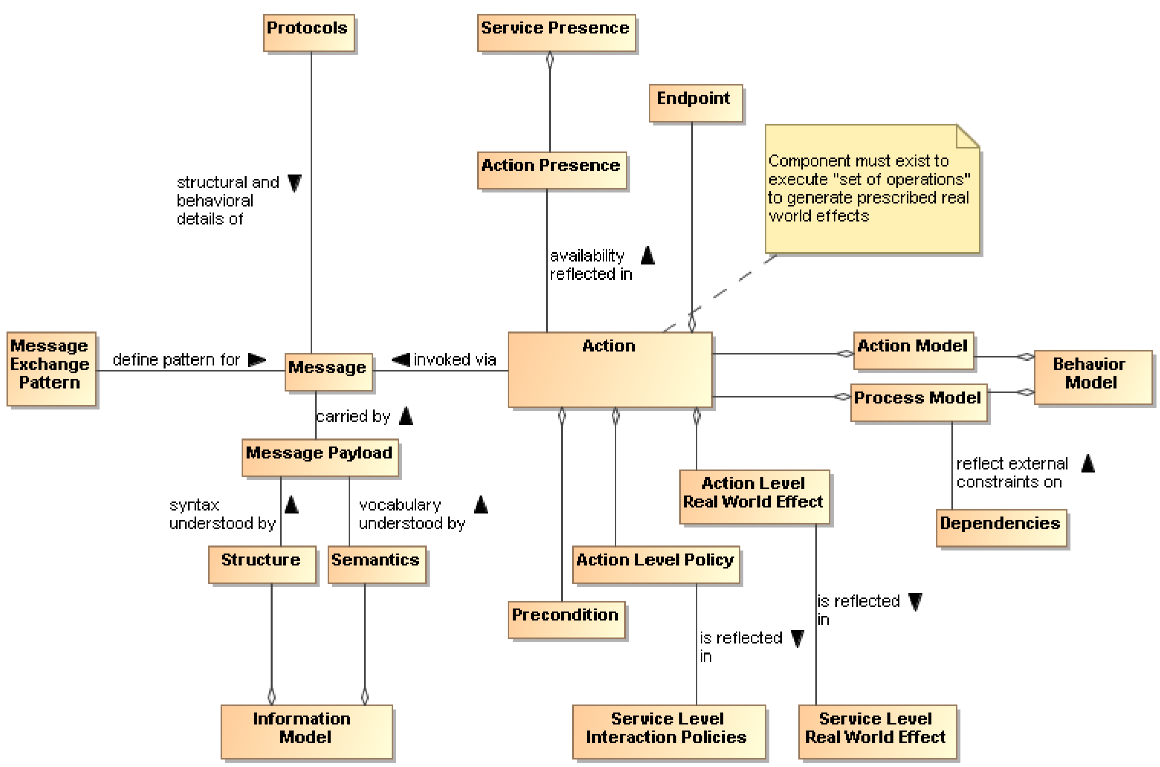


Figure 26 Model Showing Relationship Between Action and Service Description Components

> EDITOR'S NOTE:
>
> ONE QUESTION IS WHETHER THE MODEL SHOULD SHOW THE "SAME" ACTION AS POSSIBLY BEING INVOKED THROUGH THE SAME MESSAGE BUT USING A DIFFERENT PROTOCOL AT A DIFFERENT ENDPOINT AND THERE BEING A RELATIONSHIP BETWEEN ENDPOINT AND PROTOCOL. AGAIN, THIS MAY NOT BE PART OF THE SERVICE DESCRIPTION SECTION BUT OF THE DISCUSSION OF A MODEL FOR ACTION ELSEWHERE.

Figure 26 combines the Service Interface model of Figure 21 and the Service Reachability model of Figure 22 to concisely relate Action and the relevant components of Service Description. Action is invoked via a Message where the structure and behavioral details of the message conform to an identified Protocol, the message payload conforms to the service Information Model, and the message sequencing follows an identified Message Exchange Pattern. The protocol, information model, and message exchange pattern are identified in the service description.

The availability of an action is reflected in the Action Presence and each Action Presence contributes to the overall Service Presence. Each action has its own endpoint and also its own protocols associated with the endpoint[12] and to what extent, e.g. current or average availability, there is presence for the action through that endpoint. The endpoint and service presence are also part of the service description.

An action may have preconditions where a Precondition is something that needs to be in place before an action can occur, e.g. confirmation of a precursor action. Whether preconditions are satisfied is evaluated when someone tries to perform the action and not before. Presence for an action means someone can initiate it and is independent of whether the preconditions are satisfied. However, the successful completion of the action may depend on whether its preconditions were satisfied.

Presence of a service is an aggregation of the presence of the service's actions, and the service level may aggregate to some degraded or restricted presence if some action presence is not confirmed. For example, if error processing actions are not available, the service can still provide required functionality if no error processing is needed. This implies reachability relates to each action as well as applying to the service/business as a whole.

Analogous to the relationship between actions and preconditions, the Process Model may imply Dependencies for succeeding steps in a process, e.g. that a previous step has successfully completed, or may be isolated to a given step. An example of the latter would be a dependency that the host server has scheduled maintenance and access attempts at these times would fail. Dependencies related to the process model do not affect the presence of a service although these may affect whether the business function successfully completes.

The conditions under which an action can be invoked may depend on policies associated with the action. The Action Level Policies MUST be reflected in the Service Level Interaction Policies because such policies may be critical to determining whether the conditions for use of the service are consistent with the policies asserted by the service consumer. The service level interaction policies are included in the service description.

Similarly, the result of invoking an action is one or more real world effects, and the Action Level Real World Effects MUST be reflected in the Service Level Real World Effect included in the service description. If policies and real world effects at the action level are not unambiguously expressible at the service level, then the service description becomes inadequate for expressing conditions for use or results of using the service, and the understanding of what constitutes a service interaction is called into doubt.

From a description standpoint, a consumer would show interest in a service if the service functionality is what is needed and the service policies are at least worth pursuing if not immediately acceptable. By saying functionality is of interest, we are saying the (business) functions and service-level real world effects are of interest and there is nothing in the technical assumptions that preclude use of the service. Note at this level, the business functions are not concerned with the action or process models. These

[12] This is analogous to a WSDL 2.0 interface operation (WSDL 1.1 portType) having one or more defined bindings and the service identifies the endpoints (WSDL 1.1 ports) corresponding to the bindings.

models get into the nuts and bolts of making the business function happen and will be dealt with at that level later.

The service description is not intended to be isolated documentation but rather an integral part of service use. The initial use of any service should be based on information contained in the service description, and changes in service description should be pushed to known consumers. Thus, changes would not be introduced that later are captured in perpetually out-of-date documentation but rather reference to the service description should be an integral part of service use. This idea is consistent with checking the service endpoint before invoking a service action, but use of service description information should be more intrinsic than merely for a DNS-type function.

4.1.2.2.1 Description and Invoking Actions Against a Service

At this point, let us assume the descriptions were sufficient to establish willingness; see Section 4.2.3.2. Figure 26 indicates the service endpoint establishes where to go to actually carry out the interaction. This is where we have to start considering the action and process models.

The action model identifies the multiple actions a user can perform against a service and the user would perform these in the context of the process model as indicated under the Service Interface portion of Service Description. For a given business function, there is a corresponding process model, where any process model may involve multiple actions. From the above discussion of model elements of description we may conclude (1) actions have reachability information, including endpoint and presence, (2) presence of service is some aggregation of presence of its actions, (3) action preconditions and service dependencies do not affect presence although these may affect successful completion.

Having established visibility, the interaction can proceed. Given a business function, the consumer knows what will be accomplished (the service functionality), the conditions under which interaction will proceed (service policies and contracts), and the process that must be followed (the process model). Given the process model, the consumer knows which actions need to be performed; given the action, the consumer knows the endpoint and protocol to be used and whether there is presence for the action. The remaining question is how does the description information for structure and semantics enable interaction.

In the discussion above, we indicate the importance of the process model in identifying relevant actions and their sequence. Interaction with the actions are through messages and thus it is the syntax and semantics of the messages with which we are concerned. There seems to be a number of ways to approach this but the common way now is to define the structure and semantics that can appear as part of a message and then assemble the pieces into messages and associate messages with actions. Actions make use of structure and semantics as defined in the information model to describe its legal messages. In addition, the message exchange pattern defines sequencing and use of messages for a given action.

So to continue from above, the process model identifies actions to be performed against a service and the action sequence for performing the actions. For a given action, the Reachability portion of description indicates the protocol bindings that are available, the endpoint corresponding to a binding, and whether there is presence at that endpoint. The interaction with actions is through messages that conform to the structure and semantics defined in the information model and the message sequence conforming to the action’s identified MEP. The result is some portion of the real world effect initially examined in the service description (e.g. if an error exists, that part that covers the error processing would be invoked).

4.1.2.2.2 The question of multiple business functions

The service description model discussed above applies to the service and not the components of the service. For example, the Action Model identifies numerous actions that can be performed against a service and the Process Model defines the order in which the actions are performed, but the real world effects are defined for the service and not the individual actions. Similarly, numerous policies may be associated with a service, but policies at the action level must be reflected at the service level for service description to support visibility.

It is assumed that a SOA service represents an identifiable business function to which policies can be applied and from which desired business effects can be obtained. While contemporary discussions of SOA services and supporting standards do not constrain what actions or combinations of actions can or should be defined for a service, this Reference Architecture considers the implications of service description in defining the range of actions appropriate for an individual SOA service.

To begin, consider the situation if a given SOA service is the container for multiple independent (but possibly loosely related) business functions. Note, this is not multiple effects from a single function but multiple functions with potentially different sets of effects for each function. As noted above, a service can have multiple actions a user can perform against it, and this does not change with multiple business functions. An individual business function corresponds to a process model, so multiple business functions imply multiple process models because either the process is different or the specific action performed for some process step is different. The same action may be used in multiple process models but the aggregated service presence would be specific to each business function because the components being aggregated will likely be different between process models. In summary, for a service with multiple business functions, each function has (1) its own process model and dependencies, (2) its own aggregated presence, and (3) possibly its own list of policies and real world effects.

A common variation on this theme is for a single service to have multiple endpoints for different levels of quality of service (QoS). Different QoS imply separate statements of policy, separate endpoints, possibly separate dependencies, and so on. One could say the QoS variation does not require this because there can be a single QoS policy that encompasses the variations. and all other aspects of the service would be the same except for the endpoint used for each QoS. However, the different aspects of policy at the service level would need to be mapped to endpoints, and this introduces an undesirable level of coupling across the elements of description. In addition, it is obvious that description at the service level can become very complicated if combinations are allowed to grow.

One could imagine a service description that is basically a container for action descriptions, where each action description is self contained; however, this would lead to duplication of description components across actions. If common description components are factored, this either is limited to components common across all actions or requires complicated tagging to capture the components that often but do not universally apply.

If a provider cannot describe a service as a whole but must describe every action, this leads to the situation where it may be extremely difficult to construct a clear and concise service description that can effectively support discovery and use without tedious logic to process the description and assemble the available permutations. In effect, if adequate description of an action begins to look like description of a service, it may be best to have it as a separate service.

Recall, more than one service can access the same underlying capability, and this is appropriate if a different real world effect is to be exposed. Along these lines, one can argue that different QoS are different services because getting a response in one minute rather than one hour is more than a QoS difference; it is a fundamental difference in the business function being provided.

As a best practice, a criteria for whether a service is appropriately scoped may be the ease or difficulty in creating an unambiguous service description. A consequence of having tightly-scoped services is there will be a greater reliance on combining services, i.e. more fundamental business functions, to create more advanced business functions. This is consistent with the principles of service oriented architecture and is the basic position of the Reference Architecture, although not an absolute requirement. Combining services increases the reliance on understanding and implementing the concepts of orchestration, choreography, and other approaches yet to be developed; these are discussed in more detail in section 4.4 Interacting with Services.

4.1.2.2.3 Service Description, Execution Context, and Service Interaction

The service description provides sufficient information to support service visibility, including the willing of service participants to interact. However, the corresponding descriptions for providers and consumers may both contain policies, technical assumptions, constraints on semantics, and other technical and procedural conditions that must be aligned to define the terms of willingness. The agreements which encapsulate the necessary alignment form the basis upon which interactions may proceed – in the SOA Reference Model, this collection of agreements and the necessary environmental support establish the execution context.

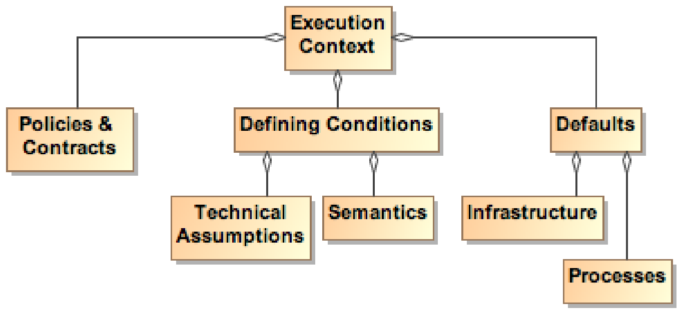


Figure 27 Execution Context model

Figure 27 shows a number of contributors to the execution context. These broad categories are meant to include any disconnects that could get in the way of interoperability and successful interactions, but other items may need to be included to collect a sufficient description of the interaction conditions. Any other items not explicitly noted in the model but needed to set the environment would also be a candidate for including in the execution context. However, as noted in the Reference Model, it is not possible to describe everything and so a set of information items as potentially extensive as the execution context will never be complete in every detail. As with the service description, the goal is to be sufficiently complete for the task at hand.

While the execution context captures the conditions under which interaction can occur, it does not capture the specific service invocations that do occur in a specific interaction. A service interaction as modeled in Figure 28 introduces the concept of an Interaction Description which is composed of both the Execution Context and an Interaction Log. The execution context specifies the set of conditions under which the interaction occurs and the interaction log captures the sequence of service interactions that occur within the execution context. The execution context can be thought of as the container in which the interaction occurs and the interaction log captures what happens inside the container. This combination is needed to support auditability and repeatability of the interactions.

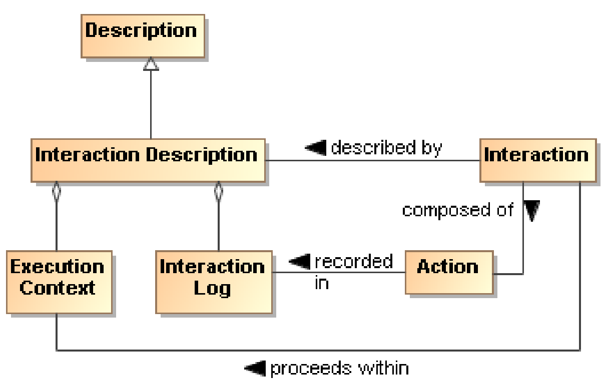


Figure 28 Service Interaction model

With respect to repeatability, SOA allows for a great deal of flexibility and one of its benefits is that services and their underlying capabilities can be updated without disturbing the consumers. So, for example, Google can improve their ranking algorithm in a manner transparent to the typical user without the user being concerned with the details of the update. Indeed, improvements in Google often depend on the user being unaware of updates because that allows Google to adapt to content providers trying to game the ranking algorithms.

However, it may also be vital for the consumer to be able to recreate past results or to generate consistent results in the future, and information such as what conditions, which services, and which versions of those services are used is indispensible in retracing one's path. The interaction log is a critical part of the resulting real world effects because it defines how the effects were generated and possibly the meaning of observed effects. This increases in importance as dynamic composability becomes more feasible. In essence, a result has limited value if one does not know how it was generated.

The interaction log is a detailed trace for a specific interaction, and its reuse is limited to duplicating that interaction. On the other hand, an execution context can be reusable for the same participants using the same services or it can act as a template for those items to consider for similar interactions. A previous execution context could provide a starting point for defining the conditions of future interactions, either between the same consumer and provider or by like-minded consumers and providers attempting to carry out similar tasks.

Such uses of execution context imply (1) a standardized format for capturing execution context and (2) a subclass of general description could be defined to support visibility of saved execution contexts. The specifics of the relevant formats and descriptions are beyond the scope of this Reference Architecture.

A service description is unlikely to track interaction descriptions or the constituent execution contexts or interaction logs that include mention of the service. However, as appropriate, linking to specific instances of either of these could be done through associated annotations.

4.1.3 Relationship to Other Description Models

While the representation shown in Figure 25 is derived from considerations related to service description, it is acknowledged that other metadata standards are relevant and should, as possible, be incorporated into this work. Two standards of particular relevance are the Dublin Core Metadata Initiative (DCMI) and ISO 11179, especially Part 5.

When the service description (or even the general description class) is considered as the DCMI "resource", Figure 25 aligns nicely with the DCMI resource model. While some differences exist, these are mostly in areas where DCMI goes into detail that is considered beyond the scope of the current Reference Architecture. For example, DCMI defines classes of "shared semantics" whereas for the Reference Architecture, it is sufficient to prescribe that an identification of relevant semantic models is sufficient. Likewise, the DCMI "description model" goes into the details of possible syntax encodings whereas for the Reference Architecture it is sufficient to identify the relevant formats.

With respect to ISO 11179 Part 5, the metadata fields defined in that reference may be used without prejudice as the properties in Figure 25 above. Additionally, other defined metadata sets may be used by the service provider if the other sets are considered more appropriate, i.e. it is fundamental to this Reference Architecture to identify the need and the means to make vocabulary declarations explicit but it is beyond the scope to specify which vocabularies are to be used. In addition, the identification of domain of the properties and range of the values has not been included in the current Reference Architecture discussion, but the text of ISO 11179 Part 5 can be used consistently with the model prescribed in this document.

Description as defined in the context of this Reference Architecture considers a wide range of applicability and support of the principles of service oriented architecture. Other metadata models can be used in concert with the model presented here because most of these focus on a finer level of detail that is outside the present scope, and so provide a level of implementation guidance that can be applied as appropriate.

4.1.4 Architectural Implications

The description of service description indicates numerous architectural implications on the SOA ecosystem:

- Description will change over time and its contents will reflect changing needs and context. This requires the existence of:
    - o mechanisms to support the storage, referencing, and access to normative definitions of one or more versioning schemes that may be applied to identify different aggregations of descriptive information, where the different schemes may be versions of a versioning scheme itself;
    - o configuration management mechanisms to capture the contents of the each aggregation and apply a unique identifier in a manner consistent with an identified versioning scheme;

    - one or more mechanisms to support the storage, referencing, and access to conversion relationships between versioning schemes, and the mechanisms to carry out such conversions.
- Description makes use of defined semantics, where the semantics may be used for categorization or providing other property and value information for description classes. This requires the existence of:
    - semantic models that provide normative descriptions of the utilized terms, where the models may range from a simple dictionary of terms to an ontology showing complex relationships and capable of supporting enhanced reasoning;
    - mechanisms to support the storage, referencing, and access to these semantic models;
    - configuration management mechanisms to capture the normative description of each semantic model and to apply a unique identifier in a manner consistent with an identified versioning scheme;
    - one or more mechanisms to support the storage, referencing, and access to conversion relationships between semantic models, and the mechanisms to carry out such conversions.
- Descriptions include reference to policies defining conditions of use and optionally contracts representing agreement on policies and other conditions. This requires the existence of (as also enumerated under governance):
    - descriptions to enable the policy modules to be visible, where the description includes a unique identifier for the policy and a sufficient, and preferably a machine processible, representation of the meaning of terms used to describe the policy, its functions, and its effects;
    - one or more discovery mechanisms that enable searching for policies that best meet the search criteria specified by the service participant; where the discovery mechanism will have access to the individual policy descriptions, possibly through some repository mechanism;
    - accessible storage of policies and policy descriptions, so service participants can access, examine, and use the policies as defined.
- Descriptions include references to metrics which describe the operational characteristics of the subjects being described. This requires the existence of (as partially enumerated under governance):
    - the infrastructure monitoring and reporting information on SOA resources;
    - possible interface requirements to make accessible metrics information generated or most easily accessed by the service itself;
    - mechanisms to catalog and enable discovery of which metrics are available for a described resources and information on how these metrics can be accessed;
    - mechanisms to catalog and enable discovery of compliance records associated with policies and contracts that are based on these metrics.
- Descriptions of the interactions are important for enabling auditability and repeatability, thereby establishing a context for results and support for understanding observed change in performance or results. This requires the existence of:
    - one or more mechanisms to capture, describe, store, discover, and retrieve interaction logs, execution contexts, and the combined interaction descriptions;
    - one or more mechanisms for attaching to any results the means to identify and retrieve the interaction description under which the results were generated.
- Descriptions may capture very focused information subsets or can be an aggregate of numerous component descriptions. Service description is an example of a likely aggregate for which manual maintenance of all aspects would not be feasible. This requires the existence of:
    - tools to facilitate identifying description elements that are to be aggregated to assemble the composite description;
    - tools to facilitate identifying the sources of information to associate with the description elements;
    - tools to collect the identified description elements and their associated sources into a standard, referenceable format that can support general access and understanding;

    - tools to automatically update the composite description as the component sources change, and to consistently apply versioning schemes to identify the new description contents and the type and significance of change that occurred.
- Descriptions provide up-to-date information on what a resource is, the conditions for interacting with the resource, and the results of such interactions. As such, the description is the source of vital information in establishing willingness to interact with a resource, reachability to make interaction possible, and compliance with relevant conditions of use. This requires the existence of:
    - one or more discovery mechanisms that enable searching for described resources that best meet the criteria specified by a service participant, where the discovery mechanism will have access to individual descriptions, possibly through some repository mechanism;
    - tools to appropriately track users of the descriptions and notify them when a new version of the description is available.

4.2 Service Visibility Model

One of the key requirements for participants interacting with each other in the context of a SOA is achieving visibility: before services can interoperate, the participants have to be visible to each other using whatever means are appropriate. The Reference Model analyzes visibility in terms of awareness, willingness, and reachability. In this section, we explore how visibility may be achieved.

4.2.1 Visibility to Business

The relationship of visibility to the SOA ecosystem encompasses both human social structures and automated IT mechanisms. Figure 29 depicts a business setting that is a basis for visibility as related to the Social Structure Model in the Business Via Services View (see Section 3.4). Service consumers and service providers may have direct awareness or mediated awareness where mediated awareness is achieved through some third party. A consumer’s willingness to use a service is reflected by the consumer’s presumption of satisfying goals and needs based on the description of the service. Service providers offer capabilities that have real world affects that result in a change in state of the consumer. Reachability of the service by the consumer leads to interactions that change the state of the consumer. The consumer can measure the change of state to determine if the claims made by description and the real world effects of consuming the service meet the consumer’s needs.

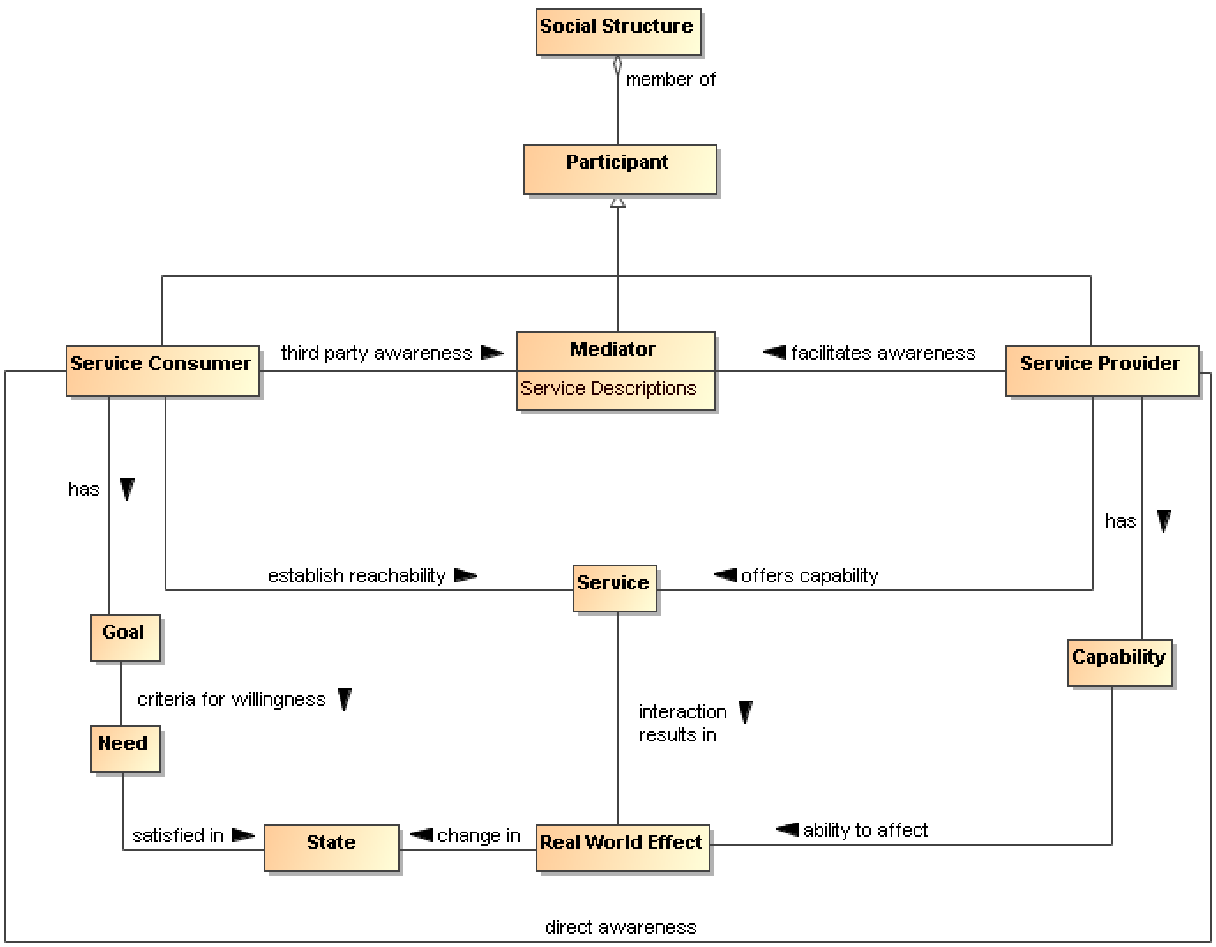


Figure 29 Visibility to Business Model

Visibility and interoperability in a SOA ecosystem requires more than location and interface information, or the traditional Application Programming Interface (API). A meta-model for this broader view of visibility is depicted in Section 4.1. In addition to providing improved awareness of service capabilities the service description may contain policies valuable for determination of willingness to interact.

Another important business capability in a SOA environment is the ability to narrow visibility to trusted members within a social structure, often referred to as Communities of Interest (COI) in government sectors. Mediators for awareness may provide policy based access to service descriptions, allowing for the dynamic formation of awareness between members of a COI.

A mediator of service descriptions may also provide event notifications to both consumers and providers about information relating to service descriptions. One example of this capability is a publish/subscribe model where the mediator allows consumers to subscribe to service description version changes made by the provider. Likewise, the mediator may provide notifications to the provider of consumers that have subscribed to service description updates.

4.2.2 Attaining Visibility

Attaining visibility is described in terms of steps that lead to visibility. While there can be many contexts for visibility within a single social structure, the same general steps can be applied to each of the contexts to accomplish visibility.

Attaining SOA visibility requires

- service description creation and maintenance,
- processes and mechanisms for achieving awareness of and accessing descriptions,

- processes and mechanisms for establishing willingness of participants,
- processes and mechanisms to determine reachability.

Visibility may occur in stages, i.e. a participant can become aware enough to look or ask for further description, and with this description, the participant can decide on willingness, possibly requiring additional description. For example, if a potential consumer has a need for a tree cutting (business) service, the consumer can use a web search engine to find web sites of providers. The web search engine (a mediator) gives the consumer links to relevant web pages and the consumer can access those descriptions. For those prospective providers that satisfy the consumer's criteria, the consumer's willingness to interact increases. The consumer likely contacts several tree services to get detailed cost information (or arrange for an estimate) and may ask for references (further description). Likely, the consumer will establish full visibility and proceed with the interaction with a tree service who mutually establishes visibility.

4.2.2.1 Achieving Awareness

A service participant is aware of another participant if it has access to a description of that participant with sufficient completeness to establish the other requirements of visibility.

Awareness is inherently a function of a participant; awareness can be established without any action on the part of the target participant other than the target providing appropriate descriptions. Awareness is often discussed in terms of consumer awareness of providers but the concepts are equally valid for provider awareness of consumers.

Awareness can be decomposed into the creation of descriptions, making them available, and discovering the descriptions. Discovery in the Service Visibility Model is the process where a consumer discovers a service description or a service provider discovers a likely consumer's description. Discovery can be initiated or it can be by notification. Initiated discovery for business may require formalization of the required capabilities and resources to achieve business goals. Figure 30 and Figure 31 depict a typical process for achieving awareness.

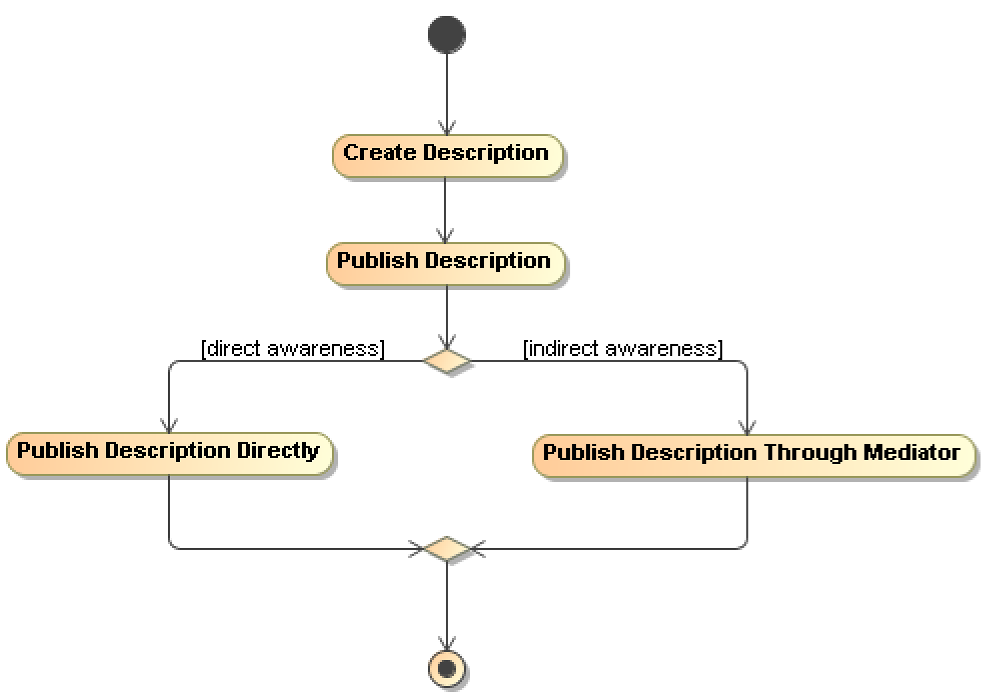


Figure 30 Publishing Description

A mediator as discussed for awareness is a third party participant that provides awareness to one or more consumers of one or more services. See Section 3.1, for an overview of participants. Direct awareness is awareness between a consumer and provider without the use of a third party. Direct awareness may be the result of having previously established an execution context and possibly indicates successful interaction has occurred in the past.

The same medium for awareness may be direct in one context and may be mediated in another context. For example, a service provider may maintain a web site with links to the provider's descriptions of services giving the consumers direct awareness to the provider's services. Alternatively, a community may maintain a mediated web site with links to various provider descriptions of services for any number of consumers. More than one mediator may be involved, as different mediators may specialize in different mediation functions.

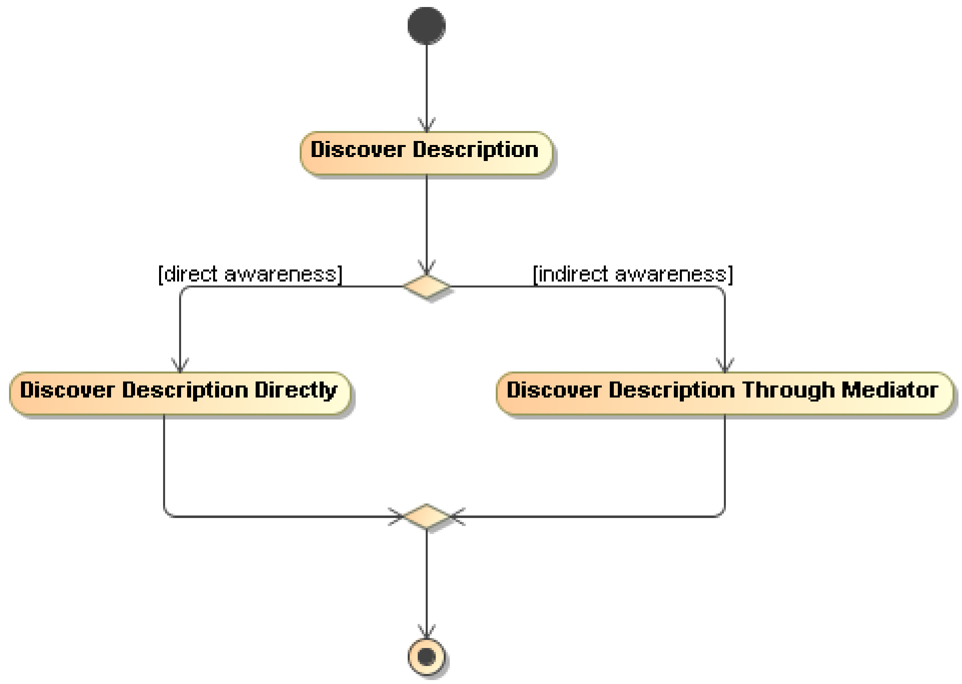


Figure 31 Discovering Description

There may be numerous methods to facilitate discovery. For example, descriptions could be discovered by browsing a web site, querying a public registry, or via email notifications.

Descriptions may be formal or informal. Section 4.1, provides a comprehensive model for service description that can be applied to formal registry/repositories used to mediate visibility. Using consistent description taxonomies and standards based mediated awareness helps provide more effective awareness.

4.2.2.1.1 Mediated Awareness

Mediated awareness promotes loose coupling by keeping the consumers and services from explicitly referring to each other and the descriptions. Mediation lets interaction vary independently. Rather than all potential service consumers being informed on a continual basis about all services, there is a known or agreed upon facility or location that houses the service description.

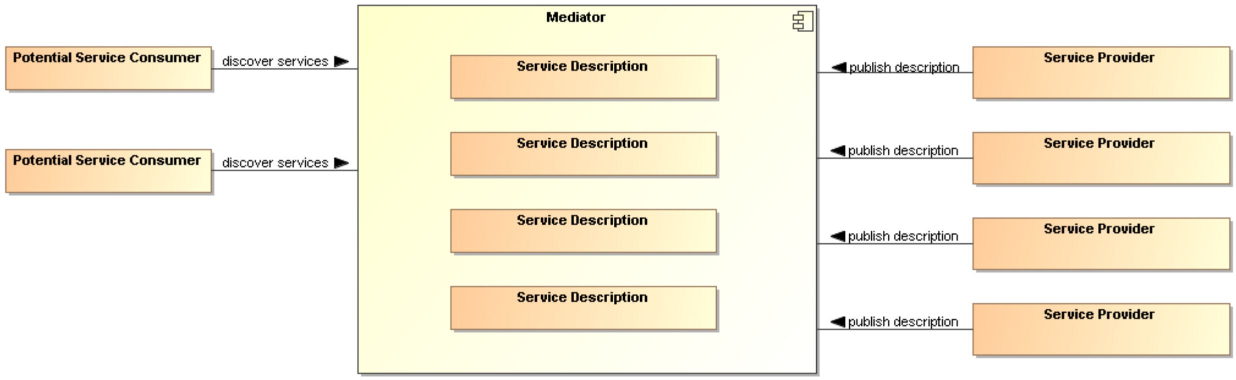


Figure 32 Mediated Service Awareness

In Figure 32, the potential service consumers perform queries or are notified in order to locate those services that satisfy their needs. As an example, the telephone book is a mediated registry where individuals perform manual searches to locate services (i.e. the yellow pages). The telephone book is also a mediated registry for solicitors to find and notify potential customers (i.e. the white pages).

In mediated service awareness for large and dynamic numbers of service consumers and service providers, the benefits typically far outweigh the management issues associated with it. Some of the benefits of mediated service awareness are

- Potential service consumers have a known location for searching thereby eliminating needless and random searches
- Typically a consortium of interested parties (or a sufficiently large corporation) signs up to host the mediation facility
- Standardized tools and methods can be developed and promulgated to promote interoperability and ease of use.

However, mediated awareness can have some risks associated with it:

- A single point of failure. If the central mediation service fails then a potentially large number of service providers and consumers will be adversely affected.
- A single point of control. If the central mediation service is owned by, or controlled by, someone other than the service consumers and/or providers then the latter may be put at a competitive disadvantage based on policies of the discovery provider.

4.2.2.1.2 Awareness in Complex Social Structures

Awareness applies to one or more communities within one or more social structures where a community consists of at least one description provider and one description consumer. These communities may be part of the same social structure or be part of different ones.

In Figure 33, awareness can be within a single community, multiple communities, or all communities in the social structure. The social structure can encourage or restrict awareness through its policies, and these policies can affect participant willingness. The information about policies should be incorporated in the relevant descriptions. The social structure also governs the conditions for establishing contracts, the results of which will be reflected in the execution context if interaction is to proceed.

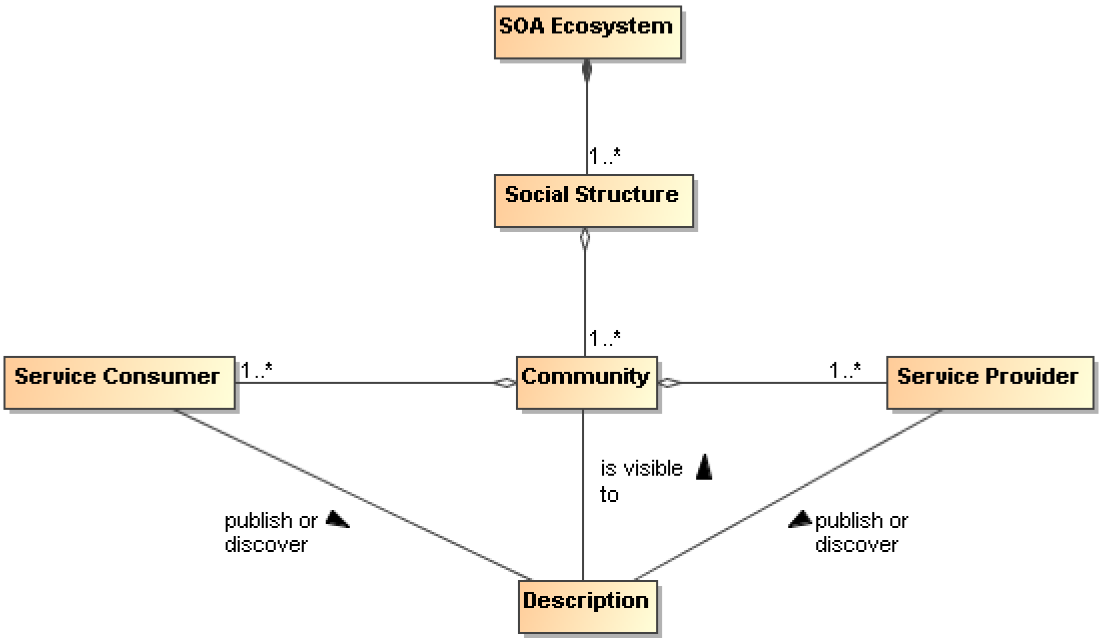


Figure 33 Awareness In a SOA Ecosystem

IT policy/contract mechanisms can be used by visibility mechanisms to provide awareness between communities. The IT mechanisms for awareness may incorporate trust mechanisms to assure awareness between trusted communities. For example, government organizations will often want to limit awareness of an organization's services to specific communities of interest.

Another common business model for awareness is maximizing awareness to communities within the social structure, the traditional market place business model. A centralized mediator often arises as a provider for this global visibility, a gatekeeper of visibility so to speak. For example, Google is a centralized mediator for accessing information on the web. As another example, television networks have centralized entities providing a level of awareness to communities that otherwise could not be achieved without going through the television network.

However, mediators have motivations, and they may be selective in which information they choose to make available to potential consumers. For example, in a secure environment, the mediator may enforce security policies and make information selectively available depending on the security clearance of the consumers.

4.2.2.2 Determining Willingness

Having achieved awareness, participants use descriptions to help determine their willingness to interact with another participant. Both awareness and willingness are determined prior to consumer/provider interaction. The activities in Figure 34, or a subset there of, can be performed to help determine willingness.

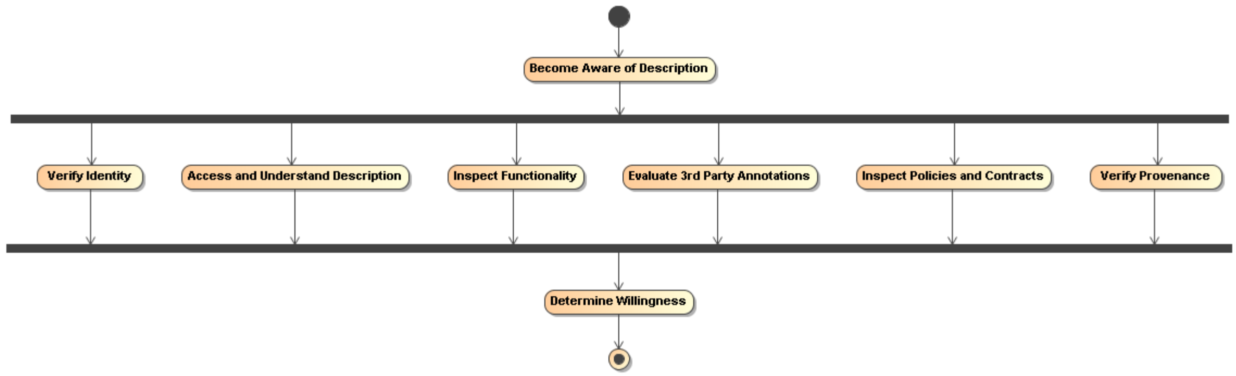


Figure 34 Determining Willingness

In any given process to determine willingness, one or more of the transitions or flows depicted above may be executed. For example, in a particular service interaction, it may be important to inspect policies and to verify provenance; another interaction may call for evaluating 3rd party annotations in addition.

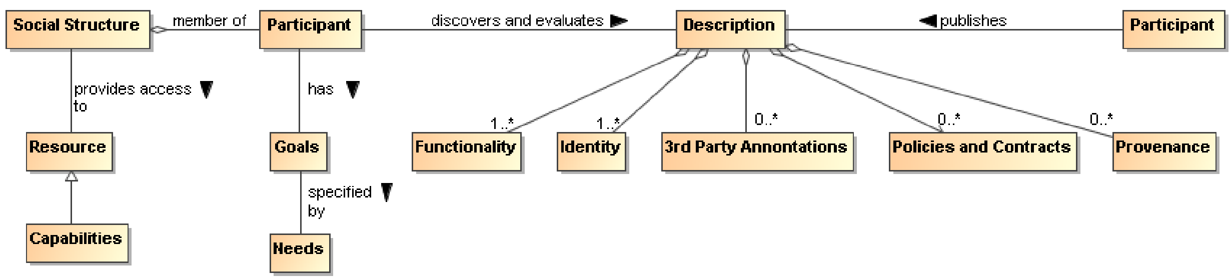


Figure 35 Business, Description and Willingness

Figure 35 relates elements of the Business via Services View, and elements from the Service Description Model to willingness. By having a willingness to interact within a particular social structure, the social structure provides the participant access to capabilities based on conditions the social structure finds appropriate for its context. The participant can use these capabilities to satisfy goals and objectives as specified by the participant's needs.

In Figure 35, information used to determine willingness is defined by Description. Information referenced by Description may come from many sources. For example, a mediator for descriptions may provide 3rd party annotations for reputation. Another source for reputation may be a participant's own history of interactions with another participant.

A participant will inspect functionality for potential satisfaction of needs. Identity is associated with any participant, however, identity may or may not be verified. If available, participant reputation may be a deciding factor for willingness to interact. Policies and contracts referenced by the description may be particularly important to determine the agreements and commitments required for business interactions. Provenance may be used for verification of authenticity of a resource.

4.2.2.3 Establishing Reachability

Reachability involves knowing the service endpoint, service interface, and presence of a service. Figure 36 lists activities involved to establish reachability. For reachability, service descriptions should include sufficient data to enable a service consumer and service provider to interact with each other. At a minimum, service descriptions should include information about the location of the service and the service interface. The subject of access control and other process model type activities to establish a connection are left for the Interacting with Services Model.

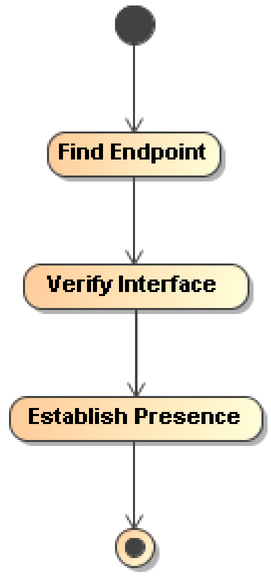


Figure 36 Establishing Reachability

Endpoint

An endpoint is a reference-able entity, processor or resource against which an action can be performed.

Interface

Interface verification involves determination of compatible communication protocols, compatible message exchange capabilities, and service interface version.

Presence

Presence is established when a service can be reached at a particular point in time. Presence may not be known in many cases until the act of interaction begins. To overcome this problem, IT mechanisms may make use of presence protocols to provide the current up/down status of a service.

Service reachability enables service participants to locate and interact with one another. Each action may have its own endpoint and also its own protocols associated with the endpoint[13] and whether there is presence for the action through that endpoint. Presence of a service is an aggregation of the presence of the service's actions, and the service level may aggregate to some degraded or restricted presence if some action presence is not confirmed. For example, if error processing actions are not available, the service can still provide required functionality if no error processing is needed. This implies reachability relates to each action as well as applying to the service/business as a whole

After reachability has been established, there may be times when participants need to re-establish reachability such as when a service fails and a new location and version for the service needs to be determined. Disconnected operations is another example for re-establishment of reachability. For SOA, both endpoint location and service interface version are important for re-establishing reachability. For example, multiple versions of a service may be in operation for backward compatibility. A Domain Name Service (DNS) lookup for service location may not be sufficient for re-establishing service reachability after a failure.

[13] This is analogous to a WSDL 2.0 interface operation (WSDL 1.1 portType) having one or more defined bindings and the service identifies the endpoints (WSDL 1.1 ports) corresponding to the bindings.

4.2.3 Mechanisms for Attaining Visibility

While there can be many mechanisms for service visibility in a SOA, this section covers some examples of those mechanisms.

4.2.3.1 Mechanisms for Awareness

Achieving awareness in a SOA can range from word of mouth to formal Service Descriptions in a standards based registry-repository.   Some other examples of achieving awareness in a SOA are the use of a web page containing description information, email notifications of descriptions, and document based descriptions.

A common mechanism for mediated awareness in the industry is a registry-repository. Figure 37 depicts a mediation facility containing a registry and a repository. The registry stores links or pointers to service description artifacts. The repository in this example is the storage location for the service description artifacts. Service descriptions can be pushed (publish/subscribe for example) or pulled from the register-repository mediator.

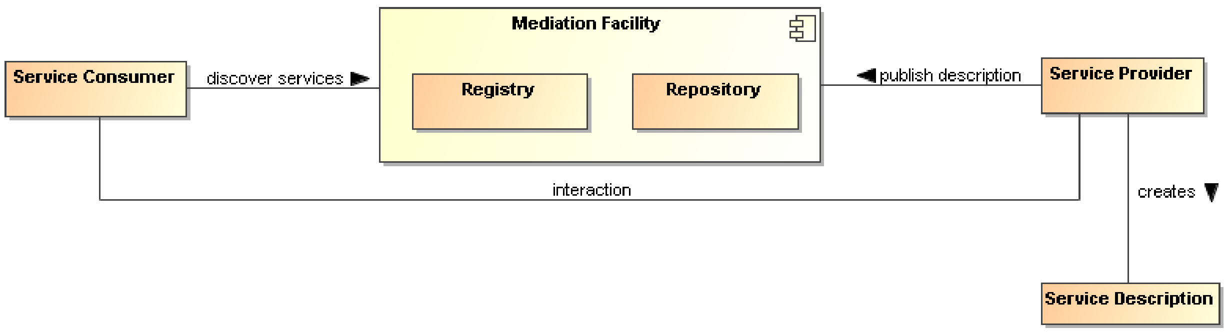


Figure 37 Mediated Registry-Repository

The registry is like a card catalog at the library and a repository is like the shelves for the books. Standardized metadata describing repository content can be stored as registry objects in a registry and any type of content can be stored as repository items in a repository. The registry may be constructed such that description items stored within the mediation facility repository will have intrinsic links in the registry while description items stored outside the mediation facility will have extrinsic links in the registry.

When like SOA IT mechanisms interoperate with one another, the IT mechanisms may be referred to as federated. An example use of federation is combining different domains of knowledge as in Figure 38.

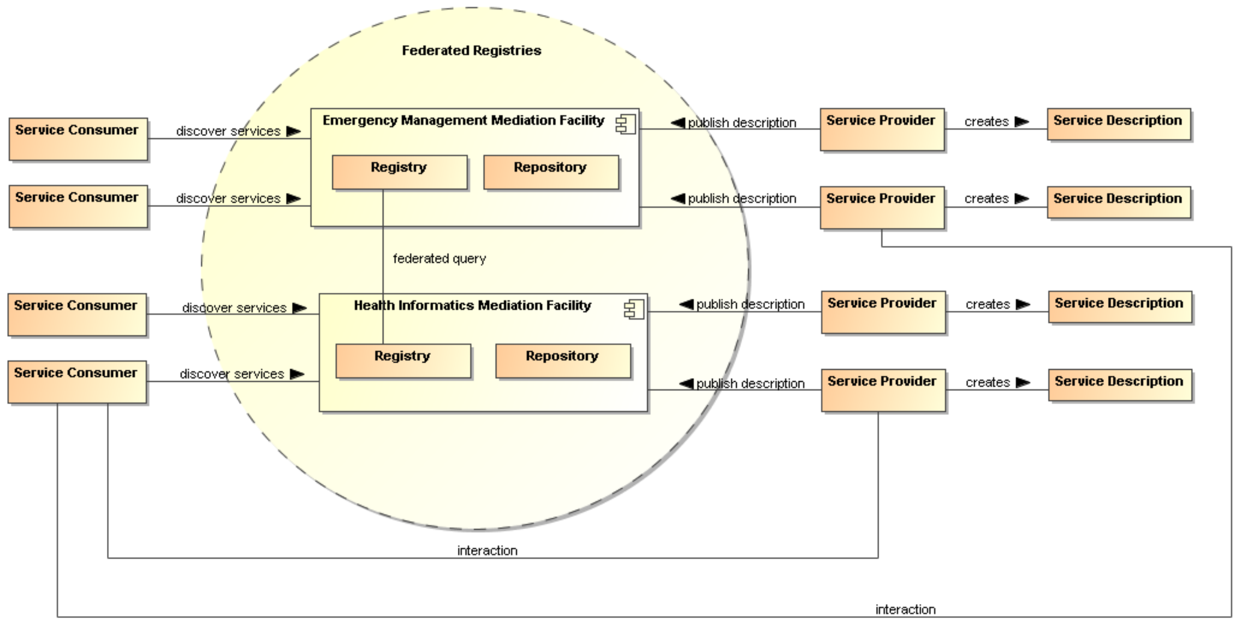


Figure 38 Federated Registry-Repository

4.2.3.2 Mechanisms for Willingness

Mechanisms that aid in determining willingness make use of the artifacts referenced by descriptions of services. Mechanisms for establishing willingness could be as simple as rendering service description information for human consumption to automated evaluation of functionality, policies, and contracts by a rules engine. The rules engine for determining willingness could operate as a policy decision point as defined in Section 4.4.

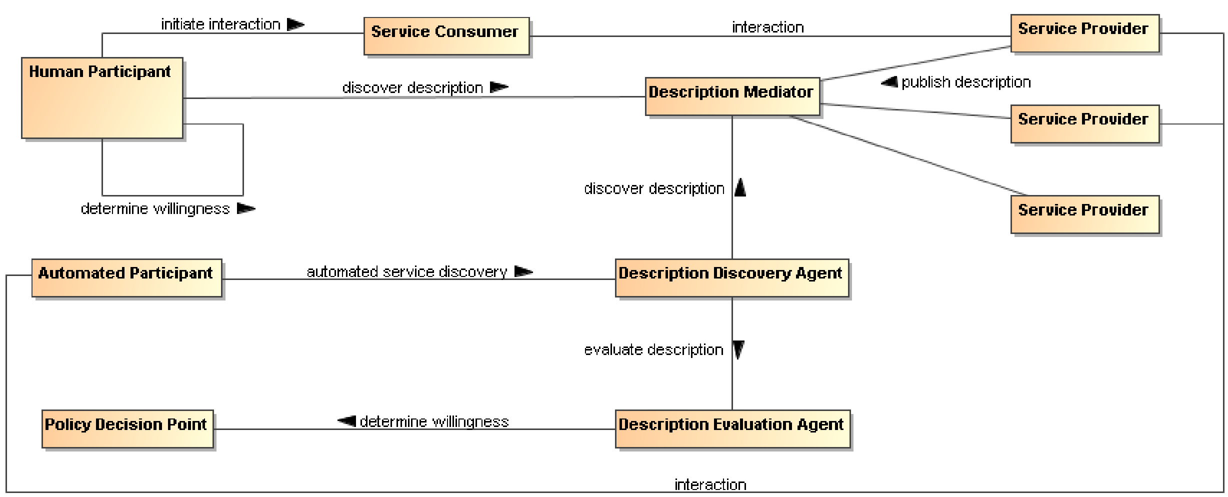


Figure 39 Mechanisms for Willingness

Figure 25 is an example of manual determination of willingness by a human participant and one possible example of automated determination of willingness. For functionality that may be provided by the Enterprise Service Bus see Section 4.3.3. For models explaining the Policy Decision Point see Section 4.4.

4.2.3.3 Mechanisms for Reachability

Reachability mechanisms will often begin with a tool that is capable of reading service description interfaces and generating a client capable of interacting with the provider's service. The establishment of presence occurs when the client has started interactions with the provider's service. Expected service operating times may be published as part of service description. Presence protocols may also be implemented to provide further assurance of presence of a service.

4.2.4 Architectural Implications

Visibility in a SOA ecosystem has the following architectural implications on mechanisms providing support for awareness, willingness, and reachability:

- Mechanisms providing support for awareness will likely have the following minimum capabilities:
    - creation of Description, preferably conforming to a standard Description format and structure;
    - publishing of Description directly to a consumer or through a third party mediator;
    - discovery of Description, preferably conforming to a standard for Description discovery;
    - notification of Description updates or notification of the addition of new and relevant Descriptions;
    - classification of Description elements according to standardized classification schemes.
- In a SOA ecosystem with complex social structures, awareness may be provided for specific communities of interest. The architectural mechanisms for providing awareness to communities of interest will require support for:
    - policies that allow dynamic formation of communities of interest;
    - trust that awareness can be provided for and only for specific communities of interest, the bases of which is typically built on keying and encryption technology.
- The architectural mechanisms for determining willingness to interact will require support for:
    - verification of identity and credentials of the provider and/or consumer;
    - access to and understanding of description;
    - inspection of functionality and capabilities;
    - inspection of policies and/or contracts.
- The architectural mechanisms for establishing reachability will require support for:
    - the location or address of an endpoint;
    - verification and use of a service interface which includes communication protocols, message exchange capabilities, and service interface version;
    - determination of presence with an endpoint which may only be determined at the point of interaction but may be further aided by the use of a presence protocol for which the endpoints actively participate.

4.3 Interacting with Services Model

Interaction is the use of a service to access capability in order to achieve a particular desired real world effect, where real world effect is the actual *result* of using a service. An interaction can be characterized by a sequence of actions. Consequently, interacting with a service involves performing actions against the service, usually through a series of information exchanges (e.g., messages), although other modes of interaction are possible such as modifying the shared state of a resource. Note that a participant (or agent acting on behalf of the participant) can be the sender of a message, the receiver of a message, or both.

For purposes of this SOA Reference Architecture, the authors have committed to the use of message exchange between service participants to denote actions against the services that *cause* a real world effect, and to denote events that *report* on real world effects that arise from those actions.

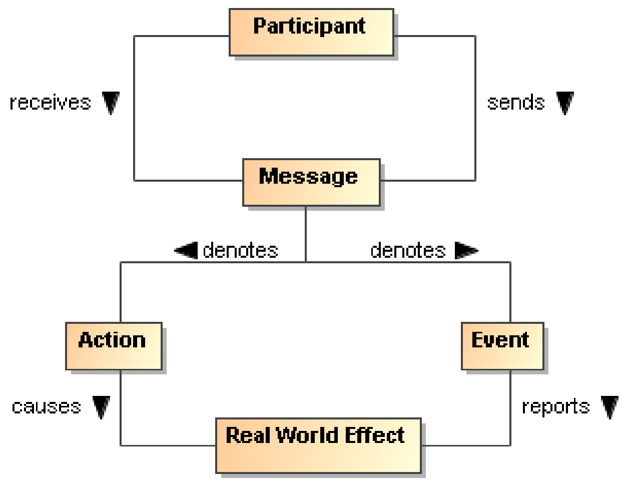


Figure 40 A "message" denotes either an action or an event.

A *Message* denotes either an action or an event. In other words, both actions and events are realized through messages. The OASIS Reference Model states that the Action Model characterizes the “permissible set of actions that may be invoked against a service.” We extend that notion here to include events as part of the action model and that messages denote either actions or events.

4.3.1 Actions and Events

In Section 3.5.1, we saw that participants interact with each other in order to perform actions. An action is not itself the same thing as the result of performing the action. When an action is performed against a service, the real world effect that results is reported in the form of events (see Section 3.5.1).

In this Reference Architecture, we use *messages* and *message exchange* to denote both actions and results of actions.

4.3.2 Message Exchange

Message exchange is the means by which service participants (or their agents) interact with each other. There are two primary modes of interaction: joint actions that cause real world effects, and notification of events that report real world effects. [14]

A message exchange is used to affect an action when the messages contain the appropriately formatted content that should be interpreted as joint action and the agents involved interpret the message appropriately.

A message exchange is also used to communicate event notifications. An event is a report of an occurrence that is of interest to some participant; in our case when some real world effect has occurred. Just as action messages will have formatting requirements, so will event notification messages. In this way, the Information Model of a service must specify the syntax (structure), and semantics (meaning) of the action messages and event notification messages as part of a service interface. It must also specify the syntax and semantics of any data that is carried as part of a payload of the action or event notification message. The Information Model is described in greater detail in the Service Description Model (see Section 4.1).

In addition to the Information Model that describes the syntax and semantics of the messages and data payloads, exception conditions and error handling in the event of faults (e.g., network outages, improper message formats, etc.) must be specified or referenced as part of the Service Description.

[14] The notion of “joint” in joint action implies that you have to have a speaker *and* a listener in order to interact.

When a message is interpreted as an action, the correct interpretation typically requires the receiver to perform a set of operations. These *operations* represent the sequence of actions (often private) a service must perform in order to validly participate in a given joint action.

Similarly, the correct consequence of realizing a real world effect may be to initiate the reporting of that real world effect via an event notification.

Message Exchange

The means by which joint actions and event notifications are coordinated by service participants (or agents).

Operations

The sequence of actions a service must perform in order to validly participate in a given joint action.

4.3.2.1 Message Exchange Patterns (MEPs)

As stated earlier, this Reference Architecture commits to the use of message exchange to denote actions against the services, and to denote events that report on real world effects that arise from those actions.

Based on these assumptions, the basic temporal aspect of service interaction can be characterized by two fundamental message exchange patterns (MEPs):

- Request/response to represent how actions cause a real world effect
- Event notification to represent how events report a real world effect

This is by no means a complete list of all possible MEPs used for inter- or intra-enterprise messaging but it does represent those that are most commonly used in exchange of information and reporting changes in state both within organizations and across organizational boundaries, a hallmark of a SOA.

Recall from the OASIS Reference Model that the Process Model characterizes “the temporal relationships between and temporal properties of actions and events associated with interacting with the service.” Thus, MEPs are a key element of the Process Model. The meta-level aspects of the Process Model (just as with the Action Model) are provided as part of the Service Description Model (see Section 4.1).

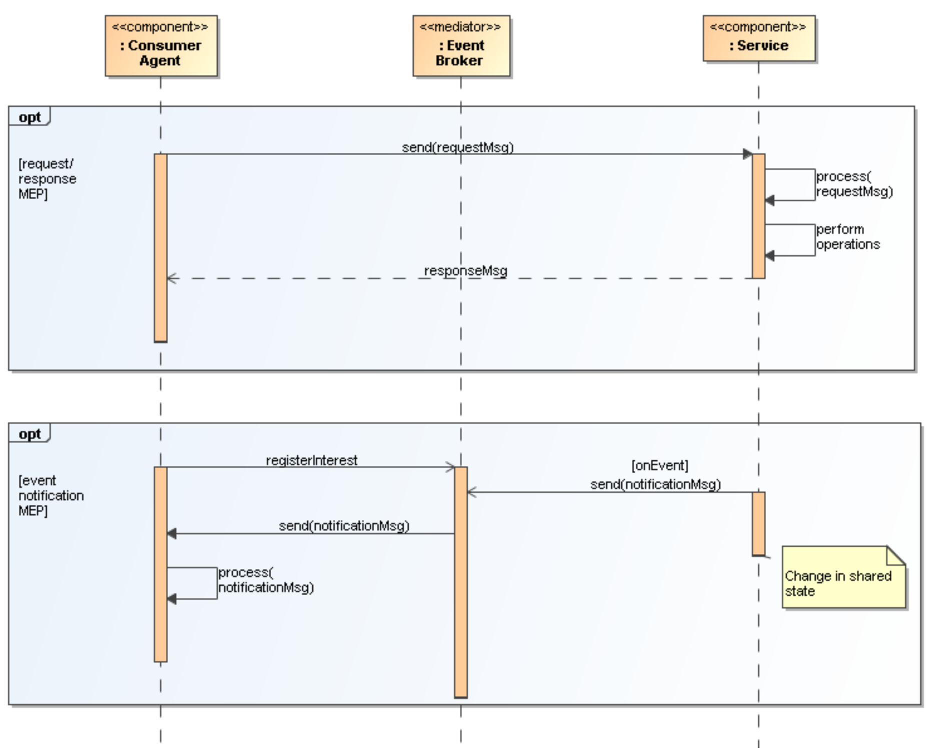


Figure 41 Fundamental SOA message exchange patterns (MEPs)

In the UML sequence diagram shown in Figure 41 it is assumed that the service participants (consumer and provider) have delegated message handling to hardware or software agents acting on their behalf. In the case of the service consumer, this is represented by the *Consumer Agent* component. In the case of the service provider, the agent is represented by the *Service* component. The message interchange model illustrated represents a logical view of the MEPs and not a physical view. In other words, specific hosts, network protocols, and underlying messaging system are not shown as these tend to be implementation specific. Although such implementation-specific elements are considered outside the scope of this Reference Architecture, they are important considerations in modeling the SOA execution context. Recall from the Reference Model that the *execution context* of a service interaction is “the set of infrastructure elements, process entities, policy assertions and agreements that are identified as part of an instantiated service interaction, and thus forms a path between those with needs and those with capabilities.”

4.3.2.2 Request/Response MEP

In a request/response MEP, the Consumer Agent component sends a request message to the Service component. The Service component then processes the request message. Based on the content of the message, the Service component performs the service operations. Following the completion of these operations, a response message is returned to the Consumer Agent component. The response could be that a step in a process is complete, the initiation of a follow-on operation, or the return of requested information.[15]

Although the sequence diagram shows a *synchronous* interaction (because the sender of the request message, i.e., Consumer Agent, is blocked from continued processing until a response is returned from the Service) other variations of request/response are valid, including *asynchronous* (non-blocking) interaction through use of queues, channels, or other messaging techniques.

What is important to convey here is that the request/response MEP represents *action*, which causes a real world effect, irrespective of the underlying messaging techniques and messaging infrastructure used to implement the request/response MEP.

4.3.2.3 Event Notification MEP

An event is realized by means of an event notification message exchange that reports a real world effect; specifically, a change in shared state between service participants. The basic event notification MEP takes the form of a one-way message sent by a notifier agent (in this case, the Service component) and received by agents with an interest in the event (here, the Consumer Agent component).

Often the sending agent may not be fully aware of all the agents that will receive the notification; particularly in so-called publish/subscribe (“pub/sub”) situations. In event notification message exchanges, it is rare to have a tightly-coupled link between the sending and the receiving agent(s) for a number of practical reasons. One of the most common is the potential for network outages or communication interrupts that can result in loss of notification of events. Therefore, a third-party agent is usually used that serves as an intermediary that may have the ability to store event notification messages and serves to decouple the sending and received agents.

Although this is typically an implementation issue, because this type of third-party decoupling is so common in event-driven systems**,** we felt that for this Reference Architecture, it was warranted for use in modeling this type of message exchange. This third-party intermediary is shown in Figure 41 as an Event Broker mediator. As with the request/response MEP, no distinction is made between synchronous versus asynchronous communication, although asynchronous message exchange is illustrated in Figure 41.

---

[15] There are cases when a response is not always desired and this would be an example of a “one-way” MEP. Similarly, while not shown here, there are cases when some type of “callback” MEP is required in which the consumer agent is actually exposed as a service itself and is able to process incoming messages from another service.

4.3.3 Composition of Services

Composition of services is the act of aggregating or “composing” a single service from one or more other services. Before we provide an architectural model of service composition, it is important that we distinguish two fundamentally different types of services, *atomic services* and *composite services*.

Atomic Service

A service visible to a service consumer (or agent) via a single interface and described via a single service description that does not use or interact with other services.

Composite Service

A service visible to a service consumer (or agent) via a single interface and described via a single service description that is the aggregation or composition of one or more other services. These other services can be atomic services, other composite services, or a combination of both.[16]

From the consumer’s point of view, the distinction is, of course, mostly irrelevant. The consumer still interacts with a composite service via a single interface and utilizes the meta-level information about the composite service provided by a single Service Description. Nevertheless, there are important dependencies that need to be considered in services that utilize other services such as propagation of policy constraints, security profiles, etc.

A simple model of service composition is illustrated in Figure 42

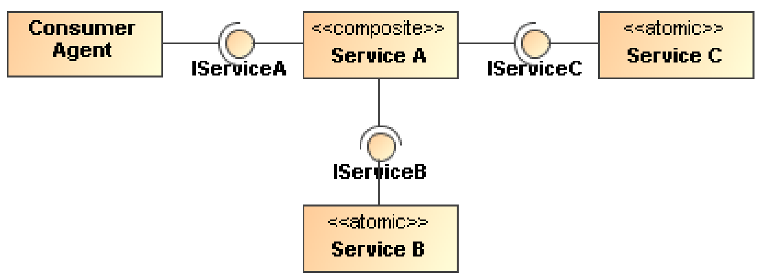


Figure 42 Simple model of service composition ("public” composition).

Here, Service A is a composite service that has an exposed interface IServiceA that is available to the Consumer Agent component and relies on two other service components in its implementation. The Consumer Agent does not know that atomic Services B and C are used by Service A, or whether they are used in serial or parallel, or if their operations succeed or fail. The Consumer Agent only cares about the success or failure of Service A. The exposed interfaces of Services B and C (IService B and IServiceC) are not necessarily hidden from the Consumer Agent; only the fact that these services are used as part of the composition of Service A. In this example, there is no practical reason the Consumer Agent could not interact with Service B or Service C in some other interaction scenario.

It is possible for a service composition to be opaque from one perspective and transparent from another. For example, a service may appear to be a single service from the Consumer Agent’s perspective, but is transparently composed of one or more services from a service management perspective. A Service Management Service needs to be able to have visibility into the composition in order to properly manage the dependencies between the services used in constructing the composite service—including managing the service’s lifecycle. The subject of services as management entities is described and modeled in the Owning Service Oriented Architectures View of this Reference Architecture and will not be further elaborated here. The point to be made here is that there can be different levels of opaqueness or transparency when it comes to visibility of service composition.

Services can be composed in variety of ways including direct service-to-service interaction by using programming techniques, or they can be aggregated by means of a scripting approach that leverages a

[16] The term *composition* as used herein does not embrace the semantics of a UML composition binary relationship. Here we are referring to the relationship between services.

service composition scripting language. Such scripting approaches are further elaborated in the following sub-sections on service-oriented business processes and collaborations.

4.3.3.1 Service-Oriented Business Processes

The concepts of business processes and collaborations in the context of transactions and exchanges across organizational boundaries are described and modeled as part of the Business via Services View of this Reference Architecture (see Section 3). Here, we focus on the belief that the principle of composition of services can be applied to business processes and collaborations. Of course, business processes and collaborations traditionally represent complex, multi-step business functions that may involve multiple participants, including internal users, external customers, and trading partners. Therefore, such complexities cannot simply be ignored when transforming traditional business processes and collaborations to their service-oriented variants.

Business processes are comprised of a set of coherent activities that, when performed in a logical sequence over a period of time and with appropriate rules applied, result in a certain business outcome. Service orientation as applied to business processes (i.e., "service-oriented business processes") means that the aggregation or composition of all of the abstracted activities, flows, and rules that govern a business process can themselves be abstracted as a service **[BLOOMBERG/SCHMELZER]**.

When business processes are abstracted in this manner and accessed through SOA services, all of the concepts used to describe and model composition of services that were articulated in Section 4.3.3 apply. There are some important differences from a composite service that represents an abstraction of a business process from a composite service that represents a single-step business interaction. As stated earlier, business processes have temporal properties and can range from short-lived processes that execute on the order of minutes or hours to long-lived processes that can execute for weeks, months, or even years. Further, these processes may involve many participants. These are important considerations for the consumer of a service-oriented business process and these temporal properties must be articulated as part of the meta-level aspects of the service-oriented business process in its Service Description, along with the meta-level aspects of any sub-processes that may be of use or need to be visible to the Service Consumer.

In addition, a workflow activity represents a unit of work that some entity acting in a described role (i.e., role player) is asked to perform. Activities can be broken down into steps with each step representing a task for the role player to perform. Based on our earlier assertion that messages denote joint action between service participants, we could model these tasks as actions, i.e., message exchanges, which would imply that activities can be modeled as a collection of action-oriented message exchanges. Of course, within a business process, the role player performing a task or sub-task of a particular activity in an overall process flow may actually be a human entity and not a software or hardware agent.

A technique that is used to compose service-oriented business processes that are hierarchical (top-down) and self-contained in nature is known as *orchestration.*

Orchestration

> A technique used to compose hierarchical and self-contained service-oriented business processes that are executed and coordinated by a single agent acting in a "conductor" role.

An orchestration is typically implemented using a scripting approach to compose service-oriented business processes. This typically involves use of a standards-based orchestration scripting language. An example of such a language is the Web Services Business Process Execution Language (WS-BPEL) **[WS-BPEL].** In terms of automation, an orchestration can be mechanized using a business process orchestration engine, which is a hardware or software component (agent) responsible for acting in the role of central conductor/coordinator responsible for executing the flows that comprise the orchestration.

A simple generic example of such an orchestration is illustrated in Figure 43.

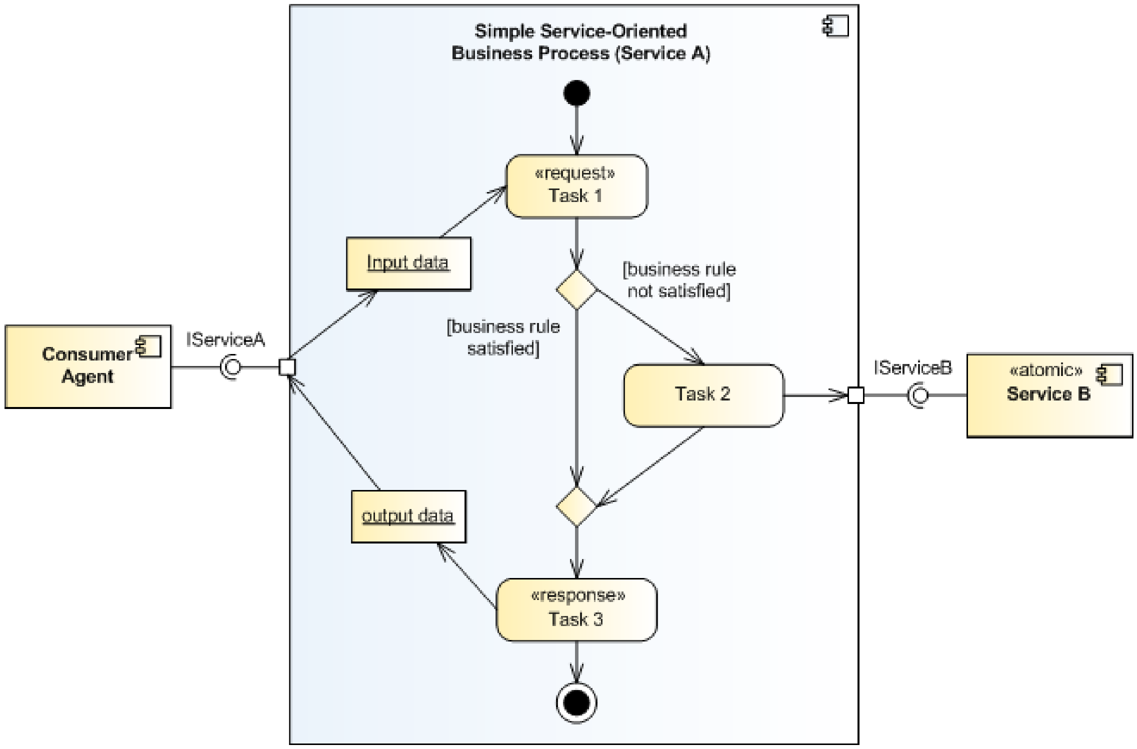


Figure 43 Abstract example of orchestration of service-oriented business process.

Here, we use a UML activity diagram to model the simple service-oriented business process as it allows us to capture the major elements of business processes such as the set of related tasks to be performed, linking between tasks in a logical flow, data that is passed between tasks, and any relevant business rules that govern the transitions between tasks. A task is a unit of work that an individual, system, or organization performs and can be accomplished in one or more steps or subtasks. While subtasks can be readily modeled, they are not illustrated in the orchestration model in Figure 43.

This particular example is based on a request/response MEP and captures how one particular task (Task 2) actually utilizes an externally-provided service, Service B. The entire service-oriented business process is exposed as Service A that is accessible via its externally visible interface, IServiceA.

Although not explicitly shown in the orchestration model above, it is assumed that there exists a software or hardware component, i.e., orchestration engine that executes the process flow. Recall that a central concept to orchestration is that process flow is coordinated and executed by a single conductor agent; hence the name “orchestration.”

4.3.3.2 Service-Oriented Business Collaborations

Turning our attention to business collaborations we note that business collaborations typically represent the interaction involved in executing business transactions, where a *business transaction* is defined in the Business via Services View as “a joint action engaged in by two or more participants in which resources are exchanged” (see Section 3.5.3).

It is important to note that business collaborations represent “peer”-style interactions; in other words, peers in a business collaboration act as equals. This means that unlike the orchestration of business processes, there is no single or central entity that coordinates or “conducts” a business collaboration. These peer styles of interactions typically occur between trading partners that span organizational boundaries.

Similar to service-enablement of business processes, business collaborations can also be service-enabled. For purposes of this Reference Architecture, we refer to these types of business collaborations as “service-oriented business collaborations.” Of course, unlike service-oriented business processes, the concept of service-oriented business collaborations does not necessarily imply exposing the entire peer-style business collaboration as a service itself but rather the collaboration uses service-based interchanges.

The technique that is used to compose service-oriented business collaborations in which multiple parties collaborate in a peer-style as part of some larger business transaction by exchanging messages with trading partners and external organizations (e.g., suppliers) is known as *choreography* **[NEWCOMER/LOMOW]**.

Choreography

> A technique used to characterize and to compose service-oriented business collaborations based on ordered message exchanges between peer entities in order to achieve a common business goal.

Choreography differs from orchestration primarily in that each party in a business collaboration describes its part in the service interaction in terms of public message exchanges that occur between the multiple parties as standard atomic or composite services, rather than as specific service-oriented business processes that a single conductor/coordinator (e.g., orchestration engine) executes. Note that choreography as we have defined it here should not be confused with the term *process choreography*, which is defined in the Business via Services View as "the description of the possible interactions that may take place between two or more participants to fulfill an objective." This is an example of domain-specific nomenclature that often leads to confusion and why we are making note of it here.

As is the case of an orchestration, a choreography is typically implemented by using a scripting approach to composing service-oriented business collaborations. This typically involves use of a standards-based choreography scripting language. An example of such a language is the Web Services Choreography Description Language **[WS-CDL]**.

A simple generic example of a choreography is illustrated in Figure 44.

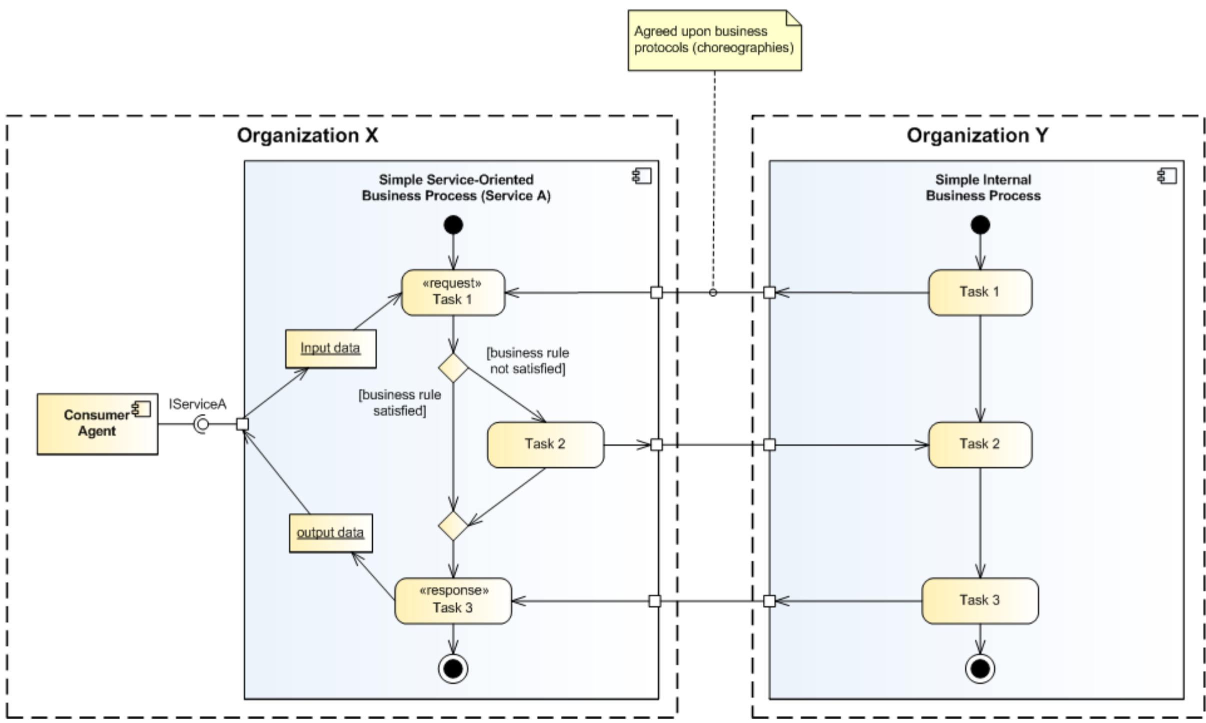


Figure 44 Abstract example of choreography of service-oriented business collaboration.

This example, which is a variant of the orchestration example illustrated earlier in Figure 43 adds trust boundaries between two organizations; namely, Organization X and Organization Y. It is assumed that these two organizations are peer entities that have an interest in a business collaboration, for example, Organization X and Organization Y could be trading partners. Organization X retains the service-oriented business process Service A, which is exposed to internal consumers via its provided service interface, IServiceA. Organization Y also has a business process that is involved in the business collaboration;

however, for this example, it is an internal business process that is not exposed to potential consumers either within or outside its organizational boundary.

The scripting language that is used for the choreography needs to define how and when to pass control from one trading partner to another, i.e., Organization X and Organization Y. Defining the business protocols used in the business collaboration involves precisely specifying the visible message exchange behavior of each of the parties involved in the protocol, without revealing internal implementation details **[NEWCOMER/LOMOW]**.

If,a peer-style business collaboration in which visibility into and use of each participating organization's internal service-oriented business processes was necessary as part of an end-to-end business transaction, then it would be desirable to select a choreography scripting language that would support interaction between different orchestration engines that spans organizational boundaries. WS-CDL is an example of such a language.

4.4 Policies and Contracts Model

As described in the Reference Model, a policy is the representation of a constraint or condition on the use, deployment, or description of an owned entity as defined by any participant. A contract is a representation of an agreement between two or more participants. Technically, the only difference between a policy and a contract is the agreement between two or more parties to a contract and the enforceability of a policy by one party on other parties.

In Section 4.4.1, Policies and contracts are discussed in the context of the Business via Services View with generalizations about IT mechanisms in support of the view. Section 4.4.2 breaks down a core aspect of policies, a proposition, and provides the basis for the IT mechanisms discussed in Section 4.4.3. Section 4.4.4 concludes with some general policy and contract principles common to SOA policies.

4.4.1 Automating Support for Policies and Contracts

Policy and contract IT mechanisms support automated governance and management within the SOA ecosystem to improve governance and management efficiency. Understanding the complete environment which policies and contracts apply in a SOA requires understanding of the processes surrounding policies and contracts in the social structure, the IT mechanisms that support automated enforcement of policies and contracts, and the traversal from/to the social structure to/from the IT policy automation mechanisms. The architecture SHOULD provide mechanisms to enforce policies and contracts to ensure efficient operations consistent with the goals of the social structure.

Figure 45 derives from Section 3, Business via Services View. Core aspects of policies and contracts are the propositions, the owners, and the measurement and enforcement of the policy or contract. In Section 3.8, Proposition Model, measurable assertions and commitments are characterized as propositions - an expression of some property of the world whose truth can be measured by examining the world and checking that the expression and the world are consistent with each other. Assertions are claims about current state while commitments are agreements to future state.

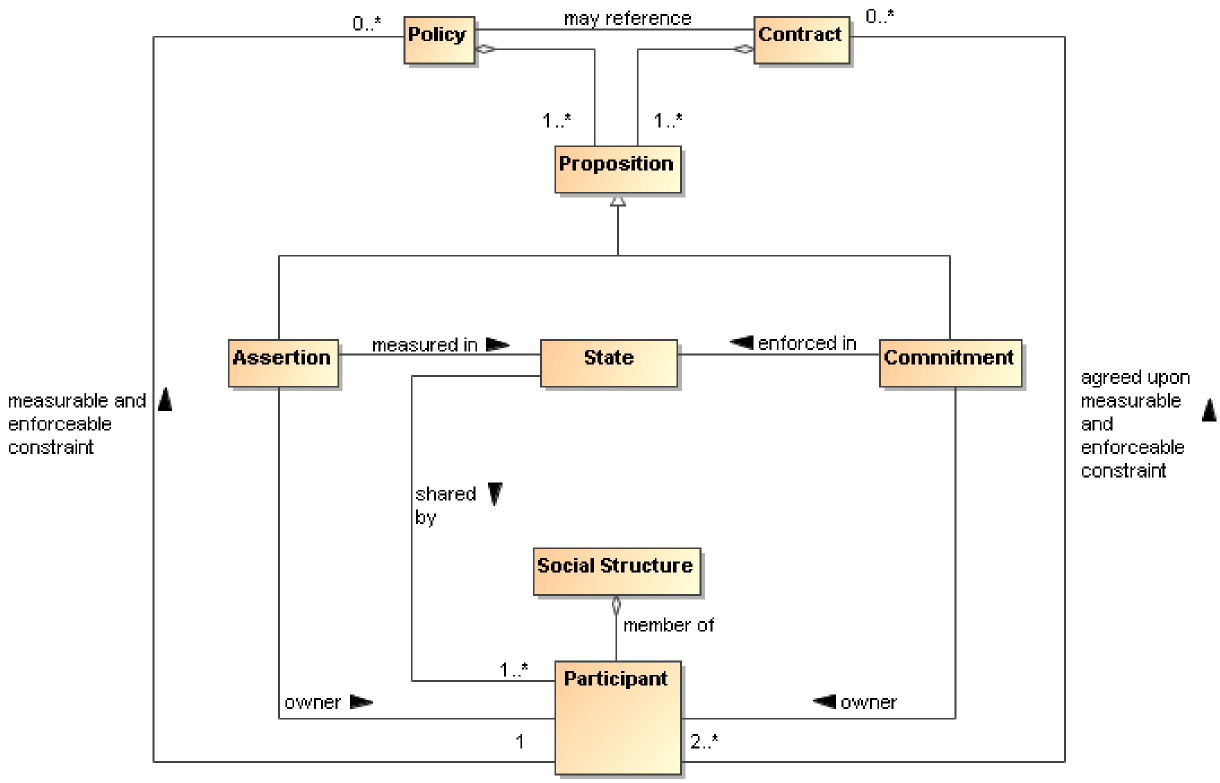


Figure 45 Distinguishing between policies and contracts

In a business context, contracts are legally binding agreements between two or more parties. A contract is formed when there is an offer that is duly made and the offer is accepted and there is evidence that indicates there was a tangible exchange of value between the two parties. While this Reference Architecture is inclusive of legally binding contracts for a SOA, contracts do not always have to be legally binding agreements.

A contract may include references to policies and other contracts while a policy may include references to contracts and other policies. For example, a contract may reference a set of policies and a policy may prioritize certain contracts over others.

The measurability and enforcement of propositions may include many indirectly related participants within the social structure. Dispute resolutions, for example, may involve courts.

From the IT perspective, high level policies and contracts are translated into low level rules and measurable properties. For low level rules and measurable properties, both contracts and policies are likely to be enforced by the same type of IT policy mechanisms.

Policies and contracts have wide applicability within the Reference Architecture. They are used to express security policies, service policies, relationships and constraints within the social structures that encapsulate service participants, management of services and many other instances. The enforcement of a policy or contract may be a part of the SOA-based computing environment or it may be handled outside of the SOA-based computing environment. The Reference Architecture is concerned with the underlying IT mechanisms and principles that support enforceable and measurable contracts and policies in the widest range of situations for a SOA.

4.4.2 Policy and Contract Types

Figure 46 depicts assertions and commitments as an aggregation of measurable constraints. We can analyze policy and contract constraints in a number of dimensions: positive constraints vs. negative constraints; and permission-style vs. obligation-style constraints.

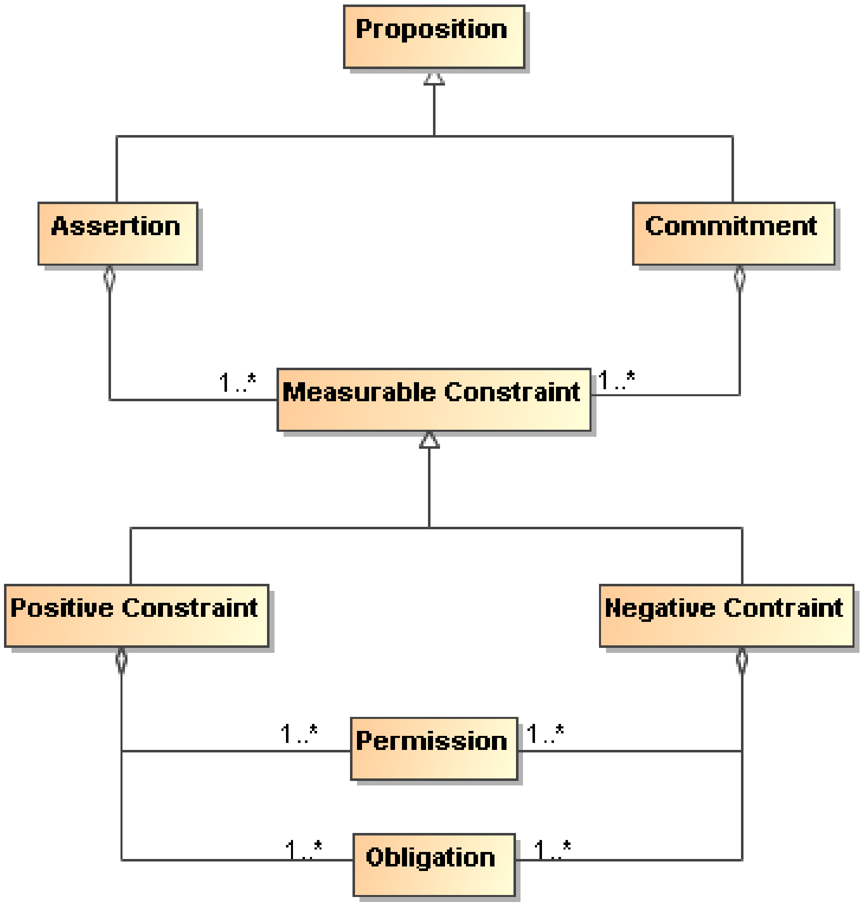


Figure 46 Policy and Contract Constraints

Positive constraints are about the things that you may/should do and negative constraints are about the things that you should not do. A permission-style constraint is about the right to access some resource or perform some action; an obligation-style constraint is about the requirement to perform some action or maintain the state of a resource.

These are combinable, in the sense that you may have a positive permission constraint (for example, you may use encryption in your messages), whereas a negative permission constraint indicates that there is something you may not do. Similarly, a positive obligation may be something like you must keep the balance of your account positive; whereas an example of a negative obligation may be that the bank will not cover a check for more than the balance in your account.

Permission-style constraints are often checkable a-priori: before the intended action or access is completed the current permission constraints may be applied to deny the access if necessary. However, obligation-style constraints can normally only be verified post-priori. Permission constraints are sometimes referred to as access control policies given the preponderance of security-related policies in many applications. One use of obligation constraints is for metrics collection and compliance.

Policies and contracts can contain a mix of permissions and obligations, and, in sufficiently rich policy management frameworks, can be combined in interesting ways: for example, you may be obliged to give permission to certain actions; or you may be permitted to enter into obligations (this is the core of the right to enter into contracts).

The mechanism for enforcing a permission-oriented constraint is typically prevention at the point of action. The mechanisms for enforcing obligation constraints are typically achieved by a combination of auditing and remedial action.

4.4.3 IT Mechanisms Supporting Policies and Contracts

A common phenomenon of many machines and systems is that they are much broader in their potential than is actually needed for a particular circumstance. As a result, the behavior and performance of the system tend to be under-constrained by the implementation. Policy statements define the choices that a service provider and/or service consumer (or other stakeholder) makes; these choices are used to guide the actual behavior of the system to the desired behavior and performance.

While there are many possible approaches to the realization of policy/contracts for a SOA, one approach based on current policy standardization efforts is depicted in this section. The common policy architectural elements that are provided in this section are based on the minimal mechanisms required to provide policy guided delivery across distributed services within an ownership domain and across ownership domains.

4.4.3.1 Permission Based Policy and Contract Mechanisms

For IT mechanisms, policies and contracts are measurable and enforceable rules that define choices in the behavior of a system. Contracts are the set of rules that define the agreements under which service functionality is delivered. Figure 47 depicts mechanisms in support of permission style policy requests where the measurement of rules occurs in decision procedures identified by a Decision Point mechanism in the diagram.

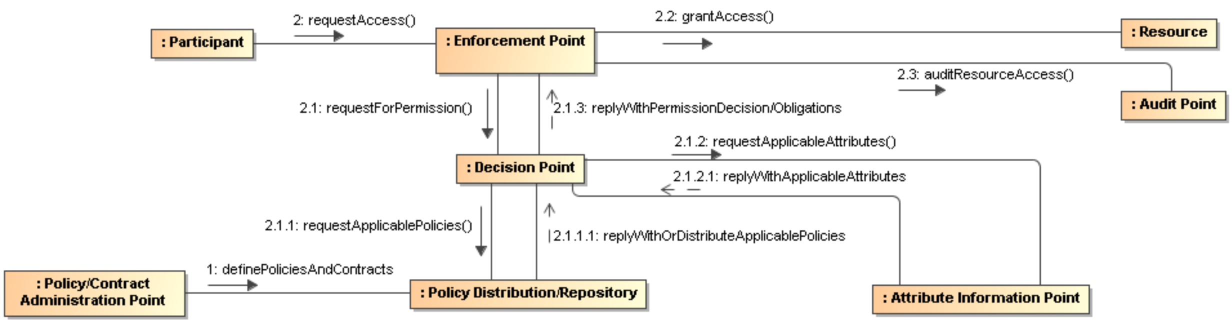


Figure 47 Permission Policy Mechanisms

Policy/Contract Administration Point

A Policy/Contract Administration Point is the mechanism for a SOA that allows a participant to administer policies for storage and/or distribution. There can be many enterprise SOA policy/contract administration capabilities and the Policy/Contract Administration Point is a generalization for any of these type of capabilities.

Policy Distribution/Repository

The Policy Distribution/Repository distributes policy to decision points or stores policies for retrieval by decision points.

Attribute Information Point

The Attribute Information Point is responsible for collecting and forwarding attributes to the Decision Point. Attributes are named values that define characteristics of participants, resources, actions, or the environment. Attributes are defined in the Service Description Model in Section 4.1.

Audit Point

In Figure 33, the Audit Point is any mechanism that records participant actions requiring permission decisions or records the measurement results for obligations discussed in Section 4.4.3.2. An auditing mechanism may store audited information and/or provide event notifications of audited information. Auditing may be used for activities like forensic investigation and regulatory compliance.

Resource

A resource is any entity of some perceived value. Resources are defined in the Resource Model in Section 3.2.

Decision Point

The Decision Point evaluates participant requests against relevant policies/contracts and attributes to render a permission decision. The Decision Point provides a measurement for an assertion. The Decision Point generally renders a permission decision in the form of permit, deny, indeterminate, not applicable, or a set of obligations. A Decision Point may obtain a permission decision from a computing mechanism or from outside the computing system, decisions by people through workflow for example.

Enforcement Point

The Enforcement Point enforces and assures the Decision Point decisions and obligations. In a Service Oriented Architecture, one policy or contract may be applicable to multiple distributed services. Due to the distributed nature of a SOA, the enforcement of permission decisions is attributed to an Enforcement Point that is separate from the Decision Point. One Decision Point can provide decisions for many distributed Enforcement Points.

For permission decisions, the Enforcement Point often performs enforcement in the form of protecting access and determining access compliance to one or more resources. When attempting to access a resource, the Enforcement Point sends a description of the attempted access to a Decision Point. The Decision Point evaluates the request against its available policies/contracts and produces a permission decision that is returned to the Enforcement Point. Like the Decision Point, an Enforcement Point may require a means of enforcement outside the computing system.

4.4.3.2 Obligation Based Policy and Contract Mechanisms

In Figure 48, the Enforcement Point creates or uses a mechanism for measuring policy obligations. Just as it is the responsibility of the Enforcement Point to ensure permission decisions, it is the responsibility of the Enforcement Point to ensure that policy obligations are met. This may require a one time measurement or ongoing monitoring of the obligation. For example, there may be the contractual obligation to allocate a certain level of bandwidth for a customer's transactions. The contractual obligation may also require ongoing monitoring to ensure the customer's transactions do not exceed allotted bandwidth and if exceeded, the provider may happily levy exorbitant over usage fees.

While Figure 48 depicts measurement of obligations based on an access request, the Enforcement Point may acquire policy obligations independent of permission requests from other participants. To provide a real-world analogy, a consciences taxicab owner may have a policy that taxis not operate when the roads are icy. At the start of a working day, the roads are clear but the forecast is for possible icy conditions later in the day. A dispatcher, a designated Enforcement Point, asks the owner, a Decision Point, whether they should send taxicabs out for the day. The owner says yes as long as the weather reports do not indicate there could be icy roads. The dispatcher checks a website which provides registry listings of service providers that provide reports for local road conditions. The dispatcher chooses a local traffic reporting service, a Measurement Point, that will send traffic reports via email about the road conditions. The dispatcher goes on with his job not worried about checking weather conditions, correctly or incorrectly relying on the email notification to meet the taxicab company's obligation as to the safety of its drivers.

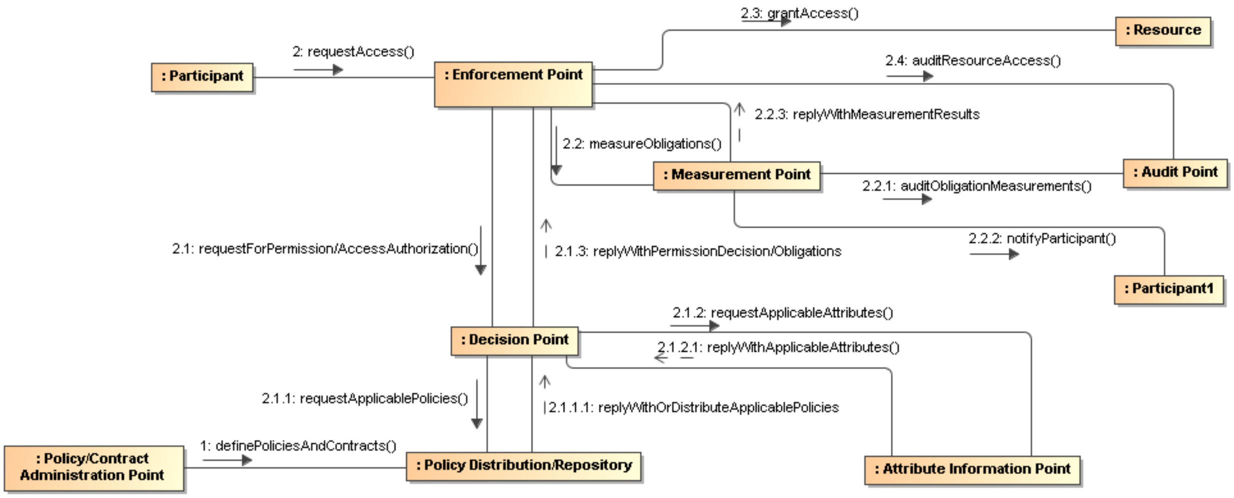


Figure 48 Obligation Policy Mechanisms

Measurement Point

The Measurement Point identifies mechanisms for measuring and monitoring policy obligations.

The Measurement Point in Figure 48 receives and responds to the Enforcement Point requests to measure policy obligations. The Measurement Point may also audit and provide event notifications of obligation measurements.

In Figure 48, the Measurement Point can be used to collect metrics and report those metrics to the Audit Point. Metrics may be used to verify compliance either in an automated fashion or at a later point in time. If compliance is automated, then the Measurement Point may adjust the behavior of the system in accordance with compliance policies or contracts.

4.4.4 Policy and Contract Principles

In the realization of policies and contracts for a SOA, there are common policy principles that will be encountered in many of the standards and/or technology choices used for the realization. Some of these common principles are covered in this section.

4.4.4.1 Policies and Contracts Goals

Policies SHOULD reflect the goals of governance or management processes, see Section 5.1 Governance of Service Oriented Architectures and section 5.3 Services as Managed Entities Model. The governance and management processes SHOULD use formal and standardized policy languages to enable the widest possible understanding and use of stated policies and contracts, and architecture components SHOULD be available to enable compliance.

4.4.4.2 Policy and Contract Specification

The language used to describe policies and contracts inevitably constrains the forms and types of policies and contracts expressible in the description. Formal policy language definitions are outside the scope of this specification. For formal policy languages, standard specifications such as XACML and WS-Policy may be referenced. Policy/Contract descriptions may be associated with a service through the Service Description as defined in Section 4.1 Service Description Model.

Regardless of the language used to describe policies and contracts, there are certain aspects to capture in any system for the representation of policies and contracts such as:

- how to describe atomic policy constraints
- how to nest policy constraints allowing for abstractions and refinements of a policy constraint

- how to reference policy constraints allowing for the reuse of a policy constraint
- how to define alternative policy constraints for the selection of compatible policy constraints between the consumer and provider
- policy versioning
- policy modules

4.4.4.3 Policy Composition

Multiple policies may be defined for one or more services in one or more ownership domains. The application of policies and contracts over distributed services requires the ability to compose one or more policies into an overarching policy. The composition of policies may be implemented as a hierarchy or nesting and/or it can be implemented as intersections and unions of sets.

4.4.4.4 Conflict Resolution

The analysis of policy rules may result in conflicts between the policy rules. There can be many causes for policy conflicts such as conflicting policy rules between ownership domains and policy language specifications that do not convert to first order predicate logic for IT policy mechanisms. This can cause policy decision results to be indeterminate. Policy administration mechanisms may provide conflict resolution capabilities prior to the storage/distribution of policies. At run time, conflicts may propagate to higher authorities inside or outside the SOA-based IT mechanisms.

4.4.4.5 Delegation of Policy

Policy authorization may be delegated to agents acting on behalf of a client to enable decentralized policy administration and/or policy enforcement. This allows policies to be administered and/or enforced in a hierarchical fashion. Policies may also be transferred to an agent or resource to effectively allow that agent or resource to separate from an ownership domain. The agent or resource may join another ownership domain or rejoin the same ownership domain at a later time.

4.4.5 Architectural Implications

While policy and contract descriptions have much of the same architectural implications as described in Service Description, languages and mechanisms supporting policies and contracts also have the following architectural implications:

- Policy and Contract language specifications will typically provide support for the following capabilities:
    - expression of assertion and commitment policy constraints;
    - expression of positive and negative policy constraints;
    - expression of permission and obligation policy constraints;
    - nesting of policy constraints allowing for abstractions and refinements of a policy constraint;
    - definition of alternative policy constraints to allow for the selection of compatible policy constraints for a consumer and provider;
    - composition of policies to combine one or more policies.
- Policy and contract mechanisms in a SOA ecosystem will require the following capabilities:
    - decision procedures which must be able to measure and render decisions on constraints;
    - enforcement of decisions;
    - measurement and notification of obligation constraints;
    - auditability of decisions, enforcement, and obligation measurements;
    - administration of policy and contract language artifacts;
    - storage of policies and contracts;
    - distribution of policies/contracts;
    - conflict resolution or elevation of conflicts in policy rules;
    - delegation of policy authority to agents acting on behalf of a client;
    - decision procedures capable of incorporating roles and/or attributes for rendered decisions.

5 Owning Service Oriented Architectures View

In the absence of policy-based governance, organizations will operate as unruly collection of factions that pull in opposing directions.
Paul A. Strassmann

The *Owning Service Oriented Architectures View* focuses on the issues, requirements and responsibilities involved in owning a SOA-based system.

Owning a SOA-based system raises significantly different challenges to owning other complex systems -- such as Enterprise suites -- because there are strong limits on the control and authority of any one party when a system spans multiple ownership domains.

Even when a SOA-based system is deployed internally within an organization, there are multiple internal stakeholders involved and there may not be a simple hierarchy of control and management.

This view focuses on the Governance of SOA-based systems, on the security challenges involved in running a SOA-based system and the management challenges.

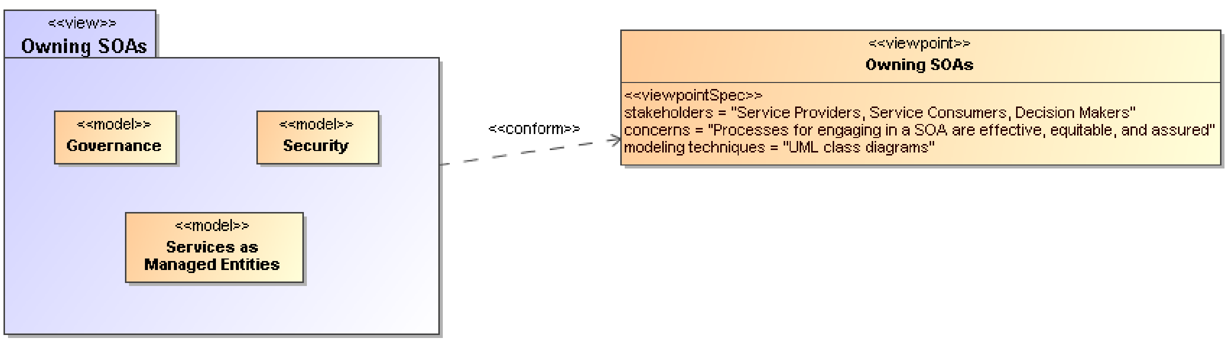


Figure 49 Model elements described in the Owning Service Oriented Architectures view

The following subsections present models of these functions.

5.1 Governance Model

The SOA-RM defines Service Oriented Architecture as an architectural paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains **[SOA-RM]**. Consequently, it is important that organizations that plan to engage in service interactions adopt governance policies and procedures sufficient to ensure that there is standardization across both internal and external organizational boundaries to promote the effective creation and use of SOA-based services.

5.1.1 Understanding Governance

5.1.1.1 Terminology

Governance is about making decisions that are aligned with the overall organizational strategy and culture of the enterprise. **[Gartner]** It specifies the decision rights and accountability framework to encourage desirable behaviors **[Weill/Ross-MIT Sloan School]** towards realizing the strategy and defines incentives (positive or negative) towards that end. It is less about overt control and strict adherence to rules, and more about guidance and effective and equitable usage of resources to ensure sustainability of an organization's strategic objectives. **[Open Group]**

To accomplish this, governance requires organizational structure and processes and must identify who has authority to define and carry out its mandates. It must address the following questions: 1) what decisions must be made to ensure effective management and use?, 2) who should make these decisions?, and 3) how will these decisions be made and monitored? The intent is to achieve goals, add value, and reduce risk.

Within a single ownership domain such as an enterprise, generally there is a hierarchy of governance structures. Some of the more common enterprise governance structures include corporate governance, technology governance, IT governance, and architecture governance **[TOGAF v8.1]**. These governance structures can exist at multiple levels (global, regional, and local) within the overall enterprise.

It is often asserted that SOA governance is a specialization of IT governance as there is a natural hierarchy of these types of governance structures; however, the focus of SOA governance is less on decisions to ensure effective management and use of IT as it is to ensure effective management and use of SOA-based systems. Certainly, SOA governance must still answer the basic questions also associated with IT governance, i.e., who should make the decisions, and how these decisions will be made and monitored.

5.1.1.2 Relationship to Management

There is often confusion centered on the relationship between governance and management. As described earlier, governance is concerned with decision making. Management, on the other hand, is concerned with execution. Put another way, governance describes the world as leadership wants it to be; management executes activities that intends to make the leadership's desired world a reality. Where governance determines who has the authority and responsibility for making decisions and the establishment of guidelines for how those decisions should be made, management is the actual process of making, implementing, and measuring the impact of those decisions **[Loeb]**. Consequently, governance and management work in concert to ensure a well-balanced and functioning organization as well as an ecosystem of inter-related organizations. In the sections that follow, we elaborate further on the relationship between governance and management in terms of setting and enforcing service policies, contracts, and standards as well as addressing issues surrounding regulatory compliance.

5.1.1.3 Why is SOA Governance Important?

One of the hallmarks of SOA that distinguishes it from other architectural paradigms for distributed computing is the ability to provide a uniform means to offer, discover, interact with and use capabilities (as well the ability to compose new capabilities from existing ones) all in an environment that transcends domains of ownership. Consequently, ownership, and issues surrounding it, such as obtaining acceptable terms and conditions (T&Cs) in a contract, is one of the primary topics for SOA governance. Generally, IT governance does not include T&Cs, for example, as a condition of use as its primary concern.

Just as other architectural paradigms, technologies, and approaches to IT are subject to change and evolution, so too is SOA. Setting policies that allow change management and evolution, establishing strategies for change, resolving disputes that arise, and ensuring that SOA-based systems continue to fulfill the goals of the business are all reasons why governance is important to SOA.

5.1.1.4 Governance Stakeholders and Concerns

As noted in Section 3.1, the participants in a service interaction include the service provider, the service consumer, and other interested or unintentional third parties. Depending on the circumstances, it may also include the owners of the underlying capabilities that the SOA services access. Governance must establish the policies and rules under which duties and responsibilities are defined and the expectations of participants are grounded. The expectations include transparency in aspects where transparency is mandated, trust in the impartial and consistent application of governance, and assurance of reliable and robust behavior throughout the SOA ecosystem.

5.1.2 A Generic Model for Governance

The following is a generic model of governance represented by segmented models that begin with motivation and proceed through measuring compliance. A given enterprise may already have portions of these models in place. To a large extent, the models shown here are not specific to SOA; discussions on direct applicability begin in section 5.1.3.

5.1.2.1 Motivating Governance

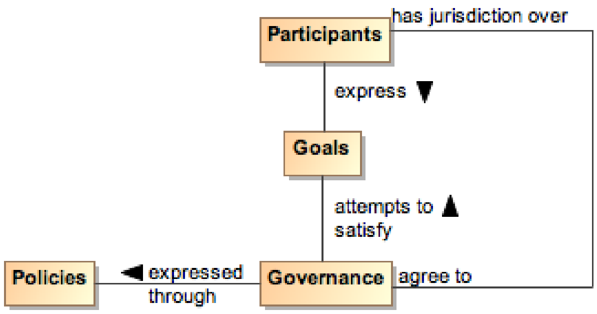


Figure 50 Motivating governance model

An organizational domain such as an enterprise is made up of Participants who may be individuals or groups of individuals forming smaller organizational units within the enterprise. The overall business strategy should be consistent with the Goals of the participants; otherwise, the business strategy would not provide value to the participants and governance towards those ends becomes difficult if not impossible. For governance to have effective jurisdiction over participants, there must be some degree of agreement by each participant that it will abide by the governance mandates. A minimal degree of agreement often presages participants who “slow-roll” if not actively reject complying with Policies that express the specifics of governance.

5.1.2.2 Setting Up Governance

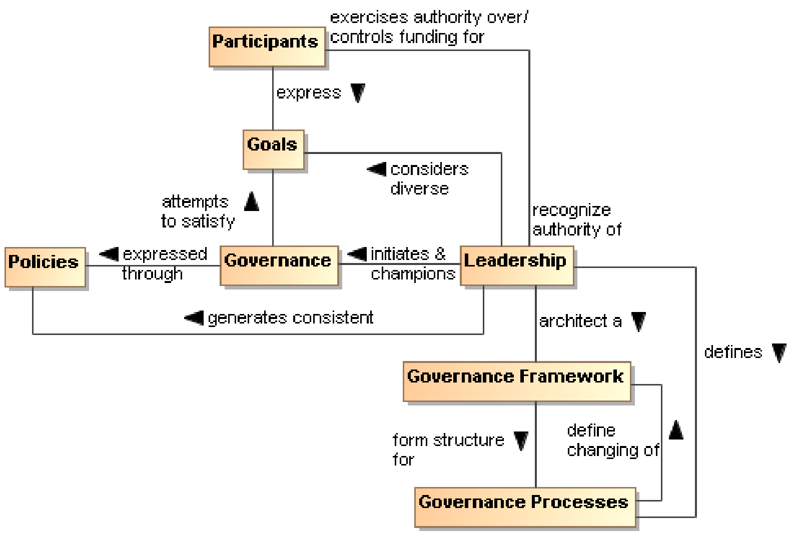


Figure 51 Setting up governance model

As noted earlier, governance requires an appropriate organizational structure and identification of who has authority to make governance decisions. In the above figure, the entity with governance authority is designated the Leadership. This is someone that Participants recognize as having authority and who typically has some control over the Participants.

The Leadership is responsible for prescribing or delegating a working group to prescribe the Governance Framework that forms the structure for Governance Processes that define how governance is to be carried out. This does not itself define the specifics of how governance is to be applied, but it does provide an unambiguous set of procedures that should ensure consistent actions which Participants agree are fair and account for sufficient input on the subjects to which governance will be applied. Note that the Governance Processes should also include those necessary to modify the Governance Framework itself. The Governance Processes are likely reviewed and agreed to by the Participants.

The Governance Framework and Processes are often documented in the charter of a body created or designated to oversee governance. This is discussed further in the next section.

An important function of Leadership is not only to initiate but also be the consistent champion of governance. Those responsible for carrying out governance mandates must have Leadership who makes it clear to Participants that expressed Policies are seen as a means to realizing established goals and that compliance with governance is required.

5.1.2.3 Carrying Out Governance

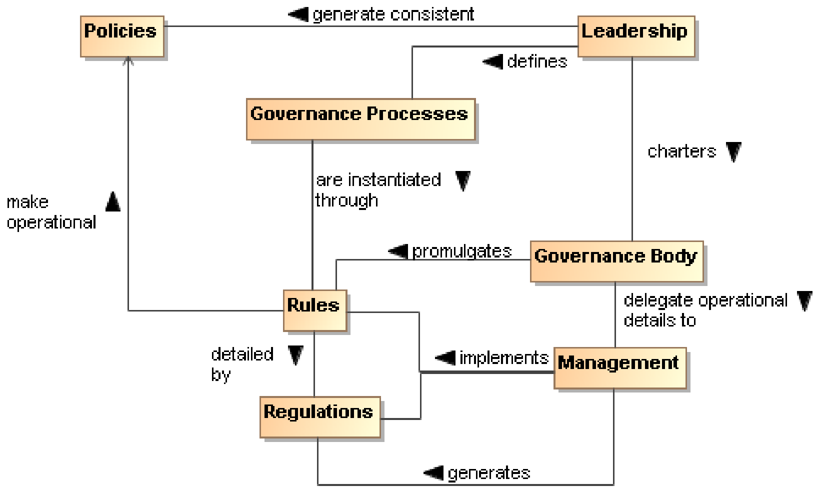


Figure 52 Carrying Out Governance Model

To carry out governance, Leadership charters a Governance Body to promulgate the Rules needed to make the Policies operational. The Governance Body acts in line with Governance Processes for its rule-making process and other functions. Whereas Governance is the setting of Policies and defining the Rules that provide an operational context for Policies, the operational details of governance are likely delegated by the Governance Body to Management. Management generates Regulations that specify details for Rules and other procedures to implement both Rules and Regulations. For example, Leadership could set a policy that all authorized parties should have access to data, the Governance Body would promulgate a Rule that PKI certificates are required to establish identity of authorized parties, and Management can specify who it deems to be a recognized PKI issuing body.

Whereas the Governance Framework and Processes are fundamental for having Participants acknowledge and commit to compliance with governance, the Rules and Regulations provide operational constraints which may require resource commitments or other levies on the Participants. It is important for Participants to consider the framework and processes to be fair, unambiguous, and capable of being carried out in a consistent manner and to have an opportunity to formally accept or ratify this situation. Rules and Regulations, however, do not require individual acceptance by any given participant although some level of community comment is likely to be part of the Governance Processes. Having agreed to governance, the Participants are bound to comply or be subject to prescribed mechanisms for enforcement.

5.1.2.4 Ensuring governance compliance

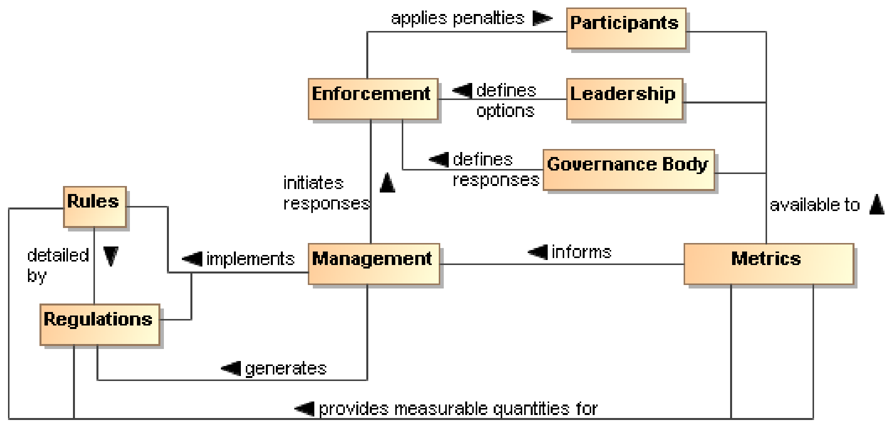


Figure 53 Ensuring governance compliance model

Setting Rules and Regulations does not ensure effective governance unless compliance can be measured and Rules and Regulations can be enforced. Metrics are those conditions and quantities that can be measured to characterize actions and results. Rules and Regulations MUST be based on collected Metrics or there will be no way for Management to assess compliance. The Metrics are available to the Participants, the Leadership, and the Governance Body so what is measured and the results of measurement are clear to everyone.

The Leadership in its relationship with Participants will have certain options that can be used for Enforcement. A common option may be to effect future funding. The Governance Body defines specific enforcement responses, such as what degree of compliance is necessary for full funding to be restored. It is up to Management to identify compliance shortfalls and to initiate the Enforcement process.

Note, enforcement does not strictly need to be negative. Management can use Metrics to identify exemplars of compliance and Leadership can provide options for rewarding the Participants. It is likely the Governance Body that defines awards or other incentives.

5.1.3 Governance Applied to SOA

5.1.3.1 Where SOA Governance is Different

Governance in the context of SOA is that organization of services that promotes their visibility, that facilitates interaction among service participants, and that enforces that the results of service interactions are those real world effects as described within the service description and constrained by policies and contracts as assembled in the execution context.

SOA governance must specifically account for control across different ownership domains, i.e. all the participants may not be under the jurisdiction of a single governance authority. However, for governance to be effective, the participants must agree to recognize the authority of the Governance Body and must operate within the Governance Framework and through the Governance Processes so defined.

Being distributed and representing different ownership domains, a SOA participant is likely under the jurisdiction of multiple governance domains simultaneously and may individually need to resolve consequent conflicts. The governance domains may specify precedence for governance conformance or it may fall to the discretion of the participant to decide on the course of actions they believe appropriate.

SOA governance must account for interactions across ownership boundaries, which likely also implies across enterprise governance boundaries. For such situations, governance emphasizes the need for agreement that some Governance Framework and Governance Processes has jurisdiction, and the governance defined must satisfy the Goals of the Participants for cooperation to continue. A standards development organization such as OASIS is an example of voluntary agreement to governance over a limited domain to satisfy common goals.

The specifics discussed in the figures in the previous sections are equally applicable to governance across ownership boundaries as it is within a single boundary. There is a charter agreed to when Participants become members of the organization, and this charter sets up the structures and processes that will be followed. Leadership may be shared by the leadership of the overall organization and the leadership of individual groups themselves chartered per the Governance Processes. There are Rules/Regulations specific to individual efforts for which Participants agree to local goals, and Enforcement can be loss of voting rights or under extreme circumstances, expulsion from the group.

Thus, the major difference for SOA governance is an appreciation for the cooperative nature of the enterprise and its reliance on furthering common goals if productive participation is to continue.

5.1.3.2 What Must be Governed

An expected benefit of employing SOA principles is the ability to quickly bring resources to bear to deal with unexpected and evolving situations. This requires a great deal of confidence in the underlying capabilities that can be accessed and in the services that enable the access. It also requires considerable flexibility in the ways these resources can be employed. Thus, SOA governance requires establishing confidence and trust while instituting a solid framework that enables flexibility, indicating a combination of strict control over a limited set of foundational aspects but minimum constraints beyond those bounds.

SOA governance applies to three aspects of service definition and use:

- SOA infrastructure – the “plumbing” that provides utility functions that enable and support the use of the service
- Service inventory – the requirements on a service to permit it to be accessed within the infrastructure
- Participant interaction – the consistent expectations with which all participants are expected to comply

5.1.3.2.1 Governance of SOA infrastructure

The SOA infrastructure is likely composed of several families of SOA services that provide access to fundamental computing business services. These include, among many others, services such as messaging, security, storage, discovery, and mediation. By characterizing the environment as containing families of SOA services, the assumption is that there may be multiple approaches to providing the business services or variations in the actual business services provided. For example, discovery could be based on text search, on metadata search, on approximate matches when exact matches are not available, and numerous other variations. The underlying implementation of search algorithms are not the purview of SOA governance, but the access to the resulting service infrastructure enabling discovery must be stable, reliable, and extremely robust to all operating conditions. Such access enables other specialized SOA services to use the infrastructure in dependable and predictable ways, and is where governance is important.

5.1.3.2.2 Governance of the service inventory

Given an infrastructure in which other SOA services can operate, a key governance issue is which SOA services to allow in the ecosystem. The major concern SHOULD be a definition of well-behaved services, where the required behavior will likely inherit their characteristics from experiences with distributed computing but will also evolve with SOA experience. A major requirement for ensuring well-behaved services is collecting sufficient metrics to know how the service affects the SOA infrastructure and whether it complies with established infrastructure policies.

Another common concern of service approval is whether there will be duplication of function by multiple services. Some governance models talk to a tightly controlled environment where a primary concern is to avoid any service duplication. Other governance models talk to a market of services where the consumers have wide choices. For the latter, it is anticipated that the better services will emerge from market consensus and the availability of alternatives will drive innovation.

It is likely that some combination of control and openness will emerge, possibly with a different appropriate balance for different categories of use. The governance issue for allowable services is in identifying the required attributes to adequately describe a service, the required target values of the attributes, and the standards for defining the meaning of the attributes and their target values. Governance may also specify the processes by which the attribute values are measured and the corresponding certification that some realized attribute set may imply.

For example, unlimited access for using a service may require a degree of life cycle maturity that has demonstrated sufficient testing over a certain size community. Alternately, the policy may specify that a service in an earlier phase of its life cycle may be made available to a smaller, more technically sophisticated group in order to collect the metrics that would eventually allow the service to advance its life cycle status.

This aspect of governance is tightly connected to description because, given a well-behaved set of services, it is the responsibility of the consumer (or policies promulgated by the consumer's organization) to decide whether a service is sufficient for that consumer's intended use. The goal is to avoid global governance specifying criteria that are too restrictive or too lax for the local needs of which global governance has little insight.

Such an approach to specifying governance allows independent domains to describe services in local terms while still having the services available for informed use across domains. In addition, changes to the attribute sets within a domain can be similarly described, thus supporting the use of newly described resources with the existing ones without having to update the description of all the legacy content.

5.1.3.2.3 Governance of participant interaction

Finally, given a reliable services infrastructure and a predictable set of services, the third aspect of governance is prescribing what is required during a service interaction. Governance would specify adherence to service interface and service reachability parameters and would require that the result of an interaction MUST correspond to the real world effects as contained in the service description. It would also rely on sufficient monitoring by the SOA infrastructure to ensure services remain well-behaved during interactions, e.g. do not use excessive resources or exhibit other prohibited behavior. Governance would also require that policy agreements as documented in the execution context for the interaction are observed and that the results and any after effects are consistent with the agreed policies. It is likely that in this area the governance will focus on more contractual and legal aspects rather than the precursor descriptive aspects. SOA governance may prescribe the processes by which SOA-specific policies are allowed to change, but there are likely more business-specific policies that will be governed by processes outside SOA governance.

5.1.3.3 Overarching governance concerns

There are numerous governance related concerns whose effects span the three areas just discussed. One is the area of standards, how these are mandated, and how the mandates may change. The Web Services standards stack is an example of relevant standards where a significant number are still under development. In addition, while there are notional scenarios that guide what standards are being developed, the fact that many of these standards do not yet exist precludes operational testing of their adequacy or effectiveness as a necessary and sufficient set.

That said, standards are critical to creating a SOA ecosystem where SOA services can be introduced, used singularly, and combined with other services to deliver complex business functionality. As with other aspects of SOA governance, the Governance Body should identify the minimum set felt to be needed and rigorously enforce that that set be used where appropriate. The Governance Body must take care to expand and evolve the mandated standards in a predictable manner and with sufficient technical guidance that new services will be able to coexist as much as possible with the old, and changes to standards do not cause major disruptions.

Another area that may see increasing activity as SOA expands will be additional regulation by governments and associated legal institutions. New laws are likely that will deal with transactions which are service based, possibly including taxes on the transactions. Disclosures laws are likely to mandate certain elements of description so both the consumer and provider act in a predictable environment and are protected from ambiguity in intent or action. Such laws are likely to spawn rules and regulations that will influence the metrics collected for evaluation of compliance.

5.1.3.4 Considerations for SOA Governance

The Reference Architecture definition of a loosely coupled system is one in which the constraints on the interactions between components is minimal: sufficient to permit interoperation without additional constraints that may be an artifact of implementation technology. While governance experience for standalone systems provides useful guides, we must be careful not to apply constraints that would preclude the flexibility, agility, and adaptability we expect to realize from a SOA ecosystem.

SOA governance must work effectively across ownership boundaries. Thus, there are likely to be multiple governance chains working in parallel. For example, a company making widgets likely has policies intended to ensure they make high quality widgets and make an impressive profit for their shareholders. On the other hand, Sarbanes-Oxley is a parallel governance chain in the United States that specifies how the management must handle its accounting and information that needs to be given to its shareholders. The parallel chains may just be additive or may be in conflict and require some harmonization.

One of the strengths of SOA is it can make effective use of diversity rather than requiring monolithic solutions. Heterogeneous organizations can interact without requiring each conforms to uniform tools, representation, and processes. However, with this diversity comes the need to adequately define those elements necessary for consistent interaction among systems and participants, such as which communication protocol, what level of security, which vocabulary for payload content of messages. The solution is not always to lock down these choices but to standardize alternatives and standardize the representations through which an unambiguous identification of the alternative chosen can be conveyed. For example, the URI standard specifies the URI string, including what protocol is being used, what is the target of the message, and how may parameters be attached. It does not limit the available protocols, the semantics of the target address, or the parameters that can be transferred. Thus, as with our definition of loose coupling, it provides absolute constraints but minimizes which constraints it imposes.

There is not a one-size-fits-all governance but a need to understand the types of things governance will be called on to do in the context of the goals of SOA. It is likely that some communities will initially desire and require very stringent governance policies and procedures while other will see need for very little. Over time, best practices will evolve, likely resulting in some consensus on a sensible minimum and, except in extreme cases where it is demonstrated to be necessary, a loosening of strict governance toward the best practice mean.

A question of how much governance may center on how much time governance activities require versus how quickly is the system being governed expected to respond to changing conditions. For large single systems that take years to develop, the governance process could move slowly without having a serious negative impact. For example, if something takes two years to develop and the steps involved in governance take two months to navigate, then the governance can go along in parallel and may not have a significant impact on system response to changes. Situations where it takes as long to navigate governance requirements as it does to develop a response are examples where governance may need to be reevaluated as to whether it facilitates or inhibits the desired results. Thus, the speed at which services are expected to appear and evolve needs to be considered when deciding the processes for control. The added weight of governance should be appropriate for overall goals of the application domain and the service environment.

Governance, as with other aspects of any SOA implementation, should start small and be conceptualized in a way that keeps it flexible, scalable, and realistic. A set of useful guidelines would include:

- Do not hardwire things that will inevitably change. For example, develop a system that uses the representation of policies rather and code the policies into the implementations.
- Avoid setting up processes that demo well for three services without considering how it will work for 300. Similarly, consider whether the display of status and activity for a small number of services will also be effective for an operator in a crisis situation looking at dozens of services, each with numerous, sometimes overlapping and sometimes differing activities.
- Maintain consistency and realism. A service solution responding to a natural disaster cannot be expected to complete a 6-week review cycle but be effective in a matter of hours.

5.1.4 Architectural Implications of SOA Governance

The description of SOA governance indicates numerous architectural requirements on the SOA ecosystem:

- Governance is expressed through policies and assumes multiple use of focused policy modules that can be employed across many common circumstances. This requires the existence of:
    - descriptions to enable the policy modules to be visible, where the description includes a unique identifier for the policy and a sufficient, and preferably a machine process-able, representation of the meaning of terms used to describe the policy, its functions, and its effects;
    - one or more discovery mechanisms that enable searching for policies that best meet the search criteria specified by the service participant; where the discovery mechanism will have access to the individual policy descriptions, possibly through some repository mechanism;
    - accessible storage of policies and policy descriptions, so service participants can access, examine, and use the policies as defined.
- Governance requires that the participants understand the intent of governance, the structures created to define and implement governance, and the processes to be followed to make governance operational. This requires the existence of:
    - an information collection site, such as a Web page or portal, where governance information is stored and from which the information is always available for access;
    - a mechanism to inform participants of significant governance events, such as changes in policies, rules, or regulations;
    - accessible storage of the specifics of Governance Processes;
    - SOA services to access automated implementations of the Governance Processes
- Governance policies are made operational through rules and regulations. This requires the existence of:
    - descriptions to enable the rules and regulations to be visible, where the description includes a unique identifier and a sufficient, and preferably a machine process-able, representation of the meaning of terms used to describe the rules and regulations;
    - one or more discovery mechanisms that enable searching for rules and regulations that may apply to situations corresponding to the search criteria specified by the service participant; where the discovery mechanism will have access to the individual descriptions of rules and regulations, possibly through some repository mechanism;
    - accessible storage of rules and regulations and their respective descriptions, so service participants can understand and prepare for compliance, as defined.
    - SOA services to access automated implementations of the Governance Processes.
- Governance implies management to define and enforce rules and regulations. Management is discussed more specifically in section 5.3, but in a parallel to governance, management requires the existence of:
    - an information collection site, such as a Web page or portal, where management information is stored and from which the information is always available for access;
    - a mechanism to inform participants of significant management events, such as changes in rules or regulations;
    - accessible storage of the specifics of processes followed by management.
- Governance relies on metrics to define and measure compliance. This requires the existence of:
    - the infrastructure monitoring and reporting information on SOA resources;
    - possible interface requirements to make accessible metrics information generated or most easily accessed by the service itself.

5.2 Security Model

Security is one aspect of confidence – the confidence in the integrity, reliability, and confidentiality of the system. In particular, security focuses on those aspects of assurance that involve the accidental or malign intent of other people to damage or compromise trust in the system and on the availability of SOA-based systems to perform desired capability.

Providing for security for Service Oriented Architecture is somewhat different than for other contexts; although many of the same principles apply equally to SOA and to other systems. The fact that SOA embraces crossing ownership boundaries makes the issues involved with moving data more visible.

Any comprehensive security solution must take into account the people that are using, maintaining and managing the SOA. Furthermore, the relationships between them must also be incorporated: any security assertions that may be associated with particular interactions originate in the people that are behind the interaction.

However, the fact that we aim to explicitly relate the IT architecture with the human architecture (see Business via Services) makes it possible to give a more complete accounting of security. In effect, an analysis of the social structures in place around a SOA-based system forms a backdrop and context for security.

Concepts such as constitutions, roles, and authority within social structures play an important part in the establishment of ownership and trust boundaries within and between social structures.

In addition, security often revolves around *resources*: the need to guard certain resources against inappropriate access – whether reading, writing or otherwise manipulating those resources. The basic resource model that informs our discussion is outlined in Section 3.2.

We analyze security in terms the social structures that define the legitimate permissions, obligations and roles of people in relation to the system, and mechanisms that must be put into place to realize a secure system. The former are typically captured in a series of security policy statements; the latter in terms of security *guards* that ensure that policies are enforced.

How and when to apply these derived security policy mechanisms is directly associated with the assessment of the *threat model* and a *security response model*. The threat model identifies the kinds of threats that directly impact the message and/or application of constraints, and the response model is the proposed mitigation to those threats. Properly implemented, the result can be an acceptable level of risk to the safety and integrity of the system.

5.2.1 Security Concepts

We can characterize security in terms of key security concepts **[ISO/IEC 27002]**: confidentiality, integrity, authentication, authorization, non-repudiation, and availability.

Confidentiality

> Confidentiality concerns the protection of privacy of participants in their interactions. Confidentiality refers to the assurance that unauthorized entities are not able to read messages or parts of messages that are transmitted.
>
> Note that confidentiality has degrees: in a completely confidential exchange, third parties would not even be aware that a confidential exchange has occurred. In a partially confidential exchange, the identities of the participants may be known but the content of the exchange obscured.

Integrity

> Integrity concerns the protection of information that is exchanged – either from unauthorized writing or inadvertent corruption. Integrity refers to the assurance that information that has been exchanged has not been altered.
>
> Integrity is different from confidentiality in that messages that are sent from one participant to another may be obscured to a third party, but the third party may still be able to introduce his own content into the exchange without the knowledge of the participants.

Availability

Availability concerns the ability of systems to use and offer the services for which they were designed. One of the threats against availability is the so-called denial of service attack in which attackers attempt to prevent legitimate access to the system.

We differentiate here between general availability – which includes aspects such as systems reliability – and availability as a security concept where we need to respond to active threats to the system.

Authentication

Authentication concerns the identity of the participants in an exchange. Authentication refers to the means by which one participant can be assured of the identity of other participants.

Authorization

Authorization concerns the legitimacy of the interaction. Authorization refers to the means by which an owner of a resource may be assured that the information and actions that are exchanged are either explicitly or implicitly approved.

Non-repudiation

Non-repudiation concerns the accountability of participants. To foster trust in the performance of a system used to conduct shared activities it is important that the participants are not able to later deny their actions: to repudiate them. Non-repudiation refers to the means by which a participant may not, at a later time, successfully deny having participated in the interaction or having performed the actions as reported by other participants.

Note that these security goals are never absolute: it is not possible to guarantee 100% confidentiality, non-repudiation, etc. However, a well designed and implemented security response model can ensure acceptable levels of security risk. For example, using a well-designed cipher to encrypt messages may make the cost of breaking communications so great and so lengthy that the information obtained is valueless.

While confidentiality and integrity can be viewed as primarily the concerns of the direct participants in an interaction; authentication, authorization, and non-repudiation imply the participants are acting within a broader social structure.

5.2.2 Where SOA Security is Different

The core security concepts are fundamental to all social interactions. The evolution of sharing information using a SOA requires the flexibility to dynamically secure computing interactions in a computing ecosystem where the owning social groups, roles, and authority are constantly changing as described in section 5.1.3.1.

SOA is primarily about action and events. This model focuses on the issues around these concepts more than simple data exchange.

SOA policy-based security can be more adaptive for a computing ecosystem than previous computing technologies allow for, and typically involves a greater degree of distributed mechanisms. Section 4.4.3.2 provides one example of distributed policy-based computing mechanisms that may be present as part of the realization of SOA security. Distributed security mechanisms allow for centralized identity and policy services as well as centralized or decentralized authentication and authorization services.

Standards for security, as is the case with all aspects of SOA, play a large role in flexible security on a global scale. SOA security may also involve greater auditing and reporting to adhere to regulatory compliance established by governance structures.

5.2.3 Trust Model

Trust is an assertion as to the behavior of participants in relation to each other. In terms of security assurance, trust often refers to the confidence that target systems may have as to the identity and validity of a participant as they interact with the system. However, in general, trust is a far larger topic.

Figure 54 models trust in terms of a participant, the participant's identity and credentials, and the participant's authorization to perform an action.

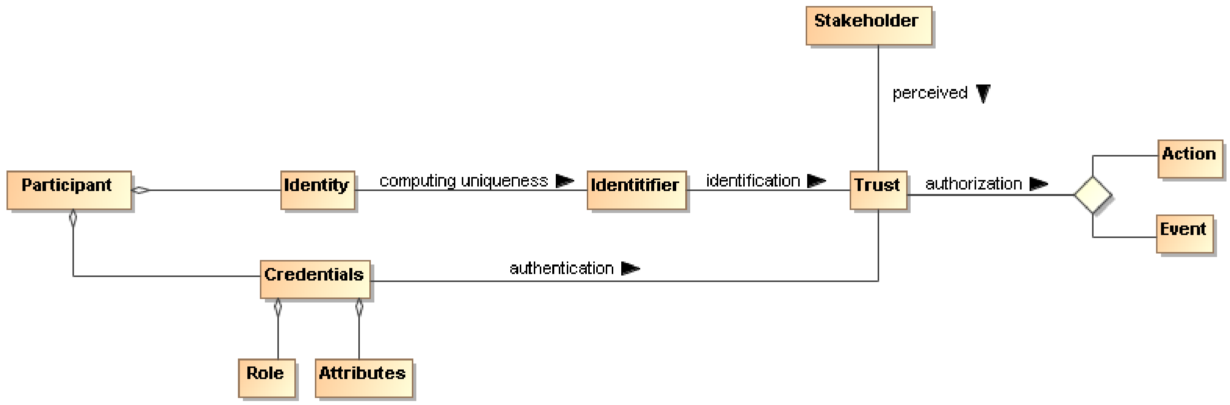


Figure 54 Trust Model

Trust

Trust is the relationship, as perceived by a stakeholder, between a participant and a set of actions and events, which concerns the legitimacy of the agent's actions and reported events.

Credentials

The role and/or set of attributes a stakeholder uses to determine authorization to actions.

Trust is not easily modeled as a single number or other scalar value. The motivation for this definition of trust is to allow us to distinguish the purpose of the trust as well as the degree of trust. For example, one may trust a stranger to hold a space in a queue for the Cinema, but one would typically not trust that same person to hold one's car keys for a fortnight's vacation.

5.2.3.1 Trust Domain

The Trust Domain in Figure 55 models abstract concepts behind the formation of policy-based trusted social groups.

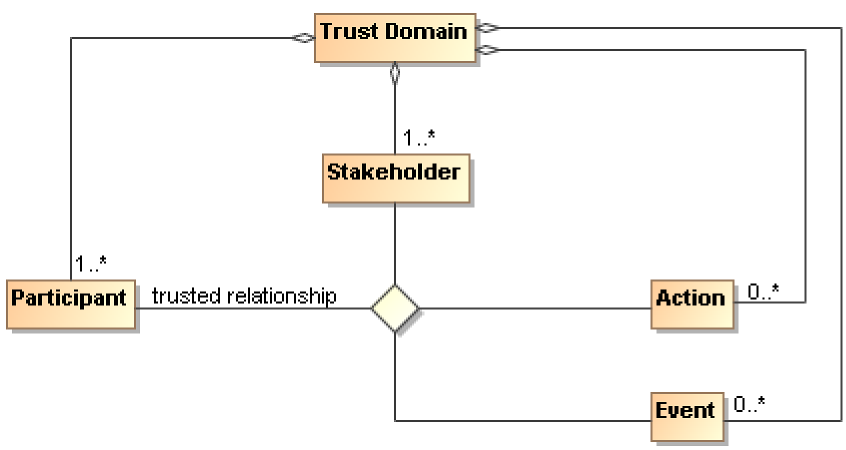


Figure 55 Trust Domain

Trust Domain

An abstract space of actions which all share a common trust requirement; i.e., all participants that perform any of the actions must be in the same trust relationship.

There are various kinds of trust domain: at the infrastructure level, a trust domain may refer to the networking equipment that is under the control of the owners of a SOA and is used to propagate communication. At an application level, a trust domain may refer to a social structure (see Section 3.4) within which members have previously established a certain degree of trust.

5.2.3.2 Centralized and Decentralized Trust Authority

Generally, there are special procedures necessary to communicate across trust domains: for example, participants may need to present credentials to participate in a trust domain. Once authenticated, credentials would typically not be needed to continue within that trust domain.

Trust domains will require a centralized and/or decentralized authentication and authorization authority to form trust relationships. An example of a centralized authority might be a governing body that requires regulatory compliance for all participants performing a specific action. A decentralized trust authority gives individual participant's more authority to authenticate and authorize actions and events.

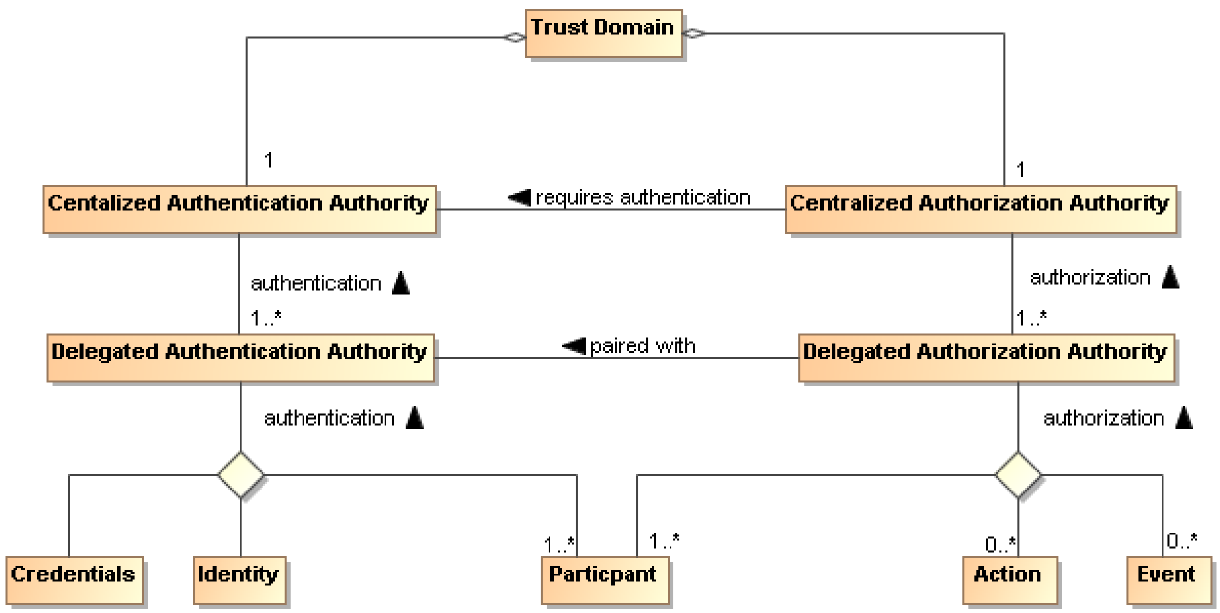


Figure 56 Centralized Trust Authority

Figure 56 depicts a hierarchical central trust authority. A participant's credentials and identity are authenticated by a centralized authentication authority. A web browser will often use a centralized authority in establishing secure communications with a service provider such as a bank. Actions and events also have centralized authorization authorities in this model. Centralized trust authorities tend to provide stronger regulatory control and more efficient revocation of participants.

In the context of a SOA that is used by many people, there may not be a single repository for information that can justify trust. Often different aspects of trust are managed by different entities. For example, a corporate directory might be used to verify the employment of an individual, whereas a bank would be used to verify their credit worthiness and a government agency used to verify their residency. Figure 57 depicts chains of trust between participants that are established by participants who introduce other participants into the chain of trust.

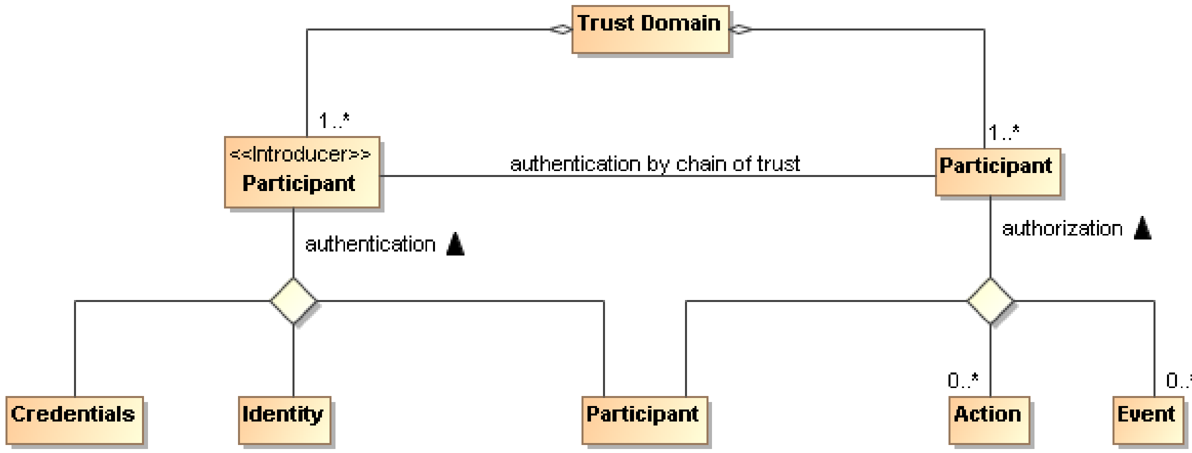


Figure 57 Decentralized Trust Authority

Together, the various entities that provide corroboration of an individual's authenticity and trustworthiness to perform actions and raise events form a chain of trust. Chain's of trust need not be functionally organized: third parties who are known to both may also be used to facilitate trust. A long chain of trust is likely to be more fragile and less trustworthy than a simple one.

Complex trust domains are likely to be composed of a combination of centralized and decentralized trust authorities. For SOA, the level of complexity of a trust domain can achieve is dependent on the policy language's and IT mechanism's ability to express trust relationships.

5.2.3.3 Policy Mechanisms for Security

When a participant wishes to perform an action that requires access to a trust domain, depending on the policies that are in place, he/she must provide suitable identification and/or credentials before continuing the interaction.

Security policies are not equivalent to security. However, they are very important as the expression of choices that can be used by security mechanisms to enforce security.

The role of a machine readable security policy is to permit stakeholders to express their choices; and, on the other hand, to act as instructions for security enforcement mechanisms.

Figure 58 depicts security interactions based on Section 4.4.3. In the context of security, the diagram has been modified with recognized policy, identity, and attribute authorities in the SOA ecosystem. Additional auditing has also been depicted.

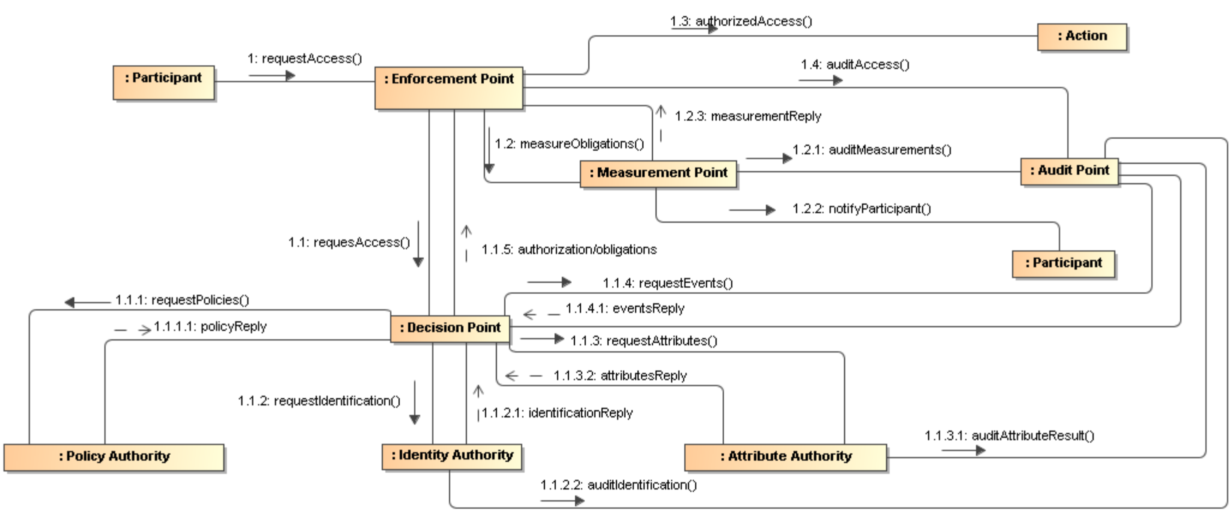


Figure 58 Policy Based Security

Mechanisms are not the same as solutions; a combination of security mechanisms and their control via explicit policies can form the basis of a solution. Elsewhere in the architecture policies are used to express routing constraints, business constraints and information processing constraints. Security policies are used to marry stakeholders’ choices with mechanisms to enforce security.

5.2.4 Security Layers

Security concepts can be described in terms of three primary layers when discussing the deployment of SOA-based systems. The commonly known OSI seven-layer model provides an expanded view of these three primary layers, each one of the OSI seven layers requires specific application of security. However, discussing the seven layers of the OSI seven-layer model is beyond the scope of this reference architecture.

Figure 59 depicts three generalized layers of security to consider and their relationship to ownership domains when deploying SOA-based systems. The lowest level of abstraction is the network layer, the next level of abstraction is the transport layer, and the third level of abstraction is the application layer.

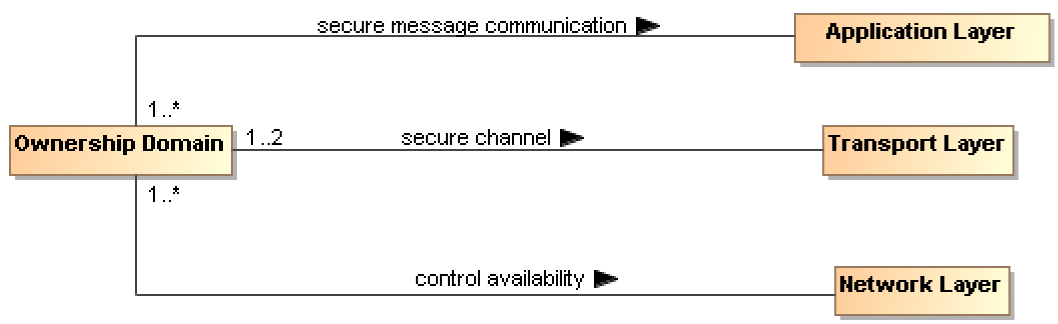


Figure 59 Security Layers

5.2.4.1 Network Layer

At the lowest level of abstraction in the security model are the network devices and the hardware that links the network devices, referred to as the network layer. The network layer includes devices like routers and firewall appliances and it also includes protocols such as the Internet Protocol (IP), Border Gateway Protocol (BGP), Open Shortest Path First (OSPF) protocol, etc. Network devices, however, can have policy-based SOA security mechanisms built in so there is not always a clear distinction between network device and network layer.

In order for a SOA-based system to operate, the network must be available to provide network services. Control of the network layer is required in order to address the security concept of availability such as protection from Denial of Service (DoS) attacks.

The network layer may also address general availability by defining policies or service level agreements (SLAs) about the quality of service of the network layer operation and then translating hose commitments into measurable constraints carried out by the network devices for such things as guaranteed service delivery or specific bandwidth allocations.

5.2.4.2 Transport Layer

The transport layer may pass through network layers belonging to many ownership domains. The transport layer is primarily concerned with establishing a secure communications channel between sender and receiver, a good example being the interaction with a bank through a web browser. The transport layer may include protocols like HTTP over Transport Layer Security (TLS) as well as HTTP over Secure Sockets Layer (SSL).

Given the nature of SOA-based communications across multiple ownership boundaries, security provided at the transport layer cannot be relied upon for protection of message confidentiality.

5.2.4.3 Application Layer

The application layer accounts for the security of messaging between participants within a SOA ecosystem, where participants may have policy based roles and authority to act within and across ownership domains. Web service standards like WS-Security, XML Digital Signature, XML Encryption, and SAML are all examples of standards addressing the security concepts at the application layer.

Application layer security for SOAs may be built into network devices so network devices may have network layer and application layer security built in.

In a SOA ecosystem where participants interact through many ownership domains and any number of unknown network domains, the application layer may be the only layer the basic security principles of confidentiality, integrity, authentication, authorization, and non-repudiation are assured. Assurance of availability is addressed at the network layer but may be controlled by the application layer and/or transport layer.

5.2.5 Threat Model

There are a number of ways in which an attacker may attempt to compromise the security of a system. The two primary sources of attack are third parties attempting to subvert interactions between legitimate participants and an entity that is participating but attempting to subvert its partner(s). The latter is particularly important in a SOA where there may be multiple ownership boundaries and trust boundaries.

The threat model lists some common threats that relate to the core security concepts listed in Section 5.2.1. Each technology choice in the realization of a SOA can potentially have many threats to consider.

Message alteration

> If an attacker is able to modify the content (or even the order) of messages that are exchanged without the legitimate participants being aware of it then the attacker has successfully compromised the security of the system. In effect, the participants may unwittingly serve the needs of the attacker rather than their own.
>
> An attacker may not need to completely replace a message with his own to achieve his objective: replacing the identity of the beneficiary of a transaction may be enough.

Message interception

> If an attacker is able to intercept and understand messages exchanged between participants, then the attacker may be able to gain advantage. This is probably the most commonly understood security threat.

Man in the middle

> In a man in the middle attack, the legitimate participants believe that they are interacting with each other; but are in fact interacting with the attacker. The attacker attempts to convince each participant that he is their correspondent; whereas in fact he is not.
>
> In a successful man-in-the-middle attack, legitimate participants will often not have a true understanding of the state of the other participants. The attacker can use this to subvert the intentions of the participants.

Spoofing

> In a spoofing attack, the attacker convinces a participant that he is really someone else – someone that the participant would normally trust.

Denial of service attack

> In a denial of service attack, the attacker attempts to prevent legitimate users from making use of the service. A DoS attack is easy to mount and can cause considerable harm: by preventing legitimate interactions, or by slowing them down enough, the attacker may be able to simultaneously prevent legitimate access to a service and to attack the service by another means.

A variation of the DoS attack is the **Distributed Denial of Service** attack. In a DDoS attack the attacker uses multiple agents to the attack the target. In some circumstances this can be extremely difficult to counteract effectively.

One of the features of a DoS attack is that it does not require valid interactions to be effective: responding to invalid messages also takes resources and that may be sufficient to cripple the target.

Replay attack

In a replay attack, the attacker captures the message traffic during a legitimate interaction and then replays part of it the target. The target is persuaded that a similar transaction to the previous one is being repeated and it will respond as though it were a legitimate interaction.

A replay attack may not require that the attacker understand any of the individual communications; the attacker may have different objectives (for example attempting to predict how the target would react to a particular request).

False Repudiation

In false repudiation, a malicious user completes a normal transaction and then later attempts to deny that the transaction occurred. For example, a customer may use a service to buy a book using a credit card; then, when the book is delivered, refuse to pay the credit card bill claiming that *someone else* must have ordered the book.

5.2.6 Security Response Model

Performing threat assessments, devising mitigation strategies, and determining acceptable levels of risk are the foundation for an effective process to mitigating threats in a cost-effective way.[17] The choice in hardware and software to realize a SOA will be the basis for threat assessments and mitigation strategies. The stakeholders of a specific SOA implementation should determine acceptable levels of risk based on threat assessments and the cost of mitigating those threats. Example mitigation strategies are provided for threats listed in Section 5.2.5.

5.2.6.1 Privacy Enforcement

The most efficient mechanism to assure confidentiality is the encryption of information. Encryption is particularly important when messages must cross trust boundaries; especially over the Internet. Note that encryption need not be limited to the content of messages: it is possible to obscure even the existence of messages themselves through encryption and ‘white noise’ generation in the communications channel.

The specifics of encryption are beyond the scope of this architecture. However, we are concerned about how the connection between privacy-related policies and their enforcement is made. In Section 4.4.3, we show how policies in general are enforced using a combination of Policy Decision Points (PDP) and Policy Enforcement Points (PEP).

A PEP for enforcing privacy may take the form of an automatic function to encrypt messages as they leave a trust boundary; or perhaps simply ensuring that such messages are suitably encrypted.

Any policies relating to the level of encryption being used would then apply to these centralized messaging functions.

5.2.6.2 Integrity Protection

To protect against message tampering or inadvertent message alteration, and to allow the receiver of a message to authenticate the sender, messages may be accompanied by a digital signature. Digital

[17] In practice, there are perceptions of security from all participants regardless of ownership boundaries. Satisfying security policy often requires asserting sensitive information about the message initiator. The perceptions of this participant about information privacy may be more important than actual security enforcement within the SOA for this stakeholder.

signatures provide a means to detect if signed data has been altered. This protection can also extend to authentication and non-repudiation of a sender.

A common way a digital signature is generated is with the use of a private key that is associated with a public key and a digital certificate. The private key of some entity in the system is used to create a digital signature for some set of data. Other entities in the system can check the integrity of the signed data set via signature verification algorithms. Any changes to the data that was signed will cause signature verification to fail, which indicates that integrity of the data set has been compromised.

A party verifying a digital signature must have access to the public key that corresponds to the private key used to generate the signature. A digital certificate contains the public key of the owner, and is itself protected by a digital signature created using the private key of the issuing Certificate Authority (CA).

5.2.6.3 Message Replay Protection

To protect against replay attacks, messages may contain information that can be used to detect replayed messages. The simplest requirement to prevent replay attacks is that each message that is ever sent is unique. For example, a message may contain a message ID, a timestamp, the intended destination.

By caching message IDs, and comparing each new message with the cache, it becomes possible to verify whether a given message has been received before (and therefore should be discarded).

The timestamp may be included in the message to help check for message freshness. Messages that arrive after their message ID could have been cleared (after receiving the same message some time previously) may also have been replayed. A common means for representing timestamps is a useful part of an interoperable replay detection mechanism.

The destination information is used to determine if the message was misdirected or replayed. If the replayed message is sent to a different endpoint than the destination of the original message, the replay could go undetected if the message does not contain information about the intended destination.

In the case of messages that are replies to prior messages, it is also possible to include seed information in the prior messages that is randomly and uniquely generated for each message that is sent out. A replay attack can then be detected if the reply does not embed the random number that corresponds to the original message.

5.2.6.4 Auditing and Logging

False repudiation involves a participant denying that it authorized a previous interaction. An effective strategy for responding to such a denial is to maintain careful and complete logs of interactions which can be used for auditing purposes. The more detailed and comprehensive an audit trail is, the less likely it is that a false repudiation would be successful.

The countermeasures assume that the non-repudiation tactic (e.g. digital signatures) is not undermined itself. For example, if private key is stolen and used by an adversary, even extensive logging cannot assist in rejecting a false repudiation.

Unlike many of the security responses discussed here, it is likely that the scope for automation in rejecting a repudiation attempt is limited to careful logging.

5.2.6.5 Graduated engagement

The key to managing and responding to DoS attacks is to be careful in the use of resources when responding to interaction. Put simply, a system has a choice to respond to a communication or to ignore it. In order to avoid vulnerability to DoS attacks a service provider should be careful not to commit resources beyond those implied by the current state of interactions; this permits a graduation in commitment by the service provider that mirrors any commitment on the part of service consumers and attackers alike.

5.2.7 Architectural Implications of SOA Security

Providing SOA security in an ecosystem of governed services has the following implications on the policy support and the distributed nature of mechanisms used to assure SOA security:

- Security expressed through policies have the same architectural implications as described in Section 4.4.5 for policies and contracts architectural implications.
- Security policies require mechanisms to support security description administration, storage, and distribution.
- Security policies should:
    - be able to express trust relationships and trust domains;
    - provide the ability to update policy trust relationships and trust domains in a way that does not require upgrades to software and hardware;
    - be able to express standard protocols used to provide confidentiality, integrity, authentication, authorization, non-repudiation, and availability.
- Service descriptions supporting security policies should:
    - have a meta-structure sufficiently rich to support security policies;
    - be able to reference one or more security policy artifacts;
    - have a framework for resolving conflicts between security policies.
- The mechanisms that make-up the execution context in secure SOA-based message exchanges should:
    - provide protection of the confidentiality and integrity of message exchanges;
    - be distributed so as to provide centralized or decentralized policy-based identification, authentication, and authorization;
    - ensure service availability to consumers;
    - be able to scale to support security for a growing ecosystem of services;
    - be able to support security between different communication technologies;
- Common security services include:
    - services that abstract encryption techniques;
    - services for auditing and logging interactions and security violations;
    - services for identification;
    - services for authentication;
    - services for authorization;
    - services for intrusion detection and prevention;
    - services for availability including support for quality of service specifications and metrics.

5.3 Services as Managed Entities Model

Management

> Management is the control of the use, configuration, and availability of resources in accordance with the policies of the stakeholders involved.

There are three separate but linked domains of interest within the management of SOA-based systems. The first and most obvious is the management and support of the resources that are involved in any complex system – of which SOA-based systems are excellent examples. The second is the promulgation and enforcement of the policies and contracts agreed to by the stakeholders in SOA-based systems. The third domain is the management of the relationships of the participants in SOA-based systems – both to each other and to the services that they use and offer.

There are many artifacts in a large system that may need management. As soon as there is the possibility of more than one instance of a thing, the issue of managing those things becomes relevant. Historically, systems management capabilities have been organized by the following functional groups known as “FCAPS” functions (based on ITU-T Rec. M.3400 (02/2000), "TMN Management Functions"): Fault management, configuration management, account management, performance and security management.

In the context of SOA we see many possible resources that may require management: services, service descriptions, service capabilities, policies, contracts, roles, relationships, security, and infrastructure elements. In addition, given the ecosystem nature of SOA, it is also potentially necessary to manage the business relationships between participants in the SOA.

Managing systems that may be used across ownership boundaries raises issues that are not normally present when managing a system within a single ownership domain. For example, care is required managing a service when the owner of the service, the provider of the service, the host of the service and access mediators to the service may all belong to different stakeholders. In addition, it may be important to allow service consumers to communicate their requirements to the service provider so that they are satisfied in a timely manner.

A given service may be provided and consumed in more than one version. Version control of services is important both for service providers and service consumers (who may need to ensure certainty in the version of the service they are interacting with).

In fact, managing a service has quite a few similarities to using a service: suggesting that we can use the service oriented model to manage SOA-based systems as well as provide them. A management service would be distinguished from a non-management service more by the nature of the capabilities involved (i.e., capabilities that relate to managing services) than by any intrinsic difference.

In this model, we show how the SOA framework may apply to managing services as well as using and offering them. There are, of course, some special considerations that apply to service management which we bring out: namely that we will be managing the life-cycle of services, managing any service level attributes, managing dependencies between services and so on.

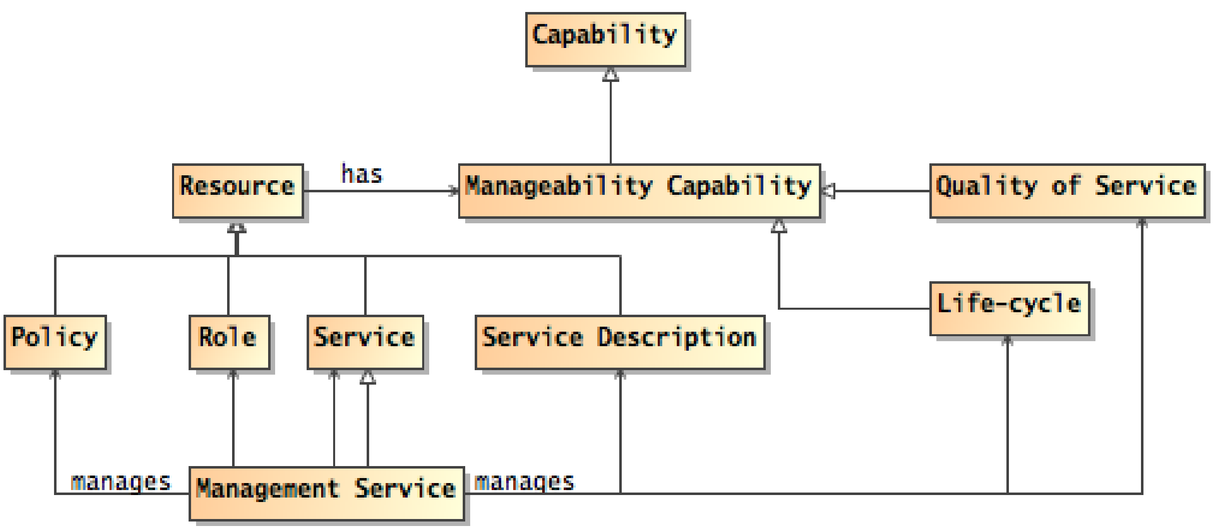


Figure 60 Managing resources in a SOA

The core concept in management is that of a manageability capability:

Manageability Capability

> The manageability capability of a resource is the capability that allows it to be managed with respect to some property. Note that manageability capabilities are not necessarily part of the managed entities themselves.
>
> Manageability capabilities are the core resources that management systems use to manage: each resource that may be managed in some way has a number of aspects that may be managed. For example, a service's life-cycle may be manageable, as may its Quality of Service parameter; a policy may also be managed for life-cycle but Quality of Service would not normally apply.

Life-cycle manageability

A manageability capability associated with a resource that permits the life cycle of the resource to be managed. As noted above, the life-cycle manageability capability of a resource is unlikely to reside within the resource itself (you cannot tell a system that is not running to start itself).

The life-cycle management of a resource typically refers to how the resource is created, how it is destroyed and what dependencies there might exist that must be simultaneously managed.

Configuration manageability

A capability that permits the configuration of resources to be managed. Service configuration, in particular, may be complex in cases where there are dependencies between services and other resources.

Event monitoring manageability

Managing the reporting of events and faults is one of the key lower-level manageability capabilities.

Accounting manageability

A capability associated with resources that allows for the use of those resources to be measured and accounted for. This implies that not only can the *use* of resources be properly measured, but also that those *using* those resources also be properly identified.

Accounting for the use of resources by participants in the SOA supports the proper budgeting and allocation of funding by participants.

Quality of service manageability

A manageability capability associated with a resource that permits any quality of service associated with the resource to be managed. Classic examples of this include bandwidth requirements and offerings associated with a service.

Business performance manageability

A manageability capability that is associated with services that permits the service's business performance to be monitored and managed. In particular, if there are business-level service level agreements that apply to a service, being able to monitor and manage those SLAs is an important role for management systems.

Building support for arbitrary business monitoring is likely to be challenging. However, given a *measure* for determining a service's compliance to business service level agreements, management systems can monitor that performance in a way that is entirely similar to other management tasks.

Policy manageability

Where the policies associated with a resource may be complex and dynamic, so those policies themselves may require management. The ability to manage those policies (such as promulgating policies, retiring policies and ensuring that policy decision points and enforcement points are current) is a management function.

In the particular case of policies, there is a special relationship between management and policies. Just like other artifacts, policies require management in a SOA. However, much of management is about *applying* policies also: where governance is often about what the policies regarding artifacts and services should be, a key management role is to ensure that those policies are consistently applied.

Management service

A management service is a service that manages other services and resources.

Management Policy

A management policy is a policy whose topic is a management topic. Just as with other aspects of a SOA, the management of resources within the SOA may be governed by management policies, contracts (such as SLAs).

In a deployed system, it may well be that different aspects of the management of a given service are managed by different management services. For example, the life-cycle management of services often involves managing dependencies between services and resource requirements. Managing quality of service is often very specific to the service itself; for example, quality of service attributes for a video streaming service are quite different to those for a banking system.

There are additional concepts of management that often also apply to IT management:

Systems management

Systems management refers to enterprise-wide maintenance and administration of distributed computer systems.

Network management

Network management refers to the maintenance and administration of large-scale networks such as computer networks and telecommunication networks. Systems and network management execute a set of functions required for controlling, planning, deploying, coordinating, and monitoring the distributed computer systems and the resources of a network.

However, for the purposes of this Reference Architecture, while recognizing their importance, we do not focus on systems management or network management.

- the specific identifier is not prescribed by this Reference Architecture but the structure and semantics of the identifier must be indicated for the identifier value to be properly used. For example, part of identity may include version identification.

For this, the configuration management plan or similar document from which the version number is derived must be identified.

5.3.1 Management and Governance

The primary role of governance in the context of SOA is to allow the stakeholders in the SOA to be able to negotiate and set the key policies that govern the running of the system. Recall that in an ecosystems perspective, the goal is less to have complete fine-grained control but more to enable the individual participants to work together. Policies that are set at the governance of a SOA will tend to focus on the rules of engagement between participants – what kind of interacts are permissible, how to resolve disputes, and so on.

While governance may be primarily focused on setting policies, management is more focused on realization and enforcement of policies.

5.3.2 Management Contracts and Policies

As we noted above, management can often be viewed as the application of contracts and policies to ensure the smooth running of the SOA. Policies play an important part in managing systems both as artifacts that need to be managed and as the guiding constraints to determine how the SOA should be managed.

5.3.2.1 Policies

"Although provision of management capabilities enables a service to become manageable, the extent and degree of permissible management are defined in management policies that are associated with the services. Management policies are used to define the obligations for, and permissions to, managing the service." **[WSA]**

On the other hand, a policy without any means of enforcing it is vacuous. In the case of management policy, we rely on a management infrastructure to realize and enforce management policy.

5.3.3 Management Infrastructure

In order for a service or other resource to be manageable there must be a corresponding manageability capability that can effect that management. The particulars of this capability will vary somewhat depending on the nature of the capability. For example, a service life-cycle manageability capability requires the ability to start a service, to stop the service, and potentially to pause the service. Conversely, in order to manage document-like artifacts, such as service descriptions, the capability of storing the artifacts, controlling access to those artifacts, allowing updates of the artifacts to be deployed are all important capabilities for managing them.

Elements of a basic service management infrastructure should include the following characteristics:

- Integrate with existing security services
- Monitoring
- Heartbeat and Ping
- Alerting
- Pause/Restore/Restart Service Access
- Logging, Auditing, Non-Repudiation
- Runtime Version Management
- Complement other infrastructure services (discovery, messaging, mediation)

 * Message Routing and Redirection
   * Failover
   * Load-balancing

 * QoS, Management of Service Level Objects and Agreements
   * Availability
   * Response Time
   * Throughput

- Fault and Exception Management

5.3.4 Service Life-cycle

Managing a service's life cycle involves managing the establishment of the service, managing its steady-state performance, and managing its termination. The most obvious feature of this is that a service cannot manage its own life cycle (imagine asking a non-functioning service to start). Another important consideration is that services may have resource requirements that must be established at various points in the services' life cycles. These dependencies may take the form of other services being established; possibly even services that are not exposed by the service's own interface.

6 Conformance

This Reference Architecture is an abstract architecture, which means that it is especially difficult to construct automated tests for conformance to the architecture. However, in order to be conformant to this architecture, it should be possible to identify in a concrete implementation the key concepts and components of this architecture, albeit in abstracted form.

A. Acknowledgements

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

Participants:

Rex Brooks, individual member
Peter Brown, Pensive.eu
Scott Came, Search Group Inc.
Joseph Chiusano, Booz Allen Hamilton
Robert Ellinger, Northrup Grumman
David Ellis, Sandia National Laboratories
Jeff A. Estefan, Jet Propulsion Laboratory
Don Flinn, Individual Member
Anil John, Johns Hopkins University
Ken Laskey, MITRE Corporation
Francis G. McCabe, Individual
Christopher McDaniels, USSTRATCOM
Tom Merkle, Lockheed martin
Jyoti Namjoshi, Patni Computer Systems Ltd.
Duane Nickull, Adobe
Michael Poulin, Fidelity Investments
Michael Stiefel, Reliable Software
Danny Thornton, Individual
Timothy Vibbert, Lockheed Martin
Robert Vitello, New York Dept. of Labor

B. Critical Factors Analysis

A critical factors analysis (CFA) is an analysis of the key properties of a project. A CFA is analyzed in terms of the goals of the project, the critical factors that will lead to its success and the measurable requirements of the project implementation that support the goals of the project. CFA is particularly suitable for capturing non-functional requirements of a project: for example, security, scalability, wide-spread adoption, and so on. As such, CFA complements rather than attempts to replace other requirements capture techniques.

B.1 Goals

A goal is an overall target that you are trying to reach with the project. Typically, goals are hard to measure by themselves. Goals are often directed at the potential consumer of the product rather than the technology developer.

B.1.1 Critical Success Factors

A critical success factor (CSF) is a property, sub-goal that directly supports a goal and there is strong belief that without it the goal is unattainable. CSFs themselves are not necessarily measurable in themselves.

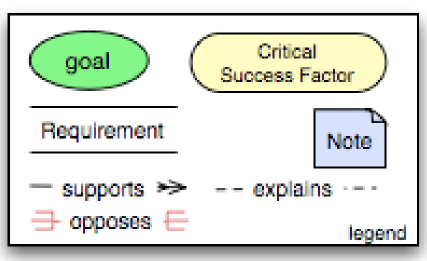
B.1.2 Requirements

A requirement is a specific measurable property that directly supports a CSF. The key here is measurability: it should be possible to unambiguously determine if a requirement has been met. While goals are typically directed at consumers of the specification, requirements are focused on technical aspects of the specification.

B.1.3 CFA Diagrams

It can often be helpful to illustrate graphically the key concepts and relationships between them. Such diagrams can act as effective indices into the written descriptions of goals etc., but is not intended to replace the text.

The legend:



illustrates the key elements of the graphical notation. Goals are written in round ovals, critical success factors are written in round-ended rectangles and requirements are written using open-ended rectangles. The arrows show whether a CSF/goal/requirement is supported by another element or opposed by it. This highlights the potential for conflict in requirements.