An Overview of Modeling Concepts for Service-Based Software Architectures

Florian Rademacher[1,2]
[1]IDiAL Institute
University of Applied Sciences and Arts Dortmund
[2]Department of Computer Science and Electrical Engineering
University of Kassel
florian.rademacher@fh-dortmund.de

Introduction

This document provides an overview of modeling concepts for service-based software architectures (SBSAs). The concepts were extracted during a survey of conceptual frameworks [1–6] for architecture modeling of SBSAs.

Each of the following sections FW.1 to FW.6 lists modeling concepts, which were extracted from a certain conceptual framework for SBSA modeling. Moreover, each modeling concept is characterized by the following extracted information:

- *Relationships:* Extracted relationships of the respective concept to other modeling concepts within the same conceptual framework for SBSA modeling. When applicable, relationships are expressed in a UML-like textual syntax. For example, the relationship specification "1..*" represents a one-to-many relationship from the regarded concept to other concepts of the same conceptual framework.
- *Structure:* Extracted properties of a modeling concept, which have a primitive type in the sense of the respective conceptual framework for SBSA modeling. That is, the structure does not repeat properties of the modeling concept that represent relationships to other modeling concepts of the same conceptual framework.
- *Constraints:* Extracted formal or informal constraints, which apply to a concept's relationships or structure.
- *Notes:* Textual notes, which further detail the definition of a modeling concept's semantics. These notes correspond to text fragments that were extracted from conceptual frameworks' publications [1–6].

FW.1 A platform independent model for service oriented architectures [1]

1.1 Information Aspect

- Relationships
  - Item **is** NamedElement
  - Item **has** 0..1 ItemType
  - ItemType **is** PackageableElement
  - PackageableElement **is** NamedElement
  - Package **is** NamedElement
  - Package **composes** * PackageableElements
  - Package **composes** * Packages
  - TypeLibrary **is** Package
  - TypeLibrary **has** * ItemTypes
  - Document **is** Package
  - Document **aggregates** * Entities
  - Entity **is** ItemType
  - Entity **composes** * Attributes
  - Entity **has** * Entities
  - Association **is** PackageableElement
- Structure
  - Attribute: (i) visibility : String; (ii) defaultValue : String
- Notes
  - In the context of virtual enterprises information represents one of the most important elements that need to be described. In fact the other aspects manage or are based on information elements.
  - The metamodel for the information aspect is inspired by the UML for EDOC standard [8] and the Class related parts of the UML metamodel.
  - Complex types, similar to UML classes without operations, are represented by the Entity element. Entities can contain a set of attributes and can have associations between them. Attributes are typed either by an ItemType or by an Entity.

1.2 Process Aspect

- Relationships
  - Scope **is** Behavior
  - Scope **composes** * Steps
  - Scope **composes** * Flows
  - Flow **has** 0..1 GuardSpecifications
  - Flow **is associated with** Interaction (association exists twice in metamodel)
  - Step **composes** * Interaction (association exists twice in metamodel)
  - Interaction **has** 0..1 JoinSpecification
  - Interaction **has** 0..1 Message

    - Message **composes** * Items
    - Interaction **composes** * Pins
    - ItemFlow **has** Pin
    - ItemFlow **is** Flow
    - Pin **has** 0..1 Item
- Notes
    - The process aspect is closely linked to the Service aspect, the primary link being an abstract class Scope that is a kind of Behaviour .A Scope can be instantiated as a Process belonging to a ServiceProvider from that aspect. Both abstract and executable Processes may be modeled, according to this property of the ServiceProvider.
    - The process contains a set of Steps (generally Tasks), representing actions carried out by the Process. These essentially fall into two categories, interactions with other service providers, or specialised actions requiring implementation beyond the scope of this model. The Process also contains a set of Flows between these actions, which may be specialised (ItemFlow) to indicate the transfer of specific data.

1.3 Service Aspect

- Relationships
    - ServiceProvider **composes** * Behaviours
    - ServiceProvider **composes** * CollaborationUses
    - Collaboration **composes** * Behaviours
    - Collaboration **composes** * CollaborationUses
    - Collaboration **has** * Roles
    - CollaborationUse **has** Collaboration
    - CollaborationUse **composes** * RoleBindings
    - RoleBinding **has** Role
    - RoleBindings **binds** Role
    - Role **composes** * Messages
    - Role **composes** * Items
    - ServiceProvider **composes** * Roles
    - Collaboration **composes** * EndPoints
- Structure
    - ServiceProvider: Type : ProviderType
    - EndPoint: address : String
    - Message: mode : MessageMode
- Notes
    - architectures are composed of functions provided by a system or a set of systems to achieve a shared goal. Its scope ranges from the description of a simple service to the description of a set of services provided by different systems to achieve a given objective. The Collaborations approach used addresses both of these levels of service description rather than tying to a specific viewpoint.
    - Collaboration: a service is viewed as a collaboration of a set of roles

    - CollaborationUse: represents the usage of collaboration
    - Role: A Role represents a part involved in a service.
    - Messages: represents the information exchanged by the different roles
    - ServiceProvider: can play roles in a collaborationUse
    - Behaviour: is an abstract class for the specification of the Process.

1.4 The QoS aspect

- Relationships
    - CollaborationUse **is associated with** NFA
    - NFA **composes** QoSCategory
    - QoSCategory **composes** * QoSCategories
    - QoSCategory **aggregates** * QoSCharacteristics
    - QoSCharacteristic **is** QoSContext
    - QoSCharacteristic **is associated with** * QoSContexts
    - QoSConstraint **has** 1..* QoSContexts
    - QoSDimension **is associated with** * QoSDimensionSlots
    - QoSDimensionSlot **has** 0..1 QoSValue
    - QoSValue **composes** * QoSDimensionSlots
    - QoSContext **aggregates** * QoSValues
    - Collaboration **composes** NFA
- Notes
    - As starting point this part of the metamodel is based on the quality of service OMG standard called UML Profile for Modelling Quality of Service and Fault Tolerance Characteristics and Mechanisms [12].

FW.2 Service-oriented Modeling Framework (SOMF) [2]

2.1 Aggregated Structural Modeling Connector

- Notes
  - Inserts a fine-grained analysis asset into a coarse-grained analysis asset

2.2 Analysis Atomic Service

- Notes
  - A fine-grained service that is impractical to decompose because of its suggested limited capabilities or processes

2.3 Analysis Cloud

- Notes
  - Represents a collection of analysis services in three different categories: Software as Service (SaaS), Platform as Service (PaaS), and Infrastructure as Service (IaaS).
  - Additional types can be added on demand

2.4 Analysis Composite Service

- Notes
  - A coarse-grained service comprised of internal fine-grained atomic or composite services, forming hierarchical parent-child associations

2.5 Analysis Service Cluster

- Notes
  - An association of services grouped by related business or technical processes that collaborate to offer solutions

2.6 Blank Cloud Typing Tag

- Notes
  - Enables other cloud definitions that are not part of this list

2.7 Bound Structural Modeling Connector

- Notes
  - Establishes a formal contract between two analysis assets?

2.8 Clipped Structural Modeling Connector

- Notes
  - Extracts analysis assets that are aggregated in composite, cluster, cloud, or compounded service formations

2.9 Cloned Structural Modeling Connector

- Notes
  - Duplicates an analysis asset

2.10 Community Cloud Typing Tag

- Notes
  - Identifies a cloud whose services are consumed by two or more organizations that share similar business or technical requirements

2.11 Compounded Structural Modeling Connector

- Notes
  - Groups two or more analysis assets to collaborate on providing a solution

2.12 Contracted Contextual Modeling Connector

- Notes
  - A modeling activity that is performed on a service’s environment to reduce the architecture and technology capabilities

2.13 Consumer

- Notes
  - Any entity that is identified with service consumption activities. This definition may include consuming applications or services

2.14 Coupled Structural Modeling Connector

- Notes
  - Links two analysis assets

2.15 De-Cloned Structural Modeling Connector

- Notes
  - Unties cloned relationship between two analysis assets

2.16 De-Coupled Structural Modeling Connector

- Notes
  - Unlinks two analysis assets

2.17 Decomposed Structural Modeling Connector

- Notes
  - Separates an analysis asset from another analysis asset

2.18 Excluded Structural Modeling Connector

- Notes
  - Identifies uncommon analysis assets that reside in an overlapped region of intersected analysis assets

2.19 Expanded Contextual Modeling Connector

- Notes
  - A modeling activity that is performed on a service’s environment to expand the architecture and technology capabilities

2.20 ExtraCloud Space

- Notes
  - A modeling area that depicts services that operate outside of a cloud

2.21 Fragmented Service Space

- Notes
  - A modeling space that is dedicated to a service that has been decomposed into smaller services and then retired

2.22 Fragmented Structural Modeling Connector

- Notes
  - Breaks down an analysis asset into smaller finer-grained analysis assets. The source entity then retires

2.23 Generalized Contextual Modeling Connector

- Notes
  - A modeling activity that is performed on a service to raise its abstraction level and increase its scope of operations and capabilities

2.24 Hybrid Cloud Typing Tag

- Notes
  - Depicts a cloud that combines the properties of two or more cloud types described on this list

2.25 InterCloud

- Notes
  - Represents the term “cloud-of- clouds.” A superior cloud that identifies a group of related clouds, working together to offer collaborative solutions

2.26 Intersected Region

- Notes
  - A common space to two or more intersecting composite, compounded, cluster, or cloud entities

2.27 Intersected Structural Modeling Connector

- Notes
  - Intersects two analysis assets

2.28 IntraCloud Space

- Notes
  - A modeling area that depicts services that operate in a cloud

2.29 Joined Structural Modeling Connector

- Notes

    - Joins two analysis assets to provide permanent or temporary solutions

2.30 Offset Structural Modeling Connector

- Notes
    - Increases the structural scale of an analysis asset

2.31 Operation Numbering Tag Structural Modeling Connector

- Notes
    - Identifies the modeling operation sequence

2.32 Organizational Boundary

- Notes
    - A computing area of an organization, such as a division, department, company, partner company, consumer, or community

2.33 Overlapped Structural Modeling Connector

- Notes
    - Identifies common analysis assets that reside in an overlapped region of intersected analysis assets

2.34 Private Cloud Typing Tag

- Notes
    - Indicates a cloud of services that is sponsored, maintained, and operated by an organization, available only on private networks, and is utilized exclusively by internal consumers

2.35 Public Cloud Typing Tag

- Notes
    - Identifies a cloud that is maintained by an off-site party service provider, which offers configurable features and deployments charged to subscribed Internet consumers

2.36 Service Attribute Tag Structural Modeling Connector

- Notes
    - Indicates a service’s attributes

2.37 Service Containment Space

- Notes
    - An area that identifies the aggregated child services contained in a parent composite service or service cluster.
    - This space can also define any collaboration of grouped services that are gathered to offer a solution

2.38 Service Stereotype

- Notes

    - A generic service that does not identify any particular service structure pattern

2.39 Service Typing Tag Structural Modeling Connector

- Notes
    - Classifies a services based on technical or business categories

2.40 Specified Contextual Modeling Connector

- Notes
    - A modeling activity that is performed on a service to reduce its abstraction level and decrease its scope of operations and capabilities

2.41 Subtracted Structural Modeling Connector

- Notes
    - Retires an analysis asset

2.42 Transformed Structural Modeling Connector

- Notes
    - Converts a structure of an analysis asset to a different structure

2.43 Trimmed Structural Modeling Connector

- Notes
    - Reduces the structural scale of an analysis asset

2.44 Unbound Structural Modeling Connector

- Notes
    - Cancels a contract between two or more analysis assets ?

2.45 Unified Structural Modeling Connector

- Notes
    - Merges two or more analysis assets into a single analysis asset ?

FW.3 SOA Reference Architecture [3]

3.1 Architectural Building Block (ABB)

- Relationships
  - Architectural Building Block **depends on** * Interaction
  - Architectural Building Block **involve** Method Activity
  - Architectural Building Block **aggregate** 1..* Capability
  - Architectural Building Block **aggregate** * Enabling Technology
  - Architectural Building Block **aggregate** * Solution Building Block
- Notes
  - A constituent of the architecture model that describes a single logical aspect of the overall model [10]. Each layer can be thought to contain a set of ABBs that define the key responsibilities of that layer. In addition, ABBs are connected to one another across layers and thus provide a natural definition of the association between layers. The particular connection between ABBs that recur consistently in order to solve certain classes of problems can be thought of as patterns of ABBs. These patterns will consist not only of a static configuration which represents the cardinality of the relationship between building blocks, but also the valid interaction sequences between the ABBs. In this SOA RA, each ABB resides in a layer, supports capabilities, and has responsibilities. It contains attributes, dependencies, constraints, and relationships with other ABBs in the same layer or different layer.

3.2 Architectural Decision

- Relationships
  - Architectural Decision **concerns** * KPI
  - Architectural Decision **concerns** * NFR
  - Architectural Decision **has** Layer
- Notes
  - A decision derived from the options. The architectural decision is driven by architectural requirements, and involves governance rules and standards, ABBs, KPIs, and Non-Functional Requirements (NFRs) to decide on standards and protocols to realize an instance of a particular logical ABB. This can be extended, based on the instantiation of the SOA RA to the configuration and usage of ABBs. Existing architectural decisions can also be re-used by other layers or ABBs.

3.3 Capability

- Relationships
  - Capability **has** 0..1 Capability
- Notes
  - An ability that an organization, person, or system possesses to deliver a product or service. A capability represents a requirement or category of requirements that fulfill a strongly cohesive set of needs. This cohesive set of needs or functionality is summarized

by name given to the capability.

- A capability, as defined by The Open Group, is: “an ability that an organization, person, or system possesses” [10]. From a TOGAF context: capabilities are typically expressed in general and high-level terms and typically require a combination of organization, people, processes, and technology to achieve. Marketing, customer contact, outbound telemarketing, etc. are illustrative examples for capabilities. The term capability can represent pure business capability such as Process Claim or Provision Service Request and can also represent technical capability such as Service Mediation or Content-Based Routing. Both business and technical capabilities are represented in SOA and enabled and supported by SOA.
- Using a capability modeling as part of the approach has some major advantages:
  1. Allows us to focus on the "what" rather than the "how". This supports an abstract approach that is focused on the requirements of the solution.
  2. Enables business capabilities to be aligned to the technical capabilities required to service them. Through the use of the SOA RA the associated enterprise-level and solution architectures can then be derived.
  3. Enables us to derive and re-balance the SOA roadmap in an agile fashion. For example, if an organization foresees the need for integrating services across different business units, it might require a certain set of SOA layers and Architecture Building Blocks (ABBs) to be enabled.
- The capability mapping process itself is out of scope of this document, and is normally a part of the organizational service modeling methodologies. The ability to derive the solution architecture from the SOA RA itself is within the scope of the SOA RA.
- A capability-based approach enables us to answer the question: “When do we need a particular SOA RA layer?” and to help facilitate making decisions when organizational priorities change. The SOA RA further helps us by determining whether there are inter-dependencies and technical requirements for a layer and its constituent building blocks, beyond those defined by business capabilities, to create a holistic set of capabilities which the SOA RA needs to satisfy.
- Services themselves have a contract element and a functional element. The service contract or service interface defines what the service does for consumers, while the functional element implements what a service is obligated to provide based on the service contract or service interface. The service contract is integrated with the underlying functional element through a component which provides a binding. This model addresses services exposing capabilities implemented through legacy assets, new assets, services composed from other services, or infrastructure services.

3.4 Enabling Technology

- Relationships
  - Enabling Technology **has** Layer
- Notes
  - A technical realization or instance of ABBs in a specific layer. Examples are web

services or REST.

3.5 Information Model

- Relationships
  - Information Model **has** Architectural Building Block
- Notes
  - A structural model of the information associated with ABBs including information exchange between layers and external services. The information model includes the metadata about the information being exchanged.

3.6 Interaction (Pattern)

- Relationships
  - Interaction **interacts** * Architectural Building Block
- Notes
  - An abstraction of the various relationships between ABBs. This includes diagrams, patterns, pattern languages, and interaction protocols.

3.7 Key Performance Indicator (KPI)

- Relationships
  - KPI **has** Architectural Building Block
- Notes
  - A KPI may act as input to an architectural decision.

3.8 Layer

- Relationships
  - Layer **has** 0..* Layer
  - Layer **has** 0..1 Method Activity
  - Layer **aggregates** Architectural Building Block
  - Layer **aggregates** * Capability
  - Layer **is associated with** Enabling Technology
  - Layer **is associated with** Architectural Decision
- Notes
  - An abstraction of a grouping of a cohesive set of ABBs, architectural decisions, interactions among ABBs, and interactions among layers, that support a set of related capabilities.
  - The layers in the SOA RA provide a convenient means of consolidating and categorizing the various capabilities and building blocks that are required to implement a given SOA.
  - The fulfillment of any service requirement may be achieved through the capabilities of a combination of one or more layers in the SOA Reference Architecture (SOA RA).
  - In order to describe each of the layers of the SOA RA we need the following for each layer:
    1. Introduction: Provide an overview of the layer itself.
    2. Requirements: Provide an understanding of the capabilities supported by the layer

and what they are (the answer to the "what does the layer do" question).
3. Logical Aspect: Provide an overview of the structural elements of the layer, applying the meta-model.
4. Interaction: Provide typical interactions among the Architecture Building Blocks (ABBs) within the layer and across layers.

- In general, we follow a theme where each layer has a part which supports a set of capabilities/ABBs which support the interaction of the layer with other elements in the SOA RA; a part which supports the actual capabilities that the layer must satisfy; and a part which supports the orchestration and management of the other ABBs to support the layer's dynamic, runtime existence.

3.9 Method Activity

- Notes
  - A set of steps that involve the definition or design associated with ABBs within a given layer. The method activity provides a dynamic view of how different ABBs within a layer will interact. Method activities can also be used to describe the interaction between ABBs across the layers, so that an entire interaction from a service invocation to service consumption is addressed.

3.10 Non-Function Requirement (NFR)

- Relationships
  - NFR **has** Architectural Building Block
- Notes
  - An NFR may act as input to an architectural decision. NFRs help address Service-Level Agreement (SLA) attributes (e.g., response time, etc.) and architectural cross-cutting concerns such as security.

3.11 Options

- Relationships
  - Options **have** 0..1 Architectural Building Block
  - Options **selected through** 0..1 Architectural Decision
- Notes
  - A collection of possible choices available in each layer that impact other artifacts of a layer. Options are the basis for architectural decisions within and between layers, and have concrete standards, protocols, and potentially solutions associated with them. An example of an option would be choosing SOAP or REST-style SOA services since they are both viable options. The selected option leads to an architectural decision.

3.12 Solution Building Block

- Notes
  - A runtime realization or instance of ABBs in a specific layer. A candidate physical solution for an ABB; e.g., a Commercial Off-The-Shelf (COTS) package such as a particular application server.

FW.4 Reference Architecture Foundation for Service Oriented Architecture (SOA-RAF) [4]

4.1 Participation in a SOA Ecosystem View

4.1.1 Actor

- Relationships
  - Actor **interacts with** SOA-based System
  - Actor **performs** Action
  - Actor **is bound by** Constraint
- Notes
  - A role played either by a Participant or its Delegate and that interacts with a SOA-based system.
  - Actors – whether stakeholder participants or delegates who act only on behalf of participants (without themselves having any stake in the actions that they have been tasked to perform) – are engaged in actions which have an impact on the real world and whose meaning and intent are determined by implied or agreed-to semantics.
  - Many actors interact with a SOA-based system, including software agents that permit people to offer, and interact with, services; delegates that represent the interests of other participants; or security agents charged with managing the security of the ecosystem.

4.1.2 Participant

- Relationships
  - Participant **has role as** Stakeholder
  - Participant **has role as** Actor
  - Participant **is represented by** Delegate
  - Participant **agree to** Contract
  - Participant **act within** Technical Assumptions
  - Participant **communicate through** Technical Assumptions
  - Participant **has role as** Third Party
  - Participant **has role as** Service Consumer
  - Participant **has role as** Service Provider
  - Participant **sends** Message
  - Participant **receives** 1..* Message
  - Participant **participates in** Message Exchange
  - Participants **express** Goals
  - Participants **agree to** Governance
  - Participants **recognize authority of** Leadership
  - Participants **agree to abide by** Governance Framework
- Notes
  - A person who plays a role both in the SOA ecosystem as a stakeholder and with the SOA-based system as an actor either

        - directly, in the case of a human participant; or
        - indirectly, via a delegate.
    - Participants (or their delegates) interact with a SOA-based system - in the role of actors - and are also members of a social structure – in the role of stakeholders.
    - A person who participates in a social structure as a stakeholder and interacts with a SOA-based system as an actor is defined as an ecosystem Participant
    - The concept of participant is particularly important as it reflects a hybrid role of a Stakeholder concerned with expressing needs and seeing those needs fulfilled and an Actor directly involved with system-level activity that result in necessary effects.
    - The hybrid role of Participant provides a bridge between social structures within the wider (real-world) ecosystem – in particular the world of the stakeholder – and the more specific (usually technology-focused) system – the world of the actor.
    - 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 participants may be part of the working group that codifies the governance framework and processes. When complete, the participants must acknowledge and agree to abide by the products generated through application of this structure.
    - Being distributed and representing different ownership domains, a SOA participant falls 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.

4.1.3 SOA-based System

- Notes
    - A technology system created to deliver a service within a SOA Ecosystem

4.1.4 SOA Ecosystem Model

- Relationships
    - SOA Ecosystem **aggregates** SOA-based System
- Notes
    - An environment encompassing one or more social structure(s) and SOA-based system(s) that interact together to enable effective business solutions
    - SOA services MUST be testable in the environment and under the conditions that can be encountered in the operational SOA ecosystem.
    - The distributed, boundary-less nature of the SOA ecosystem makes it infeasible to create and maintain a single testing substitute of the entire ecosystem to support testing activities. Test protocols MUST recognize and accommodate changes to and activities within the ecosystem.

4.1.5 Social Structure

- Relationships

    - Social Structure **aggregates** Stakeholder
    - Social Structure **provides context** SOA Ecosystem
    - Social Structure **constrains** Participant
    - Social Structure **constrains** Actor
    - Social Structure **constrains** Action
    - Social Structure **aggregates** 1..* Service Consumer
    - Social Structure **aggregates** 1..* Service Provider
- Notes
    - A nexus of relationships amongst people brought together for a specific purpose
    - The social structure is established with an implied or explicitly defined mission, usually reflected in the goals laid down in the social structure's constitution or other 'charter'.
    - A social structure will further its goals in one of two ways:
        - by acting alone, using its own resources
        - interaction with other structures and using their resources
    - Social structures are abstractions: they cannot directly perform actions with SOA-based systems – only actors can, whether they be participants acting under their own volition or delegates (human or not) simply following the instructions of participants.
    - Within a social structure, awareness can be encouraged or restricted through policies and these policies can affect participant willingness. The information about policies should be incorporated in the relevant descriptions. Additionally, the conditions for establishing contracts are governed within a social structure.
    - IT policy/contract mechanisms can be used by visibility mechanisms to provide awareness between social structures, including trust mechanisms to enable awareness between trusted social structures.

4.1.6 Stakeholder

- Relationships
    - Stakeholder **is bound by** Constitution
    - Stakeholder **approves** Constitution
    - Stakeholder **owns** 0..* Resource
    - Stakeholder **is bound by** Constraint
    - Stakeholder **is party to** Contract
    - Stakeholder **asserts** Policy
    - Stakeholder **act as or delegate responsibility to** Responsible Parties
    - Stakeholder **is bound by** Constraint
    - Stakeholder **defines** Policy
    - Stakeholder **empowered with** Authority
- Notes
    - A person with an interest (a 'stake') in a social structure.
    - Here we explicitly note that stakeholders and, thus, participants are people because machines alone cannot truly have a stake in the outcomes of a social structure. Delegates may be human and nonhuman but are not directly stakeholders. Stakeholders, both

Participants and Non-participants, may potentially benefit from the services delivered by the SOA-based system.
    - [...] a service is the mechanism that brings a SOA-based system capability together with stakeholder needs in the wider ecosystem.
    - The ecosystem includes stakeholders who are participants in the development, deployment, governance and use of a system and its services; or who may not participate in certain activities but are nonetheless affected by the system.
    - Any given person can be a stakeholder in multiple social structures and a social structure itself can be a stakeholder in its own right as part of a larger one or in another social structure entirely.

4.2 Social Structure in a SOA Ecosystem Model

4.2.1 Action

- Notes
    - The application of intent by an actor to cause an effect.
    - What constitutes an Action or an Activity will be a matter of context. For the SOA-RAF, an Action represents the smallest and most discrete activity that must be modeled for a given Viewpoint.

4.2.2 Business Functionality

- Notes
    - A defined set of business-aligned tasks that provide recognizable business value to consumer stakeholders and possibly others in the SOA ecosystem.
    - The idea of a service in a SOA ecosystem combines business functionality with implementation, including the artifacts needed and made available as IT resources.

4.2.3 Business Solution

- Notes
    - A set of defined interactions that combine implemented or notional business functionality in order to address a set of business needs.

4.2.4 Capability

- Relationships
    - Capability **ability to affect** Real World Effect
- Notes
    - An ability to deliver a real world effect.

4.2.5 Communication

- Notes
    - A process involving the exchange of information between a sender and one or more recipients and that ideally culminates in mutual understanding between them.
    - Message interpretation can itself be characterized in terms of semantic engagement: the proper understanding of a message in a given context.

    - We can characterize the necessary modes of interpretation in terms of a shared understanding of a common vocabulary (or mediation among vocabularies) and of the purpose of the communication. More formally, we can say that a communication has a combination of message and purpose.
    - In a SOA ecosystem, senders and recipients can be stakeholders, participants or actors, depending on whether execution context is being established or a specific interaction with the SOA-based system is in progress.

4.2.6 Composability

- Notes
    - The ability to combine individual services, each providing defined business functionality, so as to provide more complex business solutions.
    - Any composition can itself be made available as a service and the details of the business functionality, conditions of use, and effects are among the information documented in its service description.
    - Furthermore, the use of tools to auto-generate service software interfaces will not guarantee services than can effectively be used within compositions if the underlying code represents programming constructs rather than business functions. In such cases, services that directly expose the software details will be as brittle to change as the underlying code and will not exhibit the characteristic of loose coupling.

4.2.7 Constitution

- Relationships
    - Constitution **describes** Goal
    - Constitution **describes** Mission
    - Constitution **formalizes** Social Structure
- Notes
    - A set of rules, written or unwritten, that formalize the mission, goals, scope, and functioning of a social structure.
    - Every social structure functions according to rules by which people interact with each other within the 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. In still other cases, participants abide by the rules with some degree of reluctance. In all cases, the constitution may change over time; in those cases of implicit agreement, the change can occur quickly. Section 5.1 contains a detailed discussion of governance and SOA.

4.2.8 Constraint

- Notes
    - The constraints themselves represent some measurable limitation on the state or behavior of the object of the policy, or of those who interact with it.

4.2.9 Contract

- Relationships
  - Contract **aggregates** Constraint
- Notes
  - An agreement made by two or more participants (the contracting parties) on a set of conditions (or contractual terms) together with a set of constraints that govern their behavior and/or state in fulfilling those conditions.
  - Contracts are agreements among the participants. The contract may reconcile inconsistent policies asserted by the participants or may specify details of the interaction. Service level agreements (SLAs) are one of the commonly used categories of contracts.
  - While certain elements of contracts and contract compliance are likely private, public aspects of compliance should be reflected in the compliance record information referenced in the service description.

4.2.10 Delegate

- Relationships
  - Delegate **has role as** Actor
- Notes
  - A role played by a human or an automated or semi-automated agent and acting on behalf of a participant but not directly sharing the participant's stake in the outcome.
  - Note that automated agents are always delegates, in that they act on behalf of a participant.
  - In the different models of the SOA-RAF, the term actor is used when action is being considered at the level of the SOA-based system and when it is not relevant who is carrying out the action. However, if the actor is acting explicitly on behalf of a participant, then we use the term delegate.
  - The difference between a participant and a delegate is that a delegate acts on behalf of a participant and must have the authority to do so. Because of this, every social structure must clearly define the roles assigned to actors (whether participants or delegates) in carrying out activity within its domain.

4.2.11 Description

- Relationships
  - Description **describes** 1..* Resource
  - Description **references** 1..* Identifier
  - Description **extends** Resource
  - Description **aggregates** Associated Annotations
  - Description **aggregates** Provenance
  - Description **aggregates** Categorization
  - Description **composes** 1..* Description Class
  - Description **is visible to** Social Structure
- Notes

- Resources frequently have descriptions and the descriptions themselves may be considered resources.

4.2.12 Evidence

- Relationships
  - Evidence **accumulated from** Real World Effect

4.2.13 Goal

- Relationships
  - Goal **is compatible with** Mission
- Notes
  - Although goals are often expressed in terms of general ambitions for the social structure's work or of desired end states, objectives are expressed more formally in terms of specific, measurable, and achievable action required to realize those states.

4.2.14 Identifier

- Relationships
  - Identifier **identifies** Resource
- Notes
  - A sequence of characters that unambiguously indicates a particular resource.
  - Identifiers are assigned by social structures according to context, policies and procedures considered sufficient for that structure's purposes.
  - Resources are typically used or managed by different stakeholder groups, each of which may need to identify those resources in some particular way. As such, a given resource may have multiple identifiers, each valid for a different context.

4.2.15 Joint Action

- Relationships
  - Joint Action <<**require**>> Authorization
- Notes
  - The coordinated set of actions involving the efforts of two or more actors to achieve an effect.
  - The form of Activity that is of most interest within a SOA ecosystem is that involving Actions as defined below and their interaction across ownership boundaries (and thus involving interaction between more than one actor) – we call this joint action.
  - When a joint action is participated in with a service, the real world effect that results may be reported in the form of an event notification.
  - The notion of "joint" in joint action implies that you have to have a speaker and a listener in order to interact.

4.2.16 Mediator

- Structure
  - Service Descriptions
- Notes

- A role assumed by a participant to facilitate interaction and connectivity in the offering and use of services.
- A mediator using service descriptions may provide event notifications to both consumers and providers about information relating to the descriptions. One example of this 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.
- Another important characteristic of a SOA ecosystem is the ability to narrow visibility to trusted members within a social structure. Mediators for awareness may provide policy based access to service descriptions allowing for the dynamic formation of awareness between trusted members.
- Another common business model for awareness is maximizing awareness to those within the social structure, the traditional market place business model. A centralized awareness-mediator often arises as a provider for this global visibility, a gatekeeper of visibility so to speak.
- [...] mediators have motivations, and they may be selective in which information they choose to make available to potential consumers.

4.2.17 Mission

- Notes
  - A social structure may involve any number of persons as stakeholders and a large number of different relationships may exist among them. The organizing principle for these relationships is the social structure’s mission.

4.2.18 Need

- Relationships
  - Need **satisfied in** State
- Notes
  - A general statement expressed by a stakeholder of something deemed necessary.
  - A need may be formalized as one or more requirements that must be fulfilled in order to achieve a stated goal.

4.2.19 Non-Participant

- Relationships
  - Non-Participant **has role as** Stakeholder
- Notes
  - A person who plays no role as a participant in a social structure’s activities but nonetheless has an interest in, or is affected by, such activities.

4.2.20 Obligation

- Notes
  - A constraint that prescribes the actions that an actor must (or must not) perform and/or

the states the actor must (or must not) attain or maintain.
    - An obligation can also be a requirement to maintain a given state. This may range from a requirement to maintain a minimum balance on an account to a requirement that a service provider ‘remember’ that a particular service consumer is logged in.

4.2.21 Owner

- Notes
    - A role assumed by a participant who is claiming and exercising ownership over a service.

4.2.22 Ownership

- Notes
    - Ownership is defined as a relationship between a stakeholder and a resource, where some stakeholder (in a role as owner) has certain claims with respect to the resource.
    - Typically, the ownership relationship is one of control: the owner of a resource can control some aspect of the resource.
    - A set of claims, expressed as rights and responsibilities that a stakeholder has in relation to a resource; it may include the right to transfer that ownership, or some subset of rights and responsibilities, to another entity.
    - There may also be joint ownership of a resource, where the rights and responsibilities are shared.
    - Ownership is defined in relation to the social structure relative to which the given rights and responsibilities are exercised.

4.2.23 Ownership Boundary

- Notes
    - The extent of ownership asserted by a stakeholder or a social structure over a set of resources and for which rights and responsibilities are claimed and (usually) recognized by other stakeholders.
    - It is important to identify these ownership boundaries in a SOA ecosystem and successfully crossing them is a key aspect of establishing execution context.

4.2.24 Permission

- Notes
    - A constraint that identifies actions that an actor is (or is not) allowed to perform and/or the states in which the actor is (or is not) permitted.

4.2.25 Policy/Service Policies

- Relationships
    - Policy **expressed as** Constraint
    - Policy **reconciled in** Contract
    - Policy **may affect choice of** Semantics
    - Policy **may affect choice of** Technical Assumptions
    - Policy **tracked in** compliance Record

    - Policy **rules for** Authorization
    - Policy **drives** Management
- Notes
    - An expression of constraints made by a stakeholder that the stakeholder commits to uphold and, if desired or necessary, enforce. The constraints are usually stated as permissions and obligations that affect the behavior of stakeholders or of any actor acting on their behalf.
    - A service provider’s policy may become a service provider/consumer contract when a service consumer agrees to the provider’s policy.
    - Policies prescribe the conditions and constraints for interacting with a service and impact the willingness to continue visibility with the other participants. Whereas technical constraints 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 outside of the service description. The service description would reference the normative definition of the policy.
    - Policies may also be asserted by other participants, as illustrated by the model shown in Figure 19. Policies that are generally applicable to any interaction with the service are asserted by the service provider and included in the Service Policies section of the service description.
    - As noted in Figure 19, the policies asserted may be reflected as Technical Assumptions/ Constraints that available services or their underlying capabilities must be capable of meeting; it may similarly affect the semantics that can be used.
    - Note, even when policies relate to the perspective of a single participant, policy compliance can be measured and policies may be enforceable without contractual agreement with other participants.
    - Action level effects and policies must be reflected at the service level for service description to support visibility.
    - Policies asserted MAY be reflected as Technical Assumptions/Constraints that available services or their underlying capabilities MUST be capable of meeting.
    - [...] the actual behavior is expressed by means of policies of some form. Policies define the choices that stakeholders make; these choices are used to guide the actual behavior of the system to the desired behavior and performance.
    - As noted in Section 3.2.5.2, a policy is an expression of constraints that is promulgated by a stakeholder who has the responsibility of ensuring that the constraint is enforceable. In contrast, contracts are agreements between participants.
    - A policy constraint is a specific kind of constraint: the ontology of policies and contracts includes the core concepts of permission, obligation, owner, and subject. In addition, it may be necessary to be able to combine policy constraints and to be able to resolve policy conflicts.
    - accompanying textual definitions

        - Policy Framework: A policy framework is a language in which policy constraints may be expressed.
        - Logical Framework: A linguistic framework consisting of a syntax – a way of writing expressions – and a semantics – a way of interpreting the expressions.
        - Policy Ontology: A formalization of a set of concepts that are relevant to forming policy expressions.
    - The two primary kinds of policy constraint – permission and obligation – naturally lead to different styles of enforcement. A permission constraint must typically be enforced prior to the policy subject invoking the policy object. On the other hand, an obligation constraint must typically be enforced after the fact through some form of auditing process and remedial action.
    - Whenever it is possible that more than one policy constraint applies in a given situation, there is the potential that the policy constraints themselves are not mutually consistent.
    - Relation to SOA testing: Policies may also prescribe the standards with which an implementation must comply, as well as the qualifications of and restrictions on the actors. In addition to the functional requirements prescribing what an entity does, there may also be non-functional performance and/or quality metrics that state how well the entity performs.

4.2.26 Private State

- Notes
    - That part of an entity's state that is knowable by, and accessible to, only that entity.

4.2.27 Real World Effect

- Relationships
    - Real World Effect **change in** State
- Notes
    - A measurable change to the shared state of pertinent entities, relevant to and experienced by specific stakeholders of an ecosystem.
    - The real world effects that the service description definition support must be consistent with the technical assumptions/constraints. In particular, any Action Level Real World Effect MUST be reflected in the Service Level Real World Effect included in the description.

4.2.28 Reputation

- Relationships
    - Reputation **based on** Evidence

4.2.29 Requirement

- Notes
    - A formal statement of a desired result (a real world effect) that, if achieved, will satisfy a need.
    - This requirement can then be used to create a capability that in turn can be brought to

bear to satisfy that need. Both the requirement and the capability to fulfill it are expressed in terms of desired real world effect.

4.2.30 Resource

- Relationships
  - Resource **has** Description
  - Resource **owned by** Stakeholder
- Notes
  - A resource is generally understood as an asset: it has value to someone. Key to this concept in a SOA ecosystem is that a resource must be identifiable.
  - An identifiable entity that has value to a stakeholder.
  - Codified (but not implied) contracts, policies, obligations, and permissions are all examples of resources, as are capabilities, services, service descriptions, and SOA-based systems. An implied policy, contract, obligation or permission would not be a resource, even though it may have value to a stakeholder, because it is not an identifiable entity.
  - Testing resources MUST be described and their descriptions MUST be catalogued in a manner that enables their discovery and access.

4.2.31 Risk

- Relationships
  - Risk **assessed using** Reputation
  - Risk **assessed using** Evidence
- Notes
  - The private assessment or internal perception of the likelihood that certain undesirable real world effects will result from actions taken and the consequences or implications of such.
  - An actor perceiving risk may take actions to mitigate that risk.

4.2.32 Semantic Engagement

- Notes
  - The process by which an actor engages with a set of assertions based on that actor’s interpretation and understanding of those assertions.
  - An actor can potentially ‘understand’ an assertion in a number of ways, but it is specifically the process of arriving at a shared understanding that is important in the ecosystem. This process is semantic engagement and it takes place in different forms throughout the SOA ecosystem.

4.2.33 (Service) Consumer

- Relationships
  - Service Consumer **direct awareness** Service Provider
  - Service Consumer **gain awareness through** Mediator
  - Service Consumer **establish reachability of** Service
  - Service Consumer **has** Need

    - Service Consumer **publish or discover** Description
    - Service Consumer **uses** Resource
    - Service Consumer **considers** Service Description
    - Service Consumer **invokes** Service
    - Service Consumer **agrees with** Service Contract
- Notes
    - A role assumed by a participant who is interacting with a service in order to fulfill a need.

4.2.34 (Service) Provider

- Relationships
    - Service Provider **direct awareness** Service Consumer
    - Service Provider **facilitates awareness through** Mediator
    - Service Provider **offers capability through** Service
    - Service Provider **has** Capability
    - Service Provider **publish or discover** Description
    - Service Provider **uses** Resource
    - Service Provider **provides** Service Description
    - Service Provider **provides** Service
- Notes
    - A role assumed by a participant who is offering a service.

4.2.35 Shared State

- Notes
    - That part of an entity’s state that is knowable by, and may be accessible to, other actors.
    - It is the aggregation of the shared states of pertinent entities that constitutes the desired effect of a joint action. Thus the change to this shared state is what is experienced in the wider ecosystem as a real world effect

4.2.36 State

- Notes
    - The condition of an entity at a particular time.

4.2.37 Trust

- Relationships
    - Trust **assessed using** Reputation
    - Trust **assessed using** Evidence
- Notes
    - The private assessment or internal perception of one actor that another actor will perform actions in accordance with an assertion regarding a desired real world effect.
    - In a SOA ecosystem, the degree and nature of [...] trust is likely to be different for each actor, most especially when those actors are in different ownership boundaries.
    - The relevance of trust to interaction depends on the assessment of risk. If there is little or

no perceived risk, or the risk can be covered by another party who accepts responsibility for it, then the degree of trust may be less or not relevant in assessing possible actions.

4.2.38 Trusted Actor

- Relationships
  - Trusted Actor **has** Reputation

4.2.39 Trusting Actor

- Relationships
  - Trusting Actor **determines** Willingness
  - Trusting Actor **assesses** Trust
  - Trusting Actor **assesses** Risk
  - Trusting Actor **is associated with (Trust)** Trusted Actor
  - Trusting Actor **is associated with (Risk)** Trusted Actor

4.2.40 Willingness

- Notes
  - The internal commitment of a human actor (or of an automated non-human agent acting on a participant's behalf) to carry out its part of an interaction.
  - Important considerations in establishing willingness are both trust and risk.
  - 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.
  - Information used to determine willingness is provided by Description (see Section 4.1.1).
  - 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.
  - 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.

4.3 Realization of a SOA Ecosystem View

4.3.1 Action

- Relationships
  - Action **results in** Action Level Real World Effect
  - Action **complies with** Action Level Policy
  - Action **availability reflected in** Action Presence
  - Action **satisfies** Precondition
  - Action **recorded in** Interaction Log
  - Action **performed against** Endpoint

    - Action **invoked via** Message
    - Action **causes** Real World Effect
- Notes
    - component must exist to execute “set of operations” to generate prescribed real world effects
    - [...]
        - (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.
    - Actions make use of structure and semantics as defined in the information model to describe its legal messages.
    - 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.
    - Action level effects and policies must be reflected at the service level for service description to support visibility.
    - Actions MAY have associated policies stating conditions for performing the action, but these MUST be reflected in and be consistent with the policies made visible at the service level and thus the description of the service as a whole.
    - Each action may have its own endpoint and also its own protocols associated with the endpoint and whether there is presence for the action through that endpoint.

4.3.2 Action Level Policy

- Relationships
    - Action Level Policy **is reflected in** Service Policy
- Notes
    - 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 (or subsumed by) the Service 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 Policies are included in the service description.
    - The unambiguous expression of action level policies and real world effects as service counterparts is necessary to adequately describe what constitutes the service interaction.

4.3.3 Action Level Real World Effect

- Relationships
    - Action Level Real World Effect **is reflected in** Service Level Real World Effect
- Notes
    - A specific change in the state or the information returned as a result of interacting through a specific action.
    - [...] any Action Level Real World Effects must be reflected in the Service Level Real World Effect included in the service description.

4.3.4 Action Model

- Relationships
  - Action Model **aggregates** Action
- Structure
  - Permissible Actions
  - Operations
- Notes
  - Actions relevant to the Service Interface
  - The action model identifies the multiple actions an actor can perform against a service and the actor would perform these in the context of the process model as specified or referenced under the Service Interface Description portion of Service Description.
  - [...] the action model characterizes the “permissible set of actions that may be invoked
  - against a service.” We extend that notion here to include events and that messages are intended for invoking actions or for notification of events.

4.3.5 Action Presence

- Relationships
  - Action Presence **extends** Presence
- Notes
  - The availability of an action is reflected in the Action Presence and each Action Presence contributes to the overall Service Presence; this is discussed further in Section 4.2.2.3. Each action has its own endpoint and protocols are associated
  - Presence for an action means an actor can initiate it and is independent of whether the preconditions are satisfied.

4.3.6 Annotations from 3rd Parties

4.3.7 Annotations from Owners

4.3.8 Associated Annotations

- Relationships
  - Associated Annotations **aggregates** Annotations from Owners
  - Associated Annotations **aggregates** Annotations from 3rd Parties
- Notes
  - The general description instance may also reference associated documentation that is in addition to that considered necessary in this model.

4.3.9 Awareness

4.3.10 Behavior Model

- Relationships
  - Behavior Model **aggregates** Action Model
  - Behavior Model **aggregates** Process Model
- Notes
  - A well-defined service Behavior Model (as defined in the SOA-RM) MUST be provided

that:

- characterizes the knowledge of the actions invoked against the service and events that report real world effects as a result of those actions;
- characterizes the temporal relationships and temporal properties of actions and events associated in a service interaction;
- describe activities involved in a workflow activity that represents a unit of work;
- describes the role (s) performed in a service-oriented business process or service-oriented business collaboration;
- is both human readable and machine processable;
- is referenceable from the Service Description artifact.

4.3.11 Categorization

- Relationships
  - Categorization **aggregates** Classification Terms
  - Categorization **aggregates** Keywords

4.3.12 Classification Terms

- Notes
  - 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 Foundation 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 of any terms or keywords should be referenced, whether that be a representation in a formal ontology language, a pointer to an online dictionary, or any other accessible source.

4.3.13 Compliance Record

- Notes
  - The use of metrics to evaluate compliance and the results of compliance evaluation are maintained in compliance records and the means to access the compliance records is included in the Service Policies portion of the service description.

4.3.14 Contract

- Relationships
  - Contract **tracked in** Compliance Record

4.3.15 Defaults

- Relationships
  - Defaults **aggregates** Infrastructure
  - Defaults **aggregates** Processes

4.3.16 Defining Conditions

- Relationships

    - Defining Conditions **aggregates** Technical Assumptions
    - Defining Conditions **aggregates** Semantics

4.3.17 Dependencies

- Relationships
    - Dependencies **reflect external constraints on** Process Model
- Notes
    - 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 service as a whole may provide fallback if a dependency is not met, and the service description may indicate functionality without explicitly containing details of how dependencies are satisfied or otherwise mitigated.

4.3.18 Description Class

- Relationships
    - Description Class **aggregates** 0..* Description Class
    - Description Class **composes** 1..* Value Specifier
    - Description Class **aggregates** 0..1 Specifier Attributes

4.3.19 Event

- Relationships
    - Event **reports** Real World Effect
- Notes
    - An event is an occurrence that is of interest to some participant; in our case when some real world effect has occurred.

4.3.20 Endpoint

- Relationships
    - Endpoint **receive messages conforming to** Protocols
- Notes
    - Each action has its own endpoint and protocols are associated with the endpoint. The endpoint and service presence are the overall Service Presence; this is discussed further in Section 4.2.2.3. Each action has its own endpoint and protocols are associated with the endpoint.
    - A reference-able entity, processor or resource against which an action can be performed.

4.3.21 Execution Context

- Relationships
    - Execution Context **aggregates** Policies & Contracts
    - Execution Context **aggregates** Defaults
    - Execution Context **aggregates** Defining Conditions
- Notes
    - The service description must provide sufficient information to support service visibility, including the willingness 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 that encapsulate the necessary alignment form the basis upon which interactions may proceed – in the Reference Model, this collection of agreements and the necessary environmental support establish the execution context.
    - Any other items not explicitly noted in the model but which are needed to set the environment, would also be included in the execution context.
    - 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.
    - An execution context can act as a template for identical or similar interactions. Any given execution context can define the conditions of future interactions.
    - The mechanisms that make-up the execution context in secure SOA-based systems SHOULD:
        - provide protection of the confidentiality and integrity of message exchanges;
        - be distributed so as to provide available 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 means or channels;
        - have a framework for resolving conflicts between security policies.

4.3.22 Functions

- Relationships
    - Functions **produce** Service Level Real World Effect

4.3.23 Information Model

- Relationships
    - Information Model **aggregates** Semantics
    - Information Model **aggregates** Structure
- Structure
    - <<messages>> Message Structure
    - <<messages>> Message Semantics
    - <<data>> Data Structure
    - <<data>> Data Semantics
- Notes
    - Note we distinguish the structure and semantics of the message from that of the underlying protocol that conveys the message. The message structure may include nested structures that are independently defined, such as an enclosing envelope structure and an enclosed data structure.
    - [...] 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.
    - A well-defined service Information Model MUST be provided that:
        - describes the syntax and semantics of the messages used to denote actions and events;
        - describes the syntax and semantics of the data payload(s) contained within messages; documents exception conditions in the event of faults due to network outages, improper message/data formats, etc.;
        - is both human readable and machine processable;
        - is referenceable from the Service Description artifact.

4.3.24 Infrastructure

- Notes
    - Infrastructure MUST be specified that provides mechanisms to support service interaction, including but not limited to:
        - mediation within service interactions based on shared semantics;
        - translation and transformation of multiple application-level protocols to standard network transport protocols;
        - auditing and logging that provide a data store and mechanism to record information related to service interaction activity such as message traffic patterns, security violations, and service contract and policy violations
        - security that provides authorization and authentication support, etc., which provide protection against common security threats in a SOA ecosystem;
        - monitoring such as hardware and software mechanisms that both monitor the performance of systems that host services and network traffic during service interaction, and are capable of generating regular monitoring reports.

4.3.25 Interaction

- Relationships
    - Interaction **described by** Interaction Description
    - Interaction **composed of** Action
    - Interaction **proceeds within** Execution Context
    - Interaction **depends on** Service Visibility
    - Interaction **depends on** Service Interface Description
- Notes
    - (Service) interaction [...] has a direct dependency on the visibility of the service as well as its implementation-neutral interface (see Figure 27).

4.3.26 Interaction Description

- Relationships
    - Interaction Description **extends** Description
    - Interaction Description **aggregates** Execution Context
    - Interaction Description **aggregates** Interaction Log

4.3.27 Interaction Log

- Notes
  - 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. This sequence should follow the Process Model but can include details beyond those specified there.
  - The execution context can be thought of as a 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.
  - 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.
  - The interaction log provides a detailed trace for a specific interaction, and its reuse is limited to duplicating that interaction.

4.3.28 Keywords

- Notes
  - 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 Foundation 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 of any terms or keywords should be referenced, whether that be a representation in a formal ontology language, a pointer to an online dictionary, or any other accessible source.

4.3.29 Message

- Relationships
  - Message **conveys intent for** Action
  - Message **addressed to** Endpoint
  - Message **conveys** Action
  - Message **conveys** Event
- Notes
  - Action is typically invoked via a Message where the structure and processing details of the message conform to an identified Protocol and is directed to the address of the identified endpoint, and the message payload conforms to the service Information Model.
  - Interaction proceeds through messages and thus it is the syntax and semantics of the messages with which we are here concerned. A common approach is to define the structure and semantics that can appear as part of a message; then assemble the pieces into messages; and, associate messages with actions.
  - An interaction is through the exchange of 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 that must be assessed and/or processed (e.g. if an error exists, that part that covers the error processing would be invoked).

4.3.30 Message Exchange

- Relationships
  - Message Exchange **aggregates** Message
- Notes
  - Message exchange is the means by which service participants (or their delegates) 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.
  - A message exchange is used to affect an action when the messages contain the appropriately formatted content, are directed towards a particular action in accordance with the action model, and the delegates involved interpret the message appropriately.
  - A message exchange is also used to communicate event notifications.
  - The means by which joint action and event notifications are coordinated by service participants (or delegates).

4.3.31 Message Exchange Pattern

- Relationships
  - Message **participates in** Message Exchange Pattern
- Notes
  - 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

4.3.32 Message Payload

- Relationships
  - Message Payload **carried by** Message

4.3.33 Metrics

- Relationships
  - Metrics **aggregates** Performance Metrics
  - Metrics **aggregates** Nonperformance Metrics
  - Metrics **provide operational values for** Compliance Record
  - Metrics **provide measurable quantities for** Policy
  - Metrics **provide measurable quantities for** Contract
  - Metrics **available to** Participant
  - Metrics **available to** Leadership
  - Metrics **available to** Governance Body
  - Metrics **inform** Management
  - Metrics **provide measurable quantities for** Regulation

    - Metrics **provide measurable quantities for** Rule
- Notes
    - [...] 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 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.

4.3.34 Nonperformance Metrics

4.3.35 Operation

- Notes
    - mentioned textually only
    - When a message is used to invoke an action, the correct interpretation typically requires the receiver to perform an operation, which itself invokes a set of private, internal actions. These operations represent the sequence of (private) actions a service must perform in order to validly participate in a given joint action.
    - The sequence of actions a service must perform in order to validly participate in a given joint action.

4.3.36 Other Descriptions

- Relationships
    - Other Descriptions **extends** Description

4.3.37 Participant Description

- Relationships
    - Participant Description **extends** Description

4.3.38 Performance Metrics

4.3.39 Pointer to Description Resolving to Value Set

4.3.40 Pointer to Value Set

4.3.41 Precondition

- Notes
    - An action may have preconditions where a Precondition is something that must be in place before an action can occur, e.g. confirmation of a precursor action. Whether preconditions are satisfied is evaluated when an actor tries to perform the action and not before.
    - However, the successful completion of the action may depend on whether its preconditions were satisfied. The service as a whole may provide fallback if a precondition is not met, and the service description may indicate functionality without explicitly containing details of how preconditions are satisfied or otherwise mitigated.

4.3.42 Presence

- Notes

- The measurement of reachability of a service at a particular point in time.
- Presence is determined by interaction through a communication protocol. Presence may not be known in many cases until the interaction begins. To overcome this problem, IT mechanisms may make use of presence protocols to provide the current up/down status of a service.
- 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.

4.3.43 Process Model

- Relationships
  - Process Model **aggregates** Action
- Structure
  - Message Exchange Patterns
  - <<orchestration>> Orchestration Properties
  - <<orchestration>> Orchestration Relationships
  - <<choreography>> Choreography Properties
  - <<choreography>> Choreography Relationships
- Notes
  - Temporal sequence of Actions
  - [...] the Process Model may imply Dependencies for succeeding steps in a process, […]
  - For a given business function, there is a corresponding process model, where any process model may involve multiple actions.
  - 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 the process that must be followed (the process model). The remaining question is how the description information for structure and semantics enable interaction [see Message].
  - [...] 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.
  - [...] MEPs are a key element of the Process Model.

4.3.44 Processes

- Notes
  - Collaborations can include processes (for example, when one actor executes a particular activity according to the predefined steps of a process) as much as processes can include collaborations (a predefined step of a particular process may include agreed-upon activities provided by other participants).
  - Business processes have temporal properties and can be short-lived or long-lived. Further, these processes may involve many participants and may be important considerations for the consumer of a service-oriented business process.
  - For business processes implemented as SOA-based services, ensuring that the meta-level

aspects of the service-oriented business process are included in its Service Description can provide needed insight for the consumer.

4.3.45 Property

- Notes
  - 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.

4.3.46 Protocols

- Relationships
  - Protocol **communication description for** Message
- Notes
  - A structured means by which details of a service interaction mechanism are defined.
  - A protocol defines a structured method of communication.

4.3.47 Provenance

- Relationships
  - Provenance **aggregates** Responsible Parties
  - Provenance **aggregates** Resource History
- Notes
  - [...] Provenance as related to the Description class provides information that reflects on the quality or usability of the subject.
  - Provenance specifically identifies the stakeholder (human, defined role, organization, etc.) who 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.

4.3.48 Resource History

4.3.49 Responsible Parties

- Notes
  - […] Responsibilities may be directly assumed by the stakeholder who owns a resource (see Section 3.2.4.2) 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 stakeholder identified under Responsible Parties; for example, one stakeholder may be responsible for code maintenance while another is responsible for provisioning of the executable code.

4.3.50 Semantics

- Relationships
  - Semantics **vocabulary understood by** Message Payload
- Notes
  - The further qualifying of Structure and Semantics in the Set Attributes allows for

flexibility in defining the form of the associated values.

4.3.51 Service

- Relationships
  - Service **interaction results in** Real World Effect
  - Service **receives** Message
  - Service **sends** Message
  - Service **performs** Action
  - Service **senses** Event
  - Service **participates in** Message Exchange
- Notes
  - Services are implemented through a combination of processes and collaboration.
  - Composition of services is the act of aggregating or ‘composing’ a single service from one or more other services.
  - While the service composition is opaque from the Consumer Delegate’s perspective, it is transparent to the service owner. This transparency is necessary for service management to properly manage the dependencies between the services used in constructing the composite service—including managing the service’s lifecycle.
  - The principles involved in the composition of services (including but not limited to loose coupling, selective transparency and opacity, dynamic interactions) […]
  - Whereas a service can execute according to a predefined business process determined by one organization, service composition can also be accomplished as a cooperation, or business collaboration, between actors in different organizations and systems.
  - Common security services SHOULD include the ability for:
    - authentication and establishing/validating credentials
    - retrieval of authorization credentials (attribute services);
    - enforcing access control policies;
    - intrusion detection and prevention;
    - auditing and logging interactions and security violations.
  - Services SHOULD be available to support automated testing and regression testing.
  - Services SHOULD be available to facilitate updating service description by authorized participants who has performed testing of a service.

4.3.52 Service Description

- Relationships
  - Service Description <<artifact>> **extends** Description
  - Service Description <<artifact>> **aggregates** Metrics
  - Service Description <<artifact>> **aggregates** Service Policies
  - Service Description <<artifact>> **aggregates** Service Interface Description
  - Service Description <<artifact>> **aggregates** Service Reachability
  - Service Description <<artifact>> **aggregates** Service Functionality
  - Service Description **defines** Service
  - Service Description **transmits to** Service Contract

- Notes
  - Service Description is an artifact and requires components to store, find, access, manage artifact.
  - Service Description [is] a subclass of the general Description class. As well as describing a Resource (as we saw in Section 3.2.4.1), a Description is also a subclass of the Resource class. In addition, each resource is assumed to have a description.
  - Service Description identifies available Metrics and how to access; requires components to gather, store, provide metrics access
  - Service Description identifies known Policies; requires mechanisms to create and maintain policies (may be outside SOA), components to store, find, access, manage Policies.
  - An adequate service description must provide a consumer with information needed to determine if the service policies, the (business) functions, and service-level real world effects are of interest, and there is nothing in the technical constraints that preclude use of the service.
  - The service description is not intended to be isolated documentation but rather an integral part of service use. Changes in service description should immediately be made known to consumers and potential consumers.
  - 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.
  - While the representation shown in Figure 15 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) [DCMI] and ISO 11179 [ISO 11179], especially Part 5.
  - Description makes use of defined semantics, where the semantics may be used for categorization or providing other property and value information for description classes. In such cases, the service description MUST have:
    - 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;
    - […]
  - The Service Description MUST provide a consumer with information needed to: determine the service functionality; the conditions under which interaction can proceed (service policies and process model); the intended Service Level Real World Effects; any technical constraints that might preclude use of the service.
  - The service description (as also enumerated under governance) MUST have:
    - description of policies, including a unique identifier for the policy and a sufficient, preferably machine processable, representation of the meaning of terms used to describe the policy, its functions, and its effects;

        - a method to enable searching for policies that best meet the search criteria specified by the service participant; where the discovery mechanism has access to the individual policy descriptions, possibly through some repository mechanism;
        - […]
    - The service description definition (as also partially enumerated under governance) MUST have:
        - infrastructure monitoring and reporting information on SOA resources;
        - possible interface requirements to make accessible metrics information generated;
        - 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.
    - [...] the service description definition MUST have:
        - one or more discovery mechanisms that enable searching for described resources that best meet the criteria specified by a service participant;
        - tools to appropriately track use of the descriptions by service participants and notify them when a new version of the description is available.
    - The service description MUST provide sufficient information to support service visibility, including the willingness 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
    - 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.
    - Service descriptions SHOULD include a sufficiently rich meta-structure to unambiguously indicate which security policies are required and where policy options are possible.
    - The service description identifies several management objects such as a set of service interfaces and related set of SLAs.
    - In the service description, a service consumer can find references to management policies, SLA metrics, and the means of accessing measured values that together increase assurance in the service quality. At the same time, service description is an artifact that must be managed.
    - [...] management of service description contains, among others, management of the service description presentations, the life-cycles of the service descriptions, service description distribution practices and storage of the service descriptions and related service contracts.
    - Relation to SOA testing:
        - Service functionality is an early and ongoing focus of testing to ensure the service accurately reflects the described functionality and the described functionality

accurately addresses the consumer needs.
    - Policies constraining service development, such as coding standards and best practices, require appropriate testing and auditing during development to ensure compliance. Policies that define conditions of use are initially tested during service development and are continuously monitored during the operational lifetime of the service.
    - At any point where the interface is modified or exposes a new resource, the message exchange should be monitored both to ensure the message reaches its intended destination and it is parsed correctly once received.
    - The service interface is also tested when the service endpoint changes. Functioning of a service endpoint at one time does not guarantee it is functioning at another time, e.g. the server with the endpoint address may be down, making testing of service reachability a continual monitoring function through the life of the service's use of the endpoint.
    - Metrics are a key indicator for consumers to decide if a service is adequate for their needs. For instance, the average response time or the recent availability can be determining factors even if there are no rules or regulations promulgated through the governance process against which these metrics are assessed. Testing will ensure that the metrics access indicated in the service description is accurate.

4.3.53 Service Functionality

- Relationships
    - Service Functionality **aggregates** Functions
    - Service Functionality **aggregates** Service Level Real World Effect
    - Service Functionality **aggregates** Technical Assumptions
- Notes
    - [...] service functionality and performance metrics (discussed in Section 4.1.1.3.4) describe what can be expected as a result of interacting with a service. Service Functionality, shown in Figure 16 as part of the overall Service Description model and extended in Figure 18, is a clear expression of service function(s) and the real world effects of invoking the function. The Functions represent business activities in some domain that produce the desired real world effects.
    - Elements of Service Functionality may be expressed as natural language text, reference an existing taxonomy of functions or other formal model.

4.3.54 Service Interface Description

- Relationships
    - Service Interface Description **composes** Behavior Model
    - Service Interface Description **composes** Information Model
- Notes
    - [...] the service interface is the means for interacting with a service.
    - [...] the service interface supports an exchange of messages, where
        - the message conforms to a referenceable message exchange pattern (MEP, covered

below in Section 4.3.3.1),
        - the message payload conforms to the structure and semantics of the indicated information model,
        - the messages are used to denote events related to or 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.
    - The Service Interface Description element as shown in Figure 17 includes the information needed to carry out this message exchange in order to realize the service behavior described. In addition to the Information Model that conveys the Semantics and Structure of the message, the Service Interface Description indicates what behavior can be expected through interactions conveyed in the Action and Process Models.

4.3.55 Service Level Real World Effect

- Relationships
    - Service Level Real World Effect **must be consistent with** Technical Assumptions
- Notes
    - A specific change in the state or the information returned as a result of interacting with a service.

4.3.56 Service Presence

- Relationships
    - Service Presence **aggregates** Action Presence
    - Service Presence **extends** Presence

4.3.57 (Service) Reachability

- Relationships
    - Service Reachability **aggregates** Protocols
    - Service Reachability **aggregates** Presence
    - Service Reachability **aggregates** Endpoint
- Notes
    - Service reachability, as modeled in Section 4.2.2.3 enables service participants to locate and interact with one another. To support service reachability, the service description should indicate the endpoints (also modeled and defined in that section) to which a service consumer can direct messages to invoke actions and the protocol to be used for message exchange using that endpoint.
    - As generally applied 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.
    - Service reachability enables service participants to locate and interact with one another.
    - [...] reachability relates to each action as well as applying to the service/business as a whole.

4.3.58 Service Visibility

- Relationships
  - Service Visibility **composes** Awareness
  - Service Visibility **composes** Willingness
  - Service Visibility **composes** Reachability

4.3.59 Set Attributes

- Relationships
  - Set Attributes **aggregates** Structure
  - Set Attributes **aggregates** Semantics

4.3.60 Specifier Attributes

- Relationships
  - Specifier Attributes **aggregates** 0..1 Identifier
  - Specifier Attributes **aggregates** 0..1 Provenance
  - Specifier Attributes **aggregates** 0..1 Value Set Source

4.3.61 Structure

- Relationships
  - Structure **syntax understood by** Message Payload
- Notes
  - The further qualifying of Structure and Semantics in the Set Attributes allows for flexibility in defining the form of the associated values.

4.3.62 Technical Assumptions

- Notes
  - Technical constraints 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 constraints are related to the underlying capability accessed by the service. In any case, the real world effects must be consistent with the technical assumptions/constraints.

4.3.63 Third Party

4.3.64 Value

- Notes
  - 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.

4.3.65 Value Set

- Relationships
  - Value Set **aggregates** Semantics
  - Value Set **aggregates** Set Attributes
  - Value Set **composes** Value

    - Value Set **composes** Property
- Notes
    - 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 context conveying the meaning of the property within the execution context, where the semantic context 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.

4.3.66 Value Set Source

4.3.67 Value Specifier

- Relationships
    - Value Specifier **aggregates** 0..1 Specifier Attributes
    - Value Specifier **composes** 0..* Pointer to Description Resolving to Value Set
    - Value Specifier **composes** 0..* Pointer to Value Set
    - Value Specifier **composes** 0..* Value Set
- Notes
    - 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 person (individual or organization) who 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.
    - Provenance for a value specifier identifies who is responsible for choosing and assigning values to the value sets that comprise the value specifier.
    - If the value specifier is contained within a higher-level component (such as Service Description containing Service Functionality), the component may assume values from the attributes of its container.

4.4 Ownership in a SOA Ecosystem View

4.4.1 Attributes

4.4.2 Authentication

- Notes
  - Authentication is concerned with adequately identifying actors in a potential interaction or joint action. Various mechanisms and protocols can be used to achieve this goal. A combination of identifiers (as discussed in section 3.2.4.1) and other attributes of an actor is typically used to achieve this. The set of attribute values that claim to identify a specific actor are matched against the set of reference values expected for that actor and that are maintained by some trusted authority. If the comparison results in a sufficient match, authentication has been achieved. Which specific set of attributes is considered an adequate basis for comparison will be context-dependent [...]

4.4.3 Authority

4.4.4 Authorization

- Relationships
  - Authorization <<**require**>> Authentication
  - Authorization <<**assess**>> Role
  - Authorization <<**assess**>> Reputation
  - Authorization <<**assess**>> Attributes
- Notes
  - Authorization concerns the legitimacy of the interaction, providing assurance that the actors have permission to participate in the interaction. Authorization refers to the means by which a stakeholder may be assured that the information and actions that are exchanged are either explicitly or implicitly approved.
  - The role of access control policy for security is to permit stakeholders to express their choices. In Figure 40, such a policy is a written constraint and the role, reputation, and attribute assertions of actors are evaluated according to the constraints in the authorization policy. A combination of security mechanisms and their control via explicit policies can form the basis of an authorization solution.
  - The roles and attributes which provide a participant's credentials are expanded to include reputation. Reputation often helps determine willingness to interact; for example, reviews of a service provider will influence the decision to interact with the service provider. The roles, reputation, and attributes are represented as assertions measured by authorization decision points.

4.4.5 Enforcement

- Relationships
  - Enforcement **applies incentives or penalties** Participant
- Notes
  - The leadership in its relationship with participants has certain options that can be used

for enforcement. A common option may be to affect 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.
    - [...] enforcement does not strictly need to be negative consequences. Management can use metrics to identify exemplars of compliance and leadership can provide options for rewarding the participants. The governance body defines awards or other incentives.

4.4.6 Event Monitoring Manageability

- Notes
    - Event Monitoring Manageability allows managing the categories of events of interest related to services and reporting recognized events to the interested stakeholders. Such events may be the ones that trigger service invocations as well as execution of particular functionality provided by the service.
    - Event Monitoring Manageability is a key lower-level manageability aspect, in which the service provider and associated stakeholders are interested. Monitored events may be internal or external to the SOA ecosystem.

4.4.7 Explicit Service Contract

- Notes
    - In the case of business services, it is anticipated that the service contract may take an explicit form and the agreement between business consumer and business service provider is formalized. Formalization requires up-front interactions between service consumer and service provider. In many business interactions, especially between business organizations within or across corporate boundaries, a consumer must have a contractual assurance from the provider or wants to explicitly indicate choices among alternatives, […]
    - An explicit service contract always requires a form of interaction between the service consumer and the service provider prior to the service invocation. This interaction may regard the choice or selection of the subset of the elements of the service description or other alternatives introduced through the formal agreement process that would be applicable to the interaction with the service and affect related joint action.
    - Any form of explicit contract couples the service consumer and provider.
    - While explicit contracts may be necessary or desirable in some cases, such as in supply chain management, commerce often uses a mix of implicit and explicit contracts, and a service provider may offer (via service description) a conditional shift from implicit to explicit contract.

4.4.8 Governance

- Relationships
    - Governance **expressed through** Policies
    - Governance **attempts to satisfy common subset of** Goals
    - Governance **has jurisdiction over** Participants

    - Governance **sets context for** Management
- Notes
    - The prescription of conditions and constraints consistent with satisfying common goals and the structures and processes needed to define and respond to actions taken towards realizing those goals.
    - A policy is the formal characterization of the conditions and constraints that governance deems as necessary to realize the goals which it is attempting to satisfy. Policy may identify required conditions or actions or may prescribe limitations or other constraints on permitted conditions or actions.
    - Part of the purpose of governance is to arbitrate among diverse goals of participants and the diverse policies articulated to realize those goals.
    - For governance to have effective jurisdiction over participants, there must be some degree of agreement by all participants that they will abide by the governance mandates.
    - In addition to tiered governance, there may be multiple governance chains working in parallel.
    - The parallel chains may just be additive or may be in conflict and require some harmonization.
    - Governance is expressed through policies and assumes multiple use of focused policy modules that can be employed across many common circumstances. The following are thus REQUIRED:
        - descriptions to enable the policy modules to be visible, where the description SHOULD include a unique identifier for the policy as well as a sufficient, and preferably machine process-able, representation of the meaning of terms used to describe the policy, its functions, and its effects;
        - […]
    - 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:
        - [...]
        - SOA services to access automated implementations of the Governance Processes
    - Governance policies are made operational through rules and regulations. This REQUIRES:
        - descriptions to enable the rules and regulations to be visible, where the description SHOULD include 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;
        - [...]
        - SOA services to access automated implementations of the Governance Processes.
    - Governance relies on metrics to define and measure compliance. This REQUIRES:
        - 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.

- The primary role of governance in the context of a SOA ecosystem is to foster an atmosphere of predictability, trust, and efficiency, and it accomplishes this by allowing the stakeholders to negotiate and set the key policies that govern the running of the SOA-based solution.

4.4.9 Governance Body

- Relationships
  - Governance Body **promulgates** Rule
  - Governance Body **delegates operational details to** Management
  - Governance Body **defines responses**
- Notes
  - 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, governance body may delegate the operational details of governance to management.
  - 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 takes care to expand and evolve the mandated standards in a predictable manner and with sufficient technical guidance that new services are able to coexist as much as possible with the old, and changes to standards do not cause major disruptions.

4.4.10 Governance Framework

- Relationships
  - Governance Framework **form structure for** Governance Processes
- Notes
  - The set of organizational structures that enable governance to be consistently defined, clarified, and as needed, modified to respond to changes in its domain of concern.
  - The governance framework and processes are often documented in the constitution or charter of a body created or designated to oversee governance.
  - 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.

4.4.11 Governance Processes

- Relationships
  - Governance Processes **define changing of** Governance Framework
  - Governance Process **are instantiated through** Rule
- Notes

- The defined set of activities performed within the Governance Framework to enable the consistent definition, application, and as needed, modification of rules that organize and regulate the activities of participants for the fulfillment of expressed policies.
- The governance framework and processes are often documented in the constitution or charter of a body created or designated to oversee governance.

4.4.12 Implicit Service Contract

- Notes
  - [...] an implicit service contract is an agreement (1) on the consumer side with the terms, conditions, features and interaction means specified in the service description "as is" or (2) a selection from alternatives that are made available through mechanisms included in the service description, and neither of these require any a priori interactions between the service consumer and the service provider.

4.4.13 IT Governance

- Relationships
  - IT Governance **coordinate between** Other Governance
  - IT Governance **extends** Governance
  - SOA Governance **coordinate between** IT Governance

4.4.14 Key Performance Indicator (KPI)

4.4.15 Leadership

- Relationships
  - Leadership **exercises authority over** Participants
  - Leadership **considers diverse** Goals
  - Leadership **initiates and champions** Governance
  - Leadership **generates consistent** Policies
  - Leadership **architect a** Governance Framework
  - Leadership **defines** Governance Processes
  - Leadership **charters** Governance Body
  - Leadership **defines options** Enforcement
- Notes
  - The entity having the responsibility and authority to generate consistent policies through which the goals of governance can be expressed and to define and champion the structures and processes through which governance is realized.
  - [...] the entity with governance authority is designated the Leadership. This is someone, possibly one or more of the participants, which participants recognize as having authority for a given purpose or over a given set of issues or concerns.
  - 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 is applied.
    - 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 make it clear to participants that expressed policies are seen as a means to realizing established goals and that compliance with governance is required.

4.4.16 Manageability Capability

- Relationships
    - Manageability Capability **composes** Policy Manageability
    - Manageability Capability **composes** Service Lifecycle Manageability
    - Manageability Capability **composes** Service Configuration Manageability
    - Manageability Capability **composes** Service Combination Manageability
    - Manageability Capability **composes** Event Monitoring Manageability
    - Manageability Capability **composes** Performance Manageability
    - Manageability Capability **composes** Manageability of Quality of Service
- Notes
    - A capability that allows a resource to be controlled, monitored, and reported on with respect to some properties.
    - Manageability property: A property used in the manageability of a resource. The fundamental unit of management in systems management.
    - Each resource may be managed through a number of aspects of management, and the resources may be grouped based on similar aspects.

4.4.17 Manageability of Quality of Service

- Notes
    - Manageability of Quality of Service deals with management of service non-functional characteristics that may be of significant value to the service consumers and other stakeholders in the SOA ecosystem.
    - Manageability of quality of service assumes that the properties associated with service qualities are monitored during the service execution. Results of monitoring may be compared against an SLA or a KPI, which results in the continuous validation of how the service contract is preserved by the service provider.

4.4.18 Manageable Resource

4.4.19 Manageable Service

- Relationships
    - Manageable Service **declares** Managed Contract
    - Manageable Service **uses** Manageable Resource
    - Manageable Service **exposes** Manageability Capability
    - Management **carries out** Governane
    - Management **uses** Monitoring and Reporting

4.4.20 Managed Contract

- Relationships
  - Managed Contract **aggregates** Management Policy
- Notes
  - A service may have different management contracts for different consumers

4.4.21 Management

- Relationships
  - Management **interprets** Rule
  - Management **implements** Regulation
  - Management **generates** Regulation
  - Management **initiates actions that result in** Enforcement
  - Management **manages** Infrastructure
  - Management **aggregates** Security Management
  - Management **aggregates** Usage Management
  - Management **aggregates** Management Policy
  - Management **aggregates** Network Management
  - Management **aggregates executes** Policy
  - Management **aggregates manages** Service Lifecycle
  - Management **aggregates applies to** Service
  - Management **applies to** Service Consumer
  - Management **applies to** Service Provider
  - Management **applies to** Service Description
  - Management **applies to** Service Contract
- Notes
  - Management generates regulations that specify details for rules and other procedures to implement both rules and regulations.
  - While governance may primarily focus on setting policies, management will focus on the realization and enforcement of policies.
  - [...] the mechanisms for providing feedback from management into governance must exist.
  - Management to operationalize governance controls the life-cycle of the governing policies, including procedures and processes, for modifying the Governance Framework and Processes.
  - Management of the service contracts is based on management policies that may be mentioned in the service description and in the service contracts. Management of the service contracts is mandatory for consumer relationship management.
  - Management of the service interfaces is based on several management policies that regulate
    - availability of interfaces specified in the service contracts,
    - accessibility of interfaces,
    - procedures for interface changes,

        - interface versions as well as the versions of all parts of the interfaces,
        - traceability of the interfaces and their versions back to the service description document.
    - Management of the SLA is integral to the management of service monitoring and operational service behavior at run-time.
    - Management of the SLA includes, among others, policies to change, update, and replace the SLA. This aspect concerns service Execution Context because the business logic associated with a defined interface may differ in different Execution Contexts and affect the overall performance of the service.
    - Every resource of the SOA ecosystem and, particularly, services MUST provide manageability properties:
        - The set of manageability properties SHOULD include as minimum such properties as life-cycle, combination, configuration, event monitoring, performance, quality of services, and policy manageability;
        - Combinations of manageability properties MAY be used in different management methods and tools;

4.4.22 Management Policy

- Relationships
    - Management Policy **defines** Manageability Capability
- Notes
    - The management of resources within the SOA ecosystem may be governed by management policies. In a deployed SOA-based solution, it may well be that different aspects of the management of a given service are managed by different management services.
    - Management policies, in essence, are the realization of governing rules and regulations. As such, some management policies may target services while other policies may target the management of the services.
    - Manageability properties and applicable policies SHOULD be appropriately described in the service's description and contracts;

4.4.23 Monitoring and Reporting

4.4.24 Network Management

- Notes
    - Network management deals with the maintenance and administration of large scale physical networks such as computer networks and telecommunication networks. Specifics of the networks may affect service interactions from performance and operational perspectives.
    - Network and related system management execute a set of functions required for controlling, planning, deploying, coordinating, and monitoring the distributed services in the SOA ecosystem.

4.4.25 Other Governance

- Relationships
  - Other Governance **extends** Governance
  - Other Governance **coordinate between** IT Governance
  - Other Governance **coordinate between** SOA Governance

4.4.26 Participant Interaction Governance

- Notes
  - 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 correspond to the real world effects as contained in the service description. Governance would ensure preconditions for service use are satisfied, in particular those related to security aspects such as user authentication, authorization, and non-repudiation. If conflicts arise, governance would specify resolution processes to ensure appropriate agreements, policies, and conditions are met.
  - 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. Here, governance focuses more on 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 probably more business-specific policies that will be governed by processes outside SOA governance.

4.4.27 Performance Manageability

- Relationships
  - Performance Manageability **uses** Manageability of Quality of Service
  - Performance Manageability **evaluated against** 1..* Key Performance Indicator (KPI)
  - Performance Manageability **evaluated against** 1..* Service Level Agreement (SLA)
- Notes
  - Performance Manageability of a service allows controlling the service results, shared and sharable real world effects against the business goals and objectives of the service. This manageability assumes monitoring of the business performance as well as the management of this monitoring itself. Performance Manageability includes business and technical performance manageability through a performance criteria set, such as business key performance indicators (KPI) and service-level agreements (SLA).
  - The performance business- and technical-level characteristics of the service should be known from the service contract. The service provider and consumer must be able to monitor and measure these characteristics or be informed about the results measured by

a third party.
    - Performance Manageability is the instrument for providing compliance of the service with its service contracts. Performance Manageability utilizes Manageability of Quality of Service.

4.4.28 Policy Manageability

- Notes
    - Policy Manageability allows additions, changes and replacements of the policies associated with a resource in the SOA ecosystem. The ability to manage those policies (such as promulgating policies, retiring policies and ensuring that policy decision points and enforcement points are current) enables the ecosystem to apply policies and evaluate the results.
    - The ability to manage, i.e. use a particular manageability, requires policies from governance to be translated into detailed rules and regulations which are measured and monitored providing corresponding feedback for enforcement. At the same time, the execution of a management capability must adhere to certain policies governing the management itself.

4.4.29 Regulation

- Relationships
    - Regulation **derives from** Rule
- Notes
    - A mandated process or the specific details that derive from the interpretation of rules and lead to measureable quantities against which compliance can be measured.
    - 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 that may require resource commitments or other levies on the participants.
    - Rules and regulations [...] do not require individual acceptance by any given participant although some level of community comment may be part of the governance processes. Having agreed to governance, the participants are bound to comply or be subject to prescribed mechanisms for enforcement.
    - 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 is no means for management to assess compliance.

4.4.30 Role

4.4.31 Rule

- Relationships
    - Rule **make operational** Policy
- Notes

    - A prescribed guide for carrying out activities and processes leading to desired results, e.g. the operational realization of policies.
    - [...] policy is a predicate to be satisfied and rules prescribe the activities by which that satisfying occurs. A number of rules may be required to satisfy a given policy; the carrying out of a rule may contribute to several policies being realized.

4.4.32 Security Management

- Notes
    - Security Management includes identification of roles, permissions, access rights, and policy attributes defining security boundaries and events that may trigger a security response.

4.4.33 Service Combination Manageability

- Relationships
    - Service Combination Manageability **uses** Service Configuration Manageability
- Notes
    - Combination Manageability of a service addresses management of service characteristics that allow for creating and changing combinations in which the service participates or that the service combines itself. Known models of such combinations are aggregations and compositions. Examples of patterns of combinations are choreography and orchestration. In cases of business collaboration, combination of services appears as cooperation of services. Combination Manageability drives implementation of the Service Composability Principle of service orientation.

4.4.34 Service Configuration Manageability

- Notes
    - Configuration Manageability of a service allows managing the identity of and the interactions among internal elements of the service, for example, a use of data encryption for internal inter-component communication in particular deployment conditions. Also, Configuration Manageability correlates with the management of service versions and configuration of the deployment of new services into the ecosystem. Configuration Management differs from the Combination Manageability in the scope and scale of manageability, and addresses lower level concerns than the architectural combination of services.

4.4.35 Service Contract

- Relationships
    - Service Contract **used as** Explicit Service Contract
    - Service Contract **used as** Implicit Service Contract
- Notes
    - From the management perspective, three basic types of the contractual information relate to:
        - relationship between service provider and consumer;

        - communication with the service;
        - control of the quality of the service execution.
    - Definition: An implicit or explicit documented agreement between the service consumer and service provider about the use of the service based on
        - the commitment by a service provider to provide service functionality and results consistent with identified real world effects and
        - the commitment by a service consumer to interact with the service per specific means and per specified policies,

      where both consumer and provider actions are in the manner described in the service description.
    - The service description provides the basis for the service contract and, in some situations, may be used as an implicit default service contract. In addition, the service description may set mandatory aspects of a service contract, e.g. for security services, or may specify acceptable alternatives.

4.4.36 Service Inventory Governance

- Notes
    - 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 inherit their characteristics from experiences with distributed computing but also evolve with SOA experience. A major need 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.
    - For SOA governance, the issue is less which services are approved but rather ensuring that sufficient description is available to support informed decisions for appropriate use. Thus, SOA governance should concentrate on 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.

4.4.37 Service Level Agreement (SLA)

- Notes
    - Control of the quality of the service execution, often represented as a service level agreement (SLA), is performed by service monitoring systems and includes both technical and operational business controls. SLA is a part of the service contract and, because of the individual nature of such contracts, may vary from one service contract to another, even for the same consumer. Typically, a particular SLA in the service contract is a concrete instance of the SLA declared in the service description.
    - Management of the SLA is integral to the management of service monitoring and operational service behavior at run-time.
    - An SLA usually enumerates service characteristics and expected performances of the

service. Since an SLA carries the connotation of a ‘promise’, monitoring is needed to know if the promise is being kept. Existence of an SLA itself does not guarantee that the consumer will be provided with the service level specified in the service contract.
    - The use of an SLA in a SOA ecosystem can be wider than just an agreement on technical performances. An SLA may contain remedies for situations where the promised service cannot be maintained, or the real world effect cannot be achieved due to developments subsequent to the agreement. A service consumer that acts accordingly to realize the real world effect may be compensated for the breach of the SLA if the effect is not realized.
    - Service behavioral characteristics and performances specified in the SLA depend on the interface type and its Execution Context.

4.4.38 Service Lifecycle

- Relationships
    - Service Lifecycle **regulates** Service

4.4.39 Service Lifecycle Manageability

- Notes
    - Life-cycle Manageability of a service typically refers to how the service is created, how it is retired and how service versions must be managed.
    - Related properties may include the necessary state of the ecosystem for the creation and retirement of the service and the state of the ecosystem following the retirement of the service. The SOA ecosystem distinguishes between service composition and service aggregation: retiring of service composition leads to retiring of all services comprising the composition while retiring of service aggregation assumes that comprising services have their own life-cycle and can be used in another aggregation.

4.4.40 Service Management

- Relationships
    - Service Management **uses** Manageability Capability

4.4.41 SOA Governance

- Relationships
    - IT Governance **coordinate between** SOA Governance
    - SOA Governance **coordinate between** Other Governance
    - SOA Governance **aggregates** SOA Infrastructure Governance
    - SOA Governance **aggregates** Service Inventory Governance
    - SOA Governance **aggregates** Participant Interaction Governance
- Notes
    - There are obvious dependencies and a need for coordination between [SOA and IT Governance], but the idea of aligning IT with business already demonstrates that resource providers and resource consumers must be working towards common goals if they are to be productive and efficient. While SOA governance is shown to be active in the area of infrastructure, it is a specialized concern for having a dependable platform to

support service interaction […]
    - Governance in the context of SOA is that organization of services: that promotes their visibility; that facilitates interaction among service participants; and that directs 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.
    - SOA governance must account for interactions across ownership boundaries, which may also imply across enterprise governance boundaries.
    - [...] 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.
    - [...] SOA governance requires establishing confidence and trust (see Section 3.2.5.1) 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

4.4.42 SOA Infrastructure Governance

- Notes
    - 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. The provisioning of an infrastructure on which these services may be accessed and the general realm of those contributing as utility functions of the infrastructure are a traditional IT governance concern. In contrast, the focus of SOA governance is how the existence and use of the services enables the SOA ecosystem.

4.4.43 Test Case Specification

- Notes
    - Test conditions and expected responses are detailed in the test case specification. The test conditions should be designed to cover the areas for which the entity's response must be documented and may include:

        - nominal conditions;
        - boundaries and extremes of expected conditions;
        - breaking point where the entity has degraded below a certain level or has otherwise ceased effective functioning;
        - random conditions to investigate unidentified dependencies among combinations of conditions
        - error conditions to test error handling.

4.4.44 Usage Management

- Notes
    - Usage Management is concerned with how resources are used, including:
        - how the resource is accessed, who is using the resource, and the state of the resource (access properties);
        - controlling or shaping demand for resources to optimize the overall operation of the ecosystem (demand properties);
        - assigning costs to the use of resources and distributing those cost assignments to the participants in an appropriate manner (financial properties).

FW.5 Service oriented architecture Modeling Language (SoaML) Specification [5]

5.1 Agent

- Relationships
  - Agent **extends** Participant
  - Agent is a new stereotype in SoaML extending UML2 Component with new capabilities.
- Structure
  - Stereotype
- Constraints
  - isActive must always be true
- Notes
  - An Agent is a classification of autonomous entities that can adapt to and interact with their environment. It describes a set of agent instances that have features, constraints, and semantics in common. Agents in SoaML are also participants, providing and using services.
  - Each aspect of an agent may be expressed as a services architecture.
  - Aspects: Agent, Collaboration, Role, Interaction, Behavioral, Organization/Group aspect
  - Agent extends Participant with the ability to be active, participating components of a system. They are specialized because they have their own thread of control or lifecycle. Another way to think of agents is that they are “active participants” in a SOA system. Participants are Components whose capabilities and needs are static. In contrast, Agents are Participants whose needs and capabilities may change over time.
  - In SoaML, Agent is a Participant (a subclass of Component). A Participant represents some concrete Component that provides and/or consumes services and is considered an active class (isActive=true). However, SoaML restricts the Participant’s classifier behavior to that of a constructor, not something that is intended to be long-running, or represent an “active” lifecycle. This is typical of most Web Services implementations as reflected in WS-* and SCA.
  - Agents possess the capability to have services and Requests and can have internal structure and ports. They collaborate and interact with their environment. An Agent’s classifierBehavior, if any, is treated as its life-cycle, or what defines its emergent or adaptive behavior.

5.2 Attachment

- Relationships
  - Attachment **extends** metaclass Property
- Structure
  - Stereotype
  - encoding: String [0..1]: Denotes the platform encoding mechanism to use in generating the schema for the message; examples might be SOAP-RPC, Doc-Literal, ASN.1, etc.
  - mimeType: String [0..1]: Denotes the iana MIME media type for the Attachment

- Notes
  - Attachments use the usual UML2 notation for DataType with the addition of an “Attachment” stereotype.
  - An Attachment extends Property to distinguish attachments owned by a MessageType from other ownedAttributes. The ownedAttributes of a MessageType must be either PrimitiveType or MessageType.
  - Extends UML2 to distinguish message attachments from other message proprieties.
  - A part of a Message that is attached to rather than contained in the message.
  - An Attachment denotes some component of a message that is an attachment to it (as opposed to a direct part of the message itself). In general this is not likely to be used greatly in higher level design activities, but for many processes attached data is important to differentiate from embedded message data.
  - Attachments may be used to indicate part of service data that can be separately accessed, reducing the data sent between consumers and providers unless it is needed.

5.3 Behavior

- Notes
  - The Behavior specifies the valid interactions between the provider and consumer – the communication protocol of the interaction, constraining but without specifying how either party implements their role. Any UML behavior specification can be used, but interaction and activity diagrams are the most common.

5.4 Capability

- Relationships
  - Capability **extends** metaclass Class
- Structure
  - Stereotype
- Notes
  - Additions to UML2: Capability is a new stereotype used to describe service capabilities.
  - Participants that provide a service must have a capability to provide it, but different providers may have different capabilities to provide the same service - some may even “outsource” or delegate the service implementation through a request for services from others. The capability behind the service will provide the service according to a service description and may also have dependencies on other services to provide that capability. The service capability is frequently integral to the provider’s business process. Capabilities can be seen from two perspectives, capabilities that a participant has that can be exploited to provide services, and capabilities that an enterprise needs that can be used to identify candidate services.
  - However, when re-architecting existing applications for services or building services from scratch, even the abstract Participants may not yet be known. In these situations it is also useful to express a services architecture in terms of the logical capabilities of the services in a “Participant agnostic” way. Even though service consumers should not be

concerned with how a service is implemented, it is important to be able to specify the behavior of a service or capability that will realize or implement a ServiceInterface.
- Capabilities represent an abstraction of the ability to affect change
- Capabilities identify or specify a cohesive set of functions or resources that a service provided by one or more participants might offer. Capabilities can be used by themselves or in conjunction with Participants to represent general functionality or abilities that a Participant must have.
- […] networks of capabilities are used to identify needed services, and to organize them into catalogs in order to communicate the needs and capabilities of a service area, whether that is business or technology focused, prior to allocating those services to particular Participants.
- Capabilities can have usage dependencies on other Capabilities to show how these capabilities are related. Capabilities can also be organized into architectural layers to support separation of concern within the resulting service architecture.
- In addition to specifying abilities of Participants, one or more Capabilities can be used to specify the behavior and structure necessary to support a ServiceInterface.
- [Capabilities allow] architects to analyze how services are related and how they might be combined to form some larger capability prior to allocation to a particular Participant.
- Capabilities can, in turn be realized by a Participant. When that Capability itself realizes a ServiceInterface, that ServiceInterface will normally be the type of a Service on the Participant as shown in Figure 6.21.
- Capabilities may also be used to specify the parts of Participants, the capabilities the participant has to actually provide its services.
- ServiceInterfaces may also expose Capabilities. This is done within SoaML with the Expose Dependency. While this can be used as essentially just the inverse of a Realization between a Capability and a ServiceInterface it realizes, it can also be used to represent a means of identifying candidate services or a more general notion of “providing access” to a general capability of a Participant.
- Each Capability may have owned behaviors that are methods of its provided Operations. These methods would be used to specify how the Capabilities might be implemented, and to identify other needed Capabilities. Alternatively, ServiceInterfaces may simply expose Capabilities of a Participant.
- Capabilities, ServiceContracts, and ServicesArchitectures provide a means of bridging between business concerns and SOA solutions by tying the business requirements realized by the contracts to the services and service participants that fulfill the contracts.
- A Capability is the ability to act and produce an outcome that achieves a result. It can specify a general capability of a participant as well as the specific ability to provide a service.
- A Capability […] achieves a result that may provide a service specified by a ServiceContract or ServiceInterface irrespective of the Participant that might provide that service. A ServiceContract, alone, has no dependencies or expectation of how the capability is realized – thereby separating the concerns of “what” vs. “how.” The

Capability may specify dependencies or internal process to detail how that capability is provided including dependencies on other Capabilities. Capabilities are shown in context using a service dependencies diagram.
    - It allows architects to analyze how services are related and how they might be combined to form some larger capability prior to allocation to a particular Participant.
    - A Capability identifies or specifies a cohesive set of functions or capabilities that a service provided by one or more participants might offer. Capabilities are used to identify needed services, and to organize them into catalogs in order to communicate the needs and capabilities of a service area, whether that be business or technology focused, prior to allocating those services to particular Participants.
    - Capabilities can have usage dependencies with other Capabilities to show how these capabilities are related. Capabilities can realize ServiceInterface and so specify how those ServiceInterfaces are supported by a Participant. Capabilities can also be organized into architectural layers to support separation of concern within the resulting service architecture.
    - Each capability may have owned behaviors that are methods of its provided Operations. These methods would be used to specify how the service capabilities might be implemented, and to identify other needed service capabilities.
    - A Capability represents something a Participant needs to have or be able to do in order to support a value proposition or achieve its goals, or something a Participant has that enables it to carry out its provided services. Capabilities may be used to identify services that are needed or to describe the operations that must be provided by one or more services.
    - […] Capabilities may realize ServiceInterfaces using standard UML Realization. This approach is somewhat different in that it says that the Service Interface is a “specification” and the Capability “implements” this specification. As with the Expose dependency, the Capability is not required to have the same operations or properties as the Service Interface it realizes.

5.5 CapabilityExposure

- Relationships
    - CapabilityExposure **extends** metaclass Realization
    - * CapabilityExposure **exposed by** ServiceInterface
    - * CapabilityExposure **exposes** Capability

5.6 CapabilityRealization

- Relationships
    - CapabilityRealization **extends** metaclass Realization
    - * CapabilityRealization **is implemented by** Participant
    - * CapabilityRealization **realizes** Capability

5.7 Catalog (Categorization)

- Relationships

    - Catalog **extends** metaclass Package
    - Catalog **specializes** NodeDescriptor
- Constraints
    1. Catalogs can only contain Categories, CategoryValues, or other Catalogs.
- Notes
    - Provides a means of classifying and organizing elements by categories for any purpose.
    - A named collection of related elements, including other catalogs characterized by a specific set of categories. Applying a Category to an Element using a Categorization places that Element in the Catalog. Catalog is a RAS DescriptorGroup containing other Catalogs and/or Categories providing the mapping to RAS classification.
    - When a model Element is categorized with a Category or CategoryValue, it is effectively placed in the Catalog that contains that Category. In the case of classification by a CategoryValue, the Category is the classifier of the CategoryValue.
    - The meaning of being categorized by a Category, and therefore placed in a Catalog is not specified by this specification. It can mean whatever the modeler wishes. That meaning might be suggested by the catalog and category name, the category's attributes, and a category value's attribute values. The same model element can be categorized many ways. The same category or category value may be used to categorize many model elements.

5.8 Categorization (Categorization)

- Relationships
    - Categorization **extends** metaclass Dependency
- Constraints
    1. The target of a Categorization must be either a Category or CategoryValue.
- Notes
    - Used to categorize an Element by a Category or CategoryValue.
    - Categorization connects an Element to a Category or CategoryValue in order to categorize or classify that element. The Element then becomes a member of the Catalog that contains that Category. This allows Elements to be organized in many hierarchical Catalogs where each Catalog is described by a set of Categories.
    - The source is any Element, the target is a Category or CategoryValue.
    - The primary purpose of Category is to be able to provide information that characterizes an element by some domain of interest. Categorizing an element characterizes that element with that Category. What this means is derived from the meaning of the Category. The meaning of a Category is defined by its name, owned attributes, or constraints if any.
    - Categorization of an element may be used to provide multiple orthogonal ways of organizing elements. UML currently provides a single mechanism for organizing model elements as PackagedElements in a Package. This is useful for namespace management and any other situations where it is necessary for an element to be in one and only one container at a time. But it is insufficient for organization across many different

dimensions since a PackageableElement can only be contained in one Package. For example, model elements might also need to be organized by owner, location, cost gradient, time of production, status, portfolio, architectural layer, Web, tiers in an n-tiered application, physical boundary, service partitions, etc. Different classification hierarchies and Categories may be used to capture these concerns and be applied to elements to indicate orthogonal organizational strategies.

5.9 Category (Categorization)

- Relationships
  - Category **extends** NodeDescriptor
- Constraints
  1. A Category must be contained in a Catalog.
- Notes
  - A classification or division used to characterize the elements of a catalog and to categorize model elements.
  - A Category is a piece of information about an element. A Category has a name indicating what the information is about, and a set of attributes and constraints that characterize the Category. An Element may have many Categories, and the same Category can be applied to many Elements. Categories may be organized into Catalogs hierarchies.
  - The meaning of a Category is not specified by SoaML. Instead it may be interpreted by the modeler, viewer of the model, or any other user for any purpose they wish. For example a Catalog hierarchy of Categories could be used to indicate shared characteristics used to group species. In this case the categorization might imply inheritance and the principle of common descent. Other categorizations could represent some other taxonomy such as ownership. In this case, the term categorization is intended to mean describing the characteristics of something, not necessarily an inheritance hierarchy. All instances having categorized by a Category have the characteristics of that Category.
  - The characteristics of a Category are described by its attributes and constraints. ClassifierValues may be used to provide specific values for these attributes in order to more specifically categorize an element.
  - A Category may have ownedRules representing Constraints that further characterize the category. The meaning of these constraints when an element is categorized by a Category is not specified.

5.10 CategoryValues (Categorization)

- Relationships
  - CategoryValue **extends** FreeFormValue
- Constraints
  1. The classifier for a CategoryValue must be a Category.
- Notes
  - Provides specific values for a Category to further categorize model elements.

- A CategoryValue provides values for the attributes of a Category. It may also be used to categorize model elements providing detailed information for the category.
- The characteristics of a Category are described by its attributes and constraints. ClassifierValues may be used to provide specific values for these attributes in order to more specifically categorize an element.
- Categorizing an element with a CategoryValue categorizes the element by the Category that is the classifier of the CategoryValue.

5.11 Choreography

- Notes
  - The service choreography […] is a behavior owned by the service interface and defines the required and optional interactions between the provider and consumer.
  - […] interactions can be defined using signals, which makes [the] service interface asynchronous and document oriented, a mainstream SOA best practice.
  - The choreography defines what must go between the contract roles as defined by their service interfaces-when, and how each party is playing their role in that service without regard for who is participating.

5.12 Collaboration

- Relationships
  - Collaboration **extends** metaclass Collaboration
- Structure
  - isStrict: Boolean = true: Indicates whether this Collaboration is intended to represent a strict pattern of interaction. Establishes the default value for any CollaborationUse typed by this Collaboration.
- Notes
  - Collaboration is extended to indicate whether the role to part bindings of CollaborationUses typed by a Collaboration are strictly enforced or not.
  - A Collaboration, ServiceContract, or ServicesArchitecture represents a pattern of interaction between roles. This interaction may be informal and loosely defined as in a requirements sketch. Or it may represent formal agreements or requirements that must be fulfilled exactly. A Collaboration's isStrict property establishes the default value of the isStrict property for any CollaborationUse typed by the Collaboration.
  - Note that as a ServiceContract is binding on the ServiceInterfaces named in that contract, a CollaborationUse is not required if the types are compatible.
  - A Collaboration is a description of a pattern of interaction between roles responsible for providing operations whose use can be described by ownedBehaviors of the Collaboration. It is a description of possible structure and behavior that may be played by other parts of the model.
  - A Collaboration may have isStrict=true indicating the collaboration represents a formal interaction between its roles that all parts playing those roles are intended to follow. If isStrict=false, then the collaboration represents an informal pattern of interaction that

may be used to document the intended interaction between parts without specifically requiring parts bound to roles in CollaborationUses typed by the collaboration to be compatible.

5.13 CollaborationUse

- Relationships
  - CollaborationUse **extends** metaclass CollaborationUse
- Structure
  - isStrict: Boolean: Indicates whether this particular fulfillment is intended to be strict. A value of true indicates the roleBindings in the Fulfillment must be to compatible parts. A value of false indicates the modeler warrants the part is capable of playing the role even though the type may not be compatible. The default value is the value of the isStrict property of Collaboration used as the type of the CollaborationUse. Additions to UML2: CollaborationUse extends UML2 CollaborationUse to include the isStrict property.
- Notes
  - CollaborationUse is extended to indicate whether the role to part bindings are strictly enforced or loose.
  - A CollaborationUse explicitly indicates the ability of an owning Classifier to fulfill a ServiceContract or adhere to a ServicesArchitecture. A Classifier may contain any number of CollaborationUses that indicate what it fulfills. The CollaborationUse has roleBindings that indicate what role each part in the owning Classifier plays. If the CollaborationUse is strict, then the parts must be compatible with the roles they are bound to, and the owning Classifier must have behaviors that are behaviorally compatible with the ownedBehavior of the CollaborationUse’s Collaboration type.
  - Note that as a ServiceContract is binding on the ServiceInterfaces named in that contract, a CollaborationUse is not required if the types are compatible.
  - A CollaborationUse is a statement about the ability of a containing Classifier to provide or use capabilities, have structure, or behave in a manner consistent with that expressed in its Collaboration type. It is an assertion about the structure and behavior of the containing classifier and the suitability of its parts to play roles for a specific purpose.
  - A CollaborationUse contains roleBindings that binds each of the roles of its Collaboration to a part of the containing Classifier. If the CollaborationUse has isStrict=true, then the parts must be compatible with the roles they are bound to. For parts to be compatible with a role, one of the following must be true:
    1. The role and part have the same type.
    2. The part has a type that specializes the type of the role.
    3. The part has a type that realizes the type of the role.
    4. The part has a type that contains at least the ownedAttributes and ownedOperations of the role. In general this is a special case of item 3 where the part has an Interface type that realizes another Interface.

5.14 Consumer

- Relationships
  - Consumer **extends** metaclass Interface (in the case of a non composite service contract)
  - Consumer **extends** metaclass Class (in the case of a composite service contract)
  - […] the consumer of the service uses a «Request» port.
- Structure
  - consumer: RoleType – this is the role of the consumer in the [...] service. The type of the consumer role is the interface that a consumer will implement on a port to consume this service (note that in the case of a one-directional service, there may not be a consumer interface). This [interface] indicates all of the operations and signals a consuming participant will receive when enacting the service.
  - stereotype: <<Consumer>>
- Constraints
  - The «Consumer» is bound by the constraints and behavior of the ServiceContract of which it is a type.
- Notes
  - The interface of the consumer (if any) must be used by the [ServiceInterface] class.
  - The consumer is the participant that has some need and requests a service of a provider.
  - Consumer models the type of a service consumer. A consumer is then used as the type of a role in a service contract and the type of a port on a participant.
  - A «Consumer» models the interface provided by the consumer of a service. The consumer of the service receives the results of the service interaction. The consumer will normally be the one that initiates the service interaction. Consumer interfaces are used as the type of a «ServiceContract» and are bound by the terms and conditions of that service contract.
  - The «Consumer» is intended to be used as the port type of a participant that uses a service.
  - The consumer requests a service of the provider who then uses their capabilities to fulfill the service request and ultimately deliver value to the consumer. The interaction between the provider and consumer is governed by a «ServiceContract» where both parties are (directly or indirectly) bound by that contract.
  - The consumer interface and therefore the consumer role combine to fully define a service from the perspective of the consumer.
  - The consumer interface represents the operations and signals (if any) that the consumer will receive during the service interaction.
  - The Consumer will also have a uses dependency on the provider interface, representing the fact that the consumer must call the provider.

5.15 Expose

- Relationships
  - Expose **extends** metaclass Dependency
- Structure

    - Stereotype
- Notes
    - Additions to UML2: Extends Dependency to formalize the notion that a Capability can be exposed by a ServiceInterface.
    - An Expose dependency is used to indicate a Capability exposed through a ServiceInterface. The source of the Expose is the ServiceInterface, the target is the exposed Capability.
    - The Expose dependency provides the ability to indicate what Capabilities that are required by or are provided by a participant should be exposed through a Service Interface.
    - An Expose dependency is a relationship between a Service Interface and a Capability it exposes or that provides it. This means that the exposing Service Interface provides operations and information consistent with the capabilities it exposes. This is not the same as realization. The Service Interface is not required to have the same operations and properties as the capabilities it exposes. It is possible that services supported by capabilities could be refactored to address commonality and variability across a number of exposed capabilities, or to address other SOA concerns not accounted for in the capabilities.

5.16 MessageType

- Relationships
    - MessageType **extends** metaclass DataType
    - MessageType **extends** metaclass Class
    - MessageType **extends** metaclass Signal
- Structure
    - encoding: String [0..1]: Specifies the encoding of the message payload. Additions to UML2: Formalizes the notion of object used to represent pure data and message content packaging in UML2 recognizing the importance of distribution in the analysis and design of solutions.
- Constraints
    - MessageType cannot contain ownedOperations.
    - MessageType cannot contain ownedBehaviors.
    - All ownedAttributes must be Public.
- Notes
    - The specification of information exchanged between service consumers and providers.
    - A MessageType is a kind of value object that represents information exchanged between participant requests and services. This information consists of data passed into, and/or returned from, the invocation of an operation or event signal defined in a service interface. A MessageType is in the domain or service-specific content and does not include header or other implementation or protocol-specific information.
    - MessageTypes are used to aggregate inputs, outputs, and exceptions to service operations as in WSDL. MessageTypes represent “pure data” that may be communicated

between parties. It is then up to the parties, based on the SOA specification, to interpret this data and act accordingly.
    - The terms Data Transfer Object (DTO), Service Data Object (SDO), or value objects used in some technologies are similar in concept, though they tend to imply certain implementation techniques. A DTO represents data that can be freely exchanged between address spaces, and does not rely on specific location information to relate parts of the data. An SDO is a standard implementation of a DTO. A Value Object is a Class without identity and where equality is defined by value not reference. Also in the business world (or areas of business where EDI is commonplace) the term Document is frequently used. All these concepts can be represented by a MessageType.
    - […] SoaML allows MessageType to be applied to Class as well as DataType. In this case, the identity implied by the Class is not considered in the MessageType. The Class is treated as if it were a DataType.
    - MessageTypes represent service data exchanged between service consumers and providers. Service data is often a view (projections and selections) on information or domain class models representing the (often persistent) entity data used to implement service participants. MessageType encapsulates the inputs, outputs, and exceptions of service operations into a type based on direction. A MessageType may contain attributes with isID set to true indicating the MessageType contains information that can be used to distinguish instances of the message payload. This information may be used to correlate long running conversations between service consumers and providers.
    - Where message types contain classes as attributes or aggregated associations the message type will contain a “copy by value” of the public state of the those objects. Where those objects contain references to other objects those references will likewise be converted to value data types.

5.17 Milestone

- Relationships
    - Milestone **extends** metaclass Comment
    - signal: Signal [0..1]: A Signal associated with this Milestone.
    - value: Expression [*]: Arguments of the signal when the Milestone is reached.
- Structure
    - progress: Integer: The progress measurement. Additions to UML2: Distinguishes that this is a concept that adds nothing to the functional semantics of the behavior, and may as such be ignored by implementations.
- Notes
    - A Milestone is a means for depicting progress in behaviors in order to analyze liveness. Milestones are particularly useful for behaviors that are long lasting or even infinite.
    - A Milestone depicts progress by defining a signal that is sent to an abstract observer. The signal contains an integer value that intuitively represents the amount of progress that has been achieved when passing a point attached to this Milestone.
    - Provided that a SoaML specification is available it is possible to analyze a service

behavior (a Participant or a ServiceContract) to determine properties of the progress value.
    - A Milestone can be understood as a “mythical” Signal. A mythical Signal is a conceptual signal that is sent from the behavior every time a point connected to the Milestone is passed during execution. The signal is sent to a conceptual observer outside the system that is able to record the origin of the signal, the signal itself, and its progress value.
    - The signal is mythical in the sense that the sending of such signals may be omitted in implemented systems as they do not contribute to the functionality of the behavior. They may, however, be implemented if there is a need for run-time monitoring of the progress of the behavior.

5.18 MotivationElement (BMM Integration)

- Relationships
    - MotivationElement **extends** Extensions
- Notes
    - Any UML BehavioredClassifier (including a ServicesContract for example) may realize a BMM MotivationElement through a MotivationRealization. This allows services models to be connected to the business motivation and strategy linking the services to the things that make them business relevant.
    - A placeholder for BMM MotivationElement. This placeholder would be replaced by a BMM profile or metamodel element.

5.19 MotivationRealization (BMM Integration)

- Relationships
    - MotivationRealization **extends** Realization
- Structure
    - realizedEnd: End [*]: The ends realized by this MeansRealization. (Metamodel only)
- Notes
    - Models a realization of a BMM MotivationElement (a Vision, Goal, Objective, Mission, Strategy, Tactic, BusinessPolicy, Regulation, etc.) by some BehavioredClassifier.

5.20 Operation

- Notes
    - A service Operation is any Operation of an Interface provided or required by a Service or Request. Service Operations may use two different parameter styles, document centered (or message centered) or RPC (Remote Procedure Call) centered. Document centered parameter style uses MessageType for ownedParameter types, and the Operation can have at most one in, one out, and one exception parameter (an out parameter with isException set to true). All parameters of such an operation must be typed by a MessageType. For RPC style operations, a service Operation may have any number of in, inout, and out parameters, and may have a return parameter as in UML2. In this case, the parameter types are restricted to PrimitiveType or DataType. This ensures no service Operation makes any assumptions about the identity or location of

any of its parameters. All service Operations use call-by-value semantics in which the ownedParameters are value objects or data transfer objects.

5.21 Participant

- Relationships
  - Participant **extends** metaclass Component
  - Participant **realizes** * CapabilityRealization
- Structure
  - Additions to UML2: Participant is a stereotype of UML2 Class or component with the ability to have services and requests.
- Constraints
  1. A Participant cannot realize or use Interfaces directly; it must do so through service ports, which may be Service or Request.
  2. Note that the technology implementation of a component implementing a participant is not bound by the above rule in the case of it's internal technology implementation, the connections to a participant components “container” and other implementation components may or may not use services.
- Notes
  - Participants are either specific entities or kinds of entities that provide or use services. Participants can represent people, organizations, or information system components. Participants may provide any number of services and may consume any number of services.
  - Services are provided by participants who are responsible for implementing the services, and possibly using other services. Services implementations may be specified by methods that are behaviors of the participants expressed using interactions, activities, state machines, or opaque behaviors. Participants may also delegate service implementations to parts in their internal structure which represent an assembly of other service participants connected together to provide a complete solution, perhaps specified by, and adhering to a services architecture.
  - As mentioned earlier, provider and consumer entities may be people, organizations, technology components or systems -we call these Participants. Participants offer services through Ports that may use the “Service” stereotype and request services on Ports with the “Request” stereotype. These ports represent features of the participants where the service is offered or consumed.
  - Participants represent software components, organizations, systems, or individuals that provide and use services.
  - Participants define types of organizations, organizational roles, or components by the roles they play in services architectures and the services they provide and use.
  - Participants provide capabilities through Service ports typed by ServiceInterfaces or in simple cases, UML Interfaces that define their provided or offered capabilities.
  - [...] it is common for a participant to be the provider and consumer of many services.
  - In contrast [to a role], a Participant specifies the type of a party that fills the role in the

context of a specific services architecture.

- A participant may also have an architecture – one that specifies how parts of that participant (e.g., departments within an organization) work together to provide the services of the owning participant. The architecture of a participant is shown as a Services Architecture as well, perhaps realized by a UML class or component and frequently has an associated business process. Participants that realize this specification must adhere to the architecture it specifies.
- [...] Participants are classifiers defined both by the roles they play in services architectures (the participant role) and the “contract” requirements of entities playing those roles. Each participant type may “play a role” in any number of services architecture, as well as fulfill the requirements of each. Requirements are satisfied by the participant having service ports that have a type compatible with the services they must provide and consume.
- Participants can engage in a variety of contracts. What connects participants to particular service contract is the use of a role in the context of a ServicesArchitecture. Each time a ServiceContract is used in a ServicesArchitecture; there must also be a compliant port on a participant - possibly designated as a Service or Request. This is where the participant actually offers or uses the service.
- A Participant may have parts typed by Capabilities that indicate what the participant is able to do. Such a Participant may also Realize a set of Capabilities through its ownedBehaviors or through delegation to parts in its internal structure, or through delegation to requests for services from others. These capabilities may also be exposed by ServiceInterfaces used to specify the type of service ports through which the capabilities are accessed.
- A participant is the type of a provider and/or consumer of services. In the business domain a participant may be a person, organization, or system. In the systems domain a participant may be a system, application, or component.
- A Participant represents some (possibly concrete) party or component that provides and/or consumes services (participants may represent people, organizations, or systems that provide and/or use services). A Participant is a service provider if it offers a service. A Participant is a service consumer if it uses a service. A participant may provide or consume any number of services. Service consumer and provider are roles Participants play: the role of providers in some services and consumers in others, depending on the capabilities they provide and the needs they have to carry out their capabilities. Since most consumers and providers have both services and requests, Participant is used to model both.
- Participants have ports. These ports may use the “Service” and “Request” stereotypes that are the interaction points where services are offered or consumed respectively. Internally a participant may specify a behavior, a business process, or a more granular service contract as a Participant Architecture.
- A concrete Participant may participate in and/or adhere to any number of services architectures. A composite structure is generally used to define the concrete sub-

components of the participant.

- The full scope of a SOA is realized when the relationship between participants is described using a services architecture. A services architecture shows how participants work together for a purpose, providing, and using services.
- It represents a component that (if not a specification or abstract) can be instantiated in some execution environment or organization and connected to other participants through ServiceChannels in order to provide its services. Participants may be organizations or individuals (at the business level) or system components or agents (at the I.T. level).
- A Participant implements each of its provided service operations. Provided services may be implemented either through delegation to its parts representing capabilities or resources required to perform the service or other participants having the required capabilities and resources, through requests for services from others, through the methods of the service operations provided by its owned behaviors, or through actions that respond to received events. The implementation of the service must be consistent with the operations, protocols, and constraints specified by the ServiceInterface.
- UML2 provides three possible ways a Participant may implement a service operation:
  1. Method: A provided service operation may be the specification of an ownedBehavior of the Participant. The ownedBehavior is the method of the operation. When the operation is invoked by some other participant through a ServiceChannel connecting its Request to this Participant's Service, the method is invoked and runs in order to provide the service. Any Behavior may be used as the method of an Operation including Interaction, Activity, StateMachine, ProtocolStateMachine, or OpaqueBehavior.
  2. Event Handling: A Participant may have already running ownedBehaviors These behaviors may have forked threads of control and may contain AcceptEventAction or AcceptCallAction. An AcceptEventAction allows the Participant to respond to a triggered SignalEvent. An AcceptCallAction allows the Participant to respond to a CallEvent. This allows Participants to control when they are willing to respond to an event or service request. Contrast with the method approach above for implementing a service operation where the consumer determines when the method will be invoked.
  3. Delegation: A Participant may delegate a service to a service provided by one of its parts, or to a user. A part of a participant may also delegate a Request to a Request of the containing participant. This allows participants to be composed of other participants or components, and control what services and Requests are exposed. Delegation is the pattern often used for legacy wrapping in services implementations.
- A Participant may play a role in any number of services architecture thereby representing the role a participant plays and the requirements that each role places on the participant.

5.22 Port

- Relationships

    - Port **extends** metaclass Port
- Structure
    - Stereotype
    - connectorRequired: Boolean [0..1] = true: Indicates whether a connector is required on this Port or not. The default value is true.
- Notes
    - A port typed by a ServiceInterface and designated with the «Service» stereotype is known as a service port. A service port is the feature that represents the point of interaction on a Participant where a service is actually provided. In other words, the service port is the point of interaction for engaging participants in a service via its service interfaces.
    - A request is defined using a port stereotyped as a «Request» port. Just as we want to define the services provided by a participant using a service port, we want to define what services a participant needs or consumes. A Participant expresses their needs by making a request for services from some other Participant. […] When «Request» is used the type name will be preceded with a tilde (“~”) to show that the conjugate type is being used.
    - Participants provide or consume services via ports. A port is the part or feature of a participant that is the interaction point for a service - where it is provided or consumed. A port where a service is offered may be designated as a “Service” port and the port where a service is consumed may be designated as a “Request” port.
    - The service port has a type that describes how to use that service. That type may be either a UML Interface (for very In either case the type of the service port specifies, directly or indirectly, simple services) or a ServiceInterface. In either case the type of the service port specifies, directly or indirectly, everything that is needed to interact with that service - it is the contract between the providers and users of that service.
    - The UML interface may be used as the type of «Service» ports and «Request» ports.
    - The contract of interaction required and provided by a Service port or Request port (see below) are defined by their type which is a ServiceInterface, or in simple cases, a UML2 Interface.
    - The focus of an interface based SOA is the specification of ServiceInterfaces provided or used by a participant’s ports. The ServiceInterface specification of these ports fully specifies the participant’s requirements to provide the service on a «Service» or to request a service on a «Request». The ports are then compatible if the service interfaces are compatible.
    - A service uses the UML concept of a Port and indicates the feature or interaction point through which a classifier interacts with other classifiers (see Figure 6.6).
    - Extends UML Port with a means to indicate whether a Connection is required on this Port or not.
    - The property connectorRequired set to true on a Port indicates the Port must be connected to at least one Connector. This is used to indicate a Service port that must be used, or a Request port that must be satisfied. A Port with connectorRequired set to false indicates that no connection is required; the containing Component can function without

interacting with another Component through that Port.
    - All [...] ports are either service or request ports, but isService [from UML metaclass] is intended to distinguish those that would be involved in a client-facing value chain, and not something that is about the implementation of the participant or something provided for the detailed implementation of some other participant.

5.23 Property

- Relationships
    - Property **extends** metaclass Property
- Structure
    - isID: Boolean [0..1] = false: Indicates the property contributes to the identification of instances of the containing classifier. Additions to UML2: Adds the isID property from MOF2 in order to facilitate identification of classifier instances in a distributed environment.
- Notes
    - The Property stereotype augments the standard UML Property with the ability to be distinguished as an identifying property meaning the property can be used to distinguish instances of the containing Classifier. This is also known as a “primary key.” In the context of SoaML the ID is used to distinguish the correlation identifier in a message.
    - A property is a structural feature. It relates an instance of the class to a value or collection of values of the type of the feature. A property may be designated as an identifier property, a property that can be used to distinguish or identify instances of the containing classifier in distributed systems.
    - In distributed environments, identity is much more difficult to manage in an automated, predictable, efficient way. The same issue occurs when an instance of a Classifier must be persisted as some data source such as a table in a relational database. The identity of the Classifier must be maintained in the data source and restored when the instance is reactivated in some execution environment. The instance must be able to maintain its identity regardless of the execution environment in which it is activated. This identity is often used to maintain relationships between instances, and to identify targets for operation invocations and events.
    - Ideally modelers would not be concerned with instance identity and persistence and distribution would be transparent at the level of abstraction supporting services modeling. Service models can certainly be created ignoring these concerns. However, persistence and distribution can have a significant impact on security, availability and performance making them concerns that often affect the design of a services architecture.
    - Often important business data can be used for identification purposes such as a social security number or purchase order id. By carrying identification information in properties, it is possible to freely exchange instances into and out of persistent stores and between services in an execution environment.
    - A Classifier can have multiple properties with isID set to true with the set of such

properties capable of identifying instance of the classifier. Compound identifiers can be created by using a Class or DataType to define a property with isID=true.

5.24 Provider

- Relationships
  - Provider **extends** metaclass Interface (in the case of a non composite service contract)
  - Provider **extends** metaclass Class (in the case of a composite service contract)
- Structure
  - provider : RoleType – this defines the role of the provider in the [...] service. RoleType is the interface that a provider will require on a port to provide this service. This [interface] indicates all the operations and signals a providing participant may receive when enacting this service.
  - The provider of the service uses a «Service» port […]
  - stereotype: <<Provider>>
- Constraints
  - The “Provider” interface is bound by the constraints and behavior of the ServiceContract of which it is a type.
- Notes
  - The interface of the provider (which must be the type of one of the parts in the class) is realized (provided) by the ServiceInterface class. […] that makes [the] service interface provider centric.
  - The provider is the participant that provides something of value to the consumer.
  - [A service port on] a service provider [...] can be thought of as the “offer” of the service (based on the service interface).
  - Provider models the type of a service provider in a consumer/provider relationship. A provider is then used as the type of a role in a service contract and the type of a port on a participant.
  - A “Provider” models the interface provided by the provider of a service. The provider of the service delivers the results of the service interaction. The provider will normally be the one that responds to the service interaction. Provider interfaces are used in as the type of a “ServiceContract” and are bound by the terms and conditions of that service contract.
  - The “Provider” interface is intended to be used as the port type of a participant that provides a service.
  - The provider interface and therefore the provider role combine to fully define a service from the perspective of the provider.
  - The provider interface represents the operations and signals that the provider will receive during the service interaction.
  - The Provider may also have a uses dependency on the consumer interface, representing the fact that the provider may call the consumer as part of a bi-directional interaction. These are also known as “callbacks” in many technologies.

5.25 Request

- Relationships
  - Request **extends** metaclass Port
  - * Request **is associated with protocol** 0..1 ServiceInterface
- Constraints
  1. The type of a Request must be a ServiceInterface or an Interface.
  2. The isConjugated property of a “Request” must be set to true.
- Notes
  - A Request represents a feature of a Participant that is the consumption of a service by one participant provided by others using well-defined terms, conditions and interfaces. A Request designates ports that define the connection point through which a Participant meets its needs through the consumption of services provided by others.
  - A request port is a “conjugate” port. This means that the provided and required interfaces of the port type are inverted; this creates a port that uses the port type rather than implementing the port type.
  - A Request extends Port to specify a feature of a Participant that represents a service the Participant needs and consumes from other participants. The request is defined by a ServiceInterface. It is used by the Participant either through delegation from its parts or through actions in its methods. The request may be connected to a business MotivationalElement to indicate the intended goals the Participant wishes to achieve. There may be constraints associated with the request that define its nonfunctional characteristics or expected qualities of service. This information may be used by potential providers to determine if their service meets the participant’s needs.
  - A Request may include the specification of the value required from another, and the request to obtain value from another. A Request is the visible point at which consumer requests are connected to service providers, and through which they interact in order to produce some real world effect.
  - A Request is distinguished from a simple used Operation in that it may involve a conversation between the parties as specified by some communication protocol that is necessary to meet the needs.
  - Request extends UML2 Port and changes how provided and required interfaces are interpreted by setting the ports isConjugated property to true. The capabilities consumed through the Request - its required interfaces - are derived from the interfaces realized by the service’s ServiceInterface. The capabilities provided by a consumer in order to use the service - its provided interfaces - are derived from the interfaces used by the service’s ServiceInterface. These are the opposite of the provided and required interfaces of a Port or Service and indicate the use of a Service rather than the provision of a service. Since the provided and required interfaces are reversed, a request is the use of the service interface - or logically the conjugate type of the provider.
  - Distinguishing requests and services allows the same ServiceInterface to be used to type both the consumer and provider ports. Any Request can connect to any Service as long as their types are compatible. Request and Service effectively give Ports a direction

indicating whether the capabilities defined by a ServiceInterface are used or provided.
    - A Request is typed by an Interface or ServiceInterface which completely characterizes specific needs of the owning Participant. This includes required interfaces which designate the needs of the Participant through this Request, and the provided interfaces which represent what the Participant is willing and able to do in order to use the required capabilities. It also includes any protocols the consuming Participant is able to follow in the use of the capabilities through the Request.
    - If the type of a “Request” is a ServiceInterface, then the Request’s provided Interfaces are the Interfaces used by the ServiceInterface while its required Interfaces are those realized by the ServiceInterface. If the type of a “Request” is a simple Interface, then the required interface is that Interface and the provided interfaces are those interfaces used be the simple interface, in any.

5.26 Role

- Structure
    - roles have a name
    - roles have an interface type
- Notes
    - A role defines the basic function (or set of functions) that an entity may perform in a particular context.

5.27 Service

- Relationships
    - Service **extends** metaclass Port
    - * Service **is associated with protocol** 0..1 ServiceInterface
- Structure
    - Additions to UML2: None - Service is a stereotype of UML Port to designate a feature of a Participant.
- Constraints
    1. The type of a Service must be a ServiceInterface or an Interface.
    2. The direction property of a Service must be incoming.
- Notes
    - A Service represents a feature of a Participant that is the offer of a service by one participant to others using well defined terms, conditions and interfaces. A Service designates a Port that defines the connection point through which a Participant offers its capabilities and provides a service to clients.
    - A Service extends Port to specify a feature of a Participant that represents a service the Participant provides and offers for consumption by other participants. The service is defined by a ServiceInterface. It is implemented by the Participant either through delegation to its parts or through its methods. The service may be connected to a business MotivationalElement to indicate its intended value proposition. There may be constraints associated with the service that define its nonfunctional characteristics or

warranted qualities of service. This information may be used by potential consumers to determine if the service meets their needs.
    - A Service may include the specification of the value offered to another, and the offer to provide value to another. A Service is the visible point at which consumer requests are connected to providers and through which they interact in order to produce some real world effect.
    - A Service may also be viewed as the offer of some service or set of related services provided by a provider Participant that, when consumed by some consumer Participants, has some value or achieves some objective of the connected parties. A service is distinguished from a simple Operation in that it may involve a conversation between the parties as specified by some communication protocol that is necessary to meet the common objective.
    - The capabilities provided through the Service – its provided interfaces – are derived from the interfaces realized by the Service's ServiceInterface and further detained in the service contract. The capabilities required of consumers in order to use the Service – its required interfaces – are derived from the interfaces used by the Service's ServiceInterface. These are the same as the provided and required interfaces of the Port that is extended by Service.
    - A Service represents an interaction point through which a providing Participant with capabilities to provide a service interacts with a consuming participant having compatible needs. It represents a part at the end of a ServiceChannel connection and the point through which a provider satisfies a request.
    - A Service is typed by an Interface or ServiceInterface that, possibly together with a ServiceContract, completely characterizes specific capabilities of the producing and consuming participants' responsibilities with respect to that service. This includes provided interfaces that designate the capabilities of the Participant through this Service, and the required interfaces that represent what the Participant is required of consumers in order to use the provided capabilities. It also includes any protocols the providing Participant requires consumers to follow in the use of the capabilities of the Service.
    - If the type of a Service is a ServiceInterface, then the Service's provided Interfaces are the Interfaces realized by the ServiceInterface while its required Interfaces are those used by the ServiceInterface. If the type of a Service is a simple Interface, then the provided interface is that Interface and there is no required Interface and no protocol. If the ServiceInterface or UML interface typing a service port is defined as a role within a ServiceContract – the service port (and participant) is bound by the semantics and constraints of that service contract.

5.28 ServiceChannel

- Relationships
    - ServiceChannel **extends** metaclass Connector
- Constraints
    1. One end of a ServiceChannel must be a Request and the other a Service in an

architecture.
2. The Request and Service connected by a ServiceChannel must be compatible.
3. The contract Behavior for a ServiceChannel must be compatible with any protocols specified for the connected requests and Services.

- Notes
  - Additions to UML2: ServiceChannel extends UML2 Connector with more specific semantics for service and request compatibility.
  - A communication path between Services and Requests within an architecture.
  - A ServiceChannel is used to connect Requests of consumer Participants to Services of provider Participants at the ServiceChannel ends. A ServiceChannel enables communication between the Request and service.
  - A Request specifies a Participant's needs. A Service specifies a Participant's services offered. The type of a Request or Service is a ServiceInterface or Interface that defines the needs and capabilities accessed by a Request through Service, and the protocols for using them. Loosely coupled systems imply that services should be designed with little or no knowledge about particular consumers. Consumers may have a very different view of what to do with a service based on what they are trying to accomplish.
  - Loose coupling allows reuse in different contexts, reduces the effect of change, and is the key enabler of agile solutions through an SOA. In services models, ServiceChannels connect consumers and providers and therefore define the coupling in the system. They isolate the dependencies between consuming and providing participants to particular Request/service interaction points. However, for services to be used properly, and for Requests to be fully satisfied, Requests must be connected to compatible Services. This does not mean the Request Port and Service Port must have the same type, or that their ServiceInterfaces must be derived from some agreed upon ServiceContract as this could create additional coupling between the consumer and provider. Such coupling would for example make it more difficult for a service to evolve to meet needs of other consumers, to satisfy different contracts, or to support different versions of the same request without changing the service it is connected to.
  - Loosely coupled systems therefore require flexible compatibility across ServiceChannels. Compatibility can be established using UML2 specialization/generalization or realization rules. However, specialization/generalization, and to a lesser extent realization, are often impractical in environments where the classifiers are not all owned by the same organization. Both specialization and realization represent significant coupling between subclasses and realizing classifiers. If a superclass or realized class changes, then all the subclasses also automatically change while realizing classes must be examined to see if change is needed. This may be very undesirable if the subclasses are owned by another organization that is not in a position to synchronize its changes with the providers of other classifiers.
  - A Request is compatible with, and may be connected to a Service through a ServiceChannel if:
    1. The Request and Service have the same type, either an Interface or ServiceInterface.

2. The type of the Service is a specialization or realization of the type of the Request.
3. The Request and Service have compatible needs and capabilities respectively. This means the Service must provide an Operation for every Operation used through the Request, the Request must provide an Operation for every Operation used through the Service, and the protocols for how the capabilities are compatible between the Request and Service.
4. Any of the above are true for a subset of a ServiceInterface as defined by a port on that service interface.

5.29 ServiceContract

- Relationships
  - ServiceContract **extends** metaclass Collaboration
- Constraints
  - If the CollaborationUse for a ServiceInterface in a services architecture has isStrict=true (the default), then the parts must be compatible with the roles they are bound to. For parts to be compatible with a role, one of the following must be true: 1. The role and part have the same type.
    2. The part has a type that specializes the type of the role.
    3. The part has a type that realizes the type of the role.
    4. The part has a type that contains at least the ownedAttributes and ownedOperations of the role. In general this is a special case of item 3 where the part has an Interface type that realizes another Interface.
    5. The type of each role in a service contract shall have a uses dependency to the type of all roles that role is connected to.
- Notes
  - Additions to UML2: ServiceContract is a UML collaboration extended as a binding agreement between the parties, designed explicitly to show a service as a contract that is independent of but binding on the involved parties.
  - The basis of the service contract is also a UML collaboration that is focused on the interactions involved in providing a service.
  - A service contract is represented as a UML Collaboration (stereotype: <<ServiceContract>>) and defines specific roles each participant plays in the service contract. These roles have a name, an interface type that may be stereotyped as the “Provider” or “Consumer.” The consumer is expected to start the service, calling on the provider to provide something of value to the consumer.
  - A key part of a service is the ServiceContract (see Figure 6.9).
  - A service contract approach defines service specifications (the service contract) that specify how providers, consumers and (potentially) other roles work together to exchange value. The exchange of value is the enactment of the service. The service contract approach defines the roles each participant plays in the service (such as provider and consumer) and the interfaces they implement to play that role in that service. These interfaces are then the types of ports on the participant, which obligates the participant to

be able to play that role in that service contract. The service contract represents an agreement between the parties for how the service is to be provided and consumed. This agreement includes the interfaces, choreography, and any other terms and conditions. Note that the agreement may be asserted in advance or arrived at dynamically, as long as an agreement exists by the time the service in enacted. If a provider and consumer support different service contracts, there must be an agreement or determination that their different service contracts are compatible and consistent with other commitments of the same participants. Service contracts are frequently part of one or more services architectures that define how a set of participants provide and use services for a particular business purpose or process. In summary, the service contract approach is based on defining the service contract “in the middle” (between the parties) and having the ends (the participants) agree to their part in that contract, or adapt to it. The ServiceContract approach is most applicable where an enterprise, community SOA architecture or choreography is defined and then services built or adapted to work within that architecture, or when there are more than two parties involved in the service.

- The fundamental differences between the contract and interface based approaches is whether the interaction between participants are defined separately from the participants in a ServiceContract that defines the obligations of all the participants, or individually on each participants’ service and request.
- A ServiceContract defines the terms, conditions, interfaces, and choreography that interacting participants must agree to (directly or indirectly) for the service to be enacted; the full specification of a service that includes all the information, choreography, and any other “terms and conditions” of the service. A ServiceContract is binding on both the providers and consumers of that service or all participating parties in the case of a multi-party service. [...] A participant plays a role in the larger scope of a ServicesArchitecture and also plays a role as the provider or user of services specified by ServiceContracts.
- Each role, or party involved in a ServiceContract is defined by an Interface or ServiceInterface that is the type of the role. A ServiceContract is a binding contract - binding on any participant that has a service port typed by a role in a service contract. It defines the relationships between a set of roles defined by Interfaces and/or ServiceInterfaces.
- An important part of the ServiceContract is the choreography. The choreography is a specification of what is transmitted and when it is transmitted between parties to enact a service exchange. The choreography specifies exchanges between the parties - the data, assets, and obligations that go between the parties. The choreography defines what happens between the provider and consumer participants without defining their internal processes - their internal processes do have to be compatible with their ServiceContracts.
- The service contract separates the concerns of how all parties agree to provide or use the service from how any party implements their role in that service - or from their internal business process.
- The requirements for entities playing the roles in a ServiceContract are defined by

Interfaces or ServiceInterfaces used as the type of the role. The ServiceInterface specifies the provided and required interfaces that define all of the operations or signal receptions needed for the role it types - these will be every obligation, asset, or piece of data that the entity can send or receive as part of that service contract. Providing and using corresponding UML interfaces in this way “connects the dots” between the service contract and the requirements for any participant playing a role in that service as provider or consumer.

- A service contract represents an agreement between parties for how they will carry out the service. This agreement may be established early when the services are defined or late when they are used. But, by the time the service actually happens, it is happening with respect to some service contract. Existing “bottom up” services are frequently adapted to these service contracts, which may be specified at an enterprise, community, or system level.
- The interfaces defined as part of a service contract represent how a participant must interact to participate in that service. A participant interacts via a UML “port.” A port is a point of interaction within UML and is used in SoaML as the interaction point for providing, consuming, or playing any other role in a service as defined by the service contract.
- Multi-party service contracts are those with more than two participants - while this is a less common case, it does represent many business and technology interactions. […] As with the binary service contract, each participant provides their interface and uses the interfaces of each party they call, which is shown in terms of the interaction diagram as well as the dependencies between the interfaces. The choreography defines the rules for who calls who and when.
- Capabilities, ServiceContracts, and ServicesArchitectures provide a means of bridging between business concerns and SOA solutions by tying the business requirements realized by the contracts to the services and service participants that fulfill the contracts.
- A Collaboration, ServiceContract, or ServicesArchitecture represents a pattern of interaction between roles. This interaction may be informal and loosely defined as in a requirements sketch.
- The interaction between the provider and consumer is governed by a “ServiceContract” where both parties are (directly or indirectly) bound by that contract.
- A ServiceContract is the formalization of a binding exchange of information, goods, or obligations between parties defining a service.
- A ServiceContract is the specification of the agreement between providers and consumers of a service as to what information, products, assets, value, and obligations will flow between the providers and consumers of that service. It specifies the service without regard for realization, capabilities, or implementation. A ServiceContract does not require the specification of who, how, or why any party will fulfill their obligations under that ServiceContract, thus providing for the loose coupling of the SOA paradigm. In most cases a ServiceContract will specify two roles (provider and consumer) but other service roles may be specified as well. The ServiceContract may also own a behavior

that specifies the sequencing of the exchanges between the parties as well as the resulting state and delivery of the capability. The owned behavior is the choreography of the service and may use any of the standard UML behaviors such as an interaction, timing, state, or activity diagram.

- A ServiceContract may use other nested ServiceContracts representing nested services as a CollaborationUse. Such a nested service is performed and completed within the context of the larger grained service that uses it. A ServiceContract using nested ServiceContracts is called a compound service contract.
- One ServiceContract may specialize another service contract using UML generalization. A specialized contract must comply with the more general contract but may restrict the behavior and/or operations used. A specialized contract may be used as a general contract or as a specific agreement between specific parties for their use of that service.
- A ServicesContract (sic!) is used to model an agreement between two or more parties and may constrain the expected real world effects of a service. ServiceContracts can cover requirements, service interactions, quality of service agreements, interface and choreography agreements, and commercial agreements.
- Each service role in a service contract has a type, which must be a ServiceInterface or UML Interface or Class stereotyped as “Provider” or “Consumer.” The ServiceContract is a binding agreement on entities that implement the service type. That is, any party that “plays a role” in a Service Contract is bound by the service agreement, exchange patterns, behavior, and MessageType formats. Note that there are various ways to bind to or fulfill such an agreement, but compliance with the agreement is ultimately required to participate in the service. Due to the binding agreement, where the types of a service contract are used in a Service or Request no collaboration use is required.
- The Service contract is at the middle of the SoaML set of SOA architecture constructs. The highest level is described as a services architectures (at the community and participant levels) - where participants are working together using services. These services are then described by a ServiceContract. The details of that contract, as it relates to each participant, uses an Interface that in turn has operations that use the message data types that flow between participants. The service contract provides an explicit but high-level view of the service where the underlying details may be hidden or exposed, based on the needs of stakeholders.
- A ServiceContract can be used in support of multiple architectural goals, including:
  1. As part of the Service Oriented Architecture (SOA), including services architectures, participant architectures, information models, and business processes.
  2. Multiple views of complex systems:
     - A way of abstracting different aspects of services solutions.
     - Convey information to stakeholders and users of those services.
     - Highlight different subject areas of interest or concern.
  3. Formalizing requirements and requirement fulfillment:
     - Without constraining the architecture for how those requirements might be realized.

        - Allowing for separation of concerns.
    4. Bridge between business process models and SOA solutions:
        - Separates the what from the how.
        - Formal link between service implementation and the contracts it fulfills with more semantics than just traceability.
    5. Defining and using patterns of services.
    6. Modeling the requirements for a service:
        - Modeling the roles the consumers and providers play, the interfaces they must provide and/or require, and behavioral constraints on the protocol for using the service.
        - The foundation for formal Service Level Agreements.
    7. Modeling the requirements for a collection of services or service participants:
        - Specifying what roles other service participants are expected to play and their interaction choreography in order to achieve some desired result including the implementation of a composite service.
    8. Defining the choreography for a business process.
  - Each ServiceContract role has a type that must be a UML Interface or (in the case of a composite service contract) a ServiceInteface or Class. At least one such interface must be stereotyped as a “Provider” and one as a “Consumer.” The ServiceContract is a binding agreement on participants that implements the service type. That is, any party that “plays a role” in a Service Contract is bound by the service agreement, interfaces, exchange patterns, behavior, and Message formats. Note that there are various ways to bind to or fulfill such an agreement, but compliance with the agreement is ultimately required to participate in the service as a provider or consumer.
  - By typing a port with an interface or class that is the type of a role in a ServiceContract, the participant agrees to abide by that contract.

5.30 Service Description

- Notes
  - The description of how the participant interacts to provide or use a service is encapsulated in a specification for the service - there are three ways to specify a service interaction - a UML Interface, a ServiceInterface, and a ServiceContract. These different ways to specify a service relate to the SOA approach and the complexity of the service, but in each case they result in interfaces and behaviors that define how the participant will provide or use a service through ports. The service descriptions are independent of, but consistent with how the provider provides the service or how (or why) the consumer consumes it. This separation of concerns between the service description and how it is implemented is fundamental to SOA. A service specification specifies how consumers and providers are expected to interact through their ports to enact a service, but not how they do it.
  - Regardless of how services are identified, they are formalized by service descriptions. A service description defines the purpose of the service and any interaction or communication protocol for how to properly use and provide a service. A service

description may define the complete interface for a service from its own perspective, irrespective of any consumer request it might be connected to. Alternatively, the agreement between a consumer request and provider service may be captured in a common service contract defined in one place, and constraining both the consumer's request service interface and the provider's service interface.

5.31 ServiceInterface

- Relationships
  - ServiceInterface **extends** metaclass Class
  - ServiceInterface **realizes** * CapabilityExposure
- Structure
  - <<ServiceInterface>> stereotype
  - The name of the service indicating what it does or is about.
  - The provided service interactions through realized interfaces.
  - The needs of consumers in order to use the service through the used interfaces.
  - A detailed specification of each exchange of information, obligations, or assets using an operation or reception; including its name, preconditions, post conditions, inputs and outputs, and any exceptions that might be raised.
  - Any protocol or rules for using the service or how consumers are expected to respond and when through an owned behavior of the service interface.
  - Rules for how the service must be implemented by providers.
  - Constraints that can determine if the service has successfully achieved its intended purpose.
  - If exposed by the provider, what capabilities are used to provide the service or are made available through the service.
- Constraints
  - All parts of a ServiceInterface must be typed by the Interfaces realized or used by the ServiceInterface.
- Notes
  - The ServiceInterface is used to type «Service» ports and «Request» ports of participants. The provider of the service uses a «Service» port and the consumer of the service uses a «Request» port. The «Service» ports and «Request» ports are the points of interaction for the service.
  - A ServiceInterface based approach allows for bi-directional services, those where there are "callbacks" from the provider to the consumer as part of a conversation between the parties. A service interface is defined in terms of the provider of the service and specifies the interface that the provider offers as well as the interface, if any, it expects from the consumer. The service interface may also specify the choreography of the service - what information is sent between the provider and consumer and in what order. A consumer of a service specifies the service interface they require using a request port. The provider and consumer interfaces must either be the same or compatible. If they are compatible, the provider can provide the service to that consumer. The consumer must adhere to the

provider’s service interface, but there may not be any prior agreement between the provider and consumer of a service. Compatibility of service interfaces determines whether these agreements are consistent and can therefore be connected to accomplish the real world effect of the service and any exchange in value. The ServiceInterface is the type of a “Service” port on a provider and the type of a “Request” port on the consumer. In summary, the consumer agrees to use the service as defined by its service interface, and the provider agrees to provide the service according to its service interface. Compatibility of service interfaces determines whether these agreements are consistent and can therefore be connected. The ServiceInterface approach is most applicable where existing capabilities are directly exposed as services and then used in various ways, or in situations that involve one or two parties in the service protocol.

- Like a UML interface, a ServiceInterface defines or specifies a service and can be the type of a service port. The service interface has the additional feature that it can specify a bi-directional service having a protocol – where both the provider and consumer have responsibilities to invoke and respond to operations, send and receive messages or events. The service interface is defined from the perspective of the service provider using three primary sections: the provided and required Interfaces, the ServiceInterface class and the protocol Behavior.
- The provided and required Interfaces are standard UML interfaces that are realized or used by the ServiceInterface. The interfaces that are realized specify the value provided, the messages that will be received by the provider (and correspondingly sent by the consumer). The interfaces that are used by the ServiceInterface define the value required, the messages or events that will be received by the consumer (and correspondingly sent by the provider). Typically only one interface will be provided or required, but not always.
- The enclosed parts of the ServiceInterface represent the roles that will be played by the connected participants involved with the service. The role that is typed by the realized interface will be played by the service provider; the role that is typed by the used interface will be played by the consumer.
- A ServiceInterface is not a UML interface, but a class providing and requiring the interfaces of the provider.
- A ServiceInterface is a UML Class and defines specific roles each participant plays in the service interaction. These roles have a name and an interface type. The interface of the provider (which must be the type of one of the parts in the class) is realized (provided) by the ServiceInterface class. The interface of the consumer (if any) must be used by the class.
- […] root of the service interface and represents the service itself – the terms and conditions under which the service can be enacted and the results of the service.
- The service interface may be related to business goals or requirements. The service interface can also be used in services architectures to show how multiple services and participants work together for a business purpose.
- Where a ServiceInterface has a behavior and is also used as a type in a ServiceContract,

the behavior of that ServiceInterface must comply with the service contract. However, common practice would be to specify a behavior in the service contract or service interface, not both.

- Provides the definition of a service. Defines the specification of a service interaction as the type of a «Service» or «Request» port.
- A ServiceInterface defines the interface and responsibilities of a participant to provide or consume a service. It is used as the type of a Service or Request Port. A ServiceInterface is the means for specifying how a participant is to interact to provide or consume a Service. A ServiceInterface may include specific protocols, commands, and information exchange by which actions are initiated and the result of the real world effects are made available as specified through the functionality portion of a service. A ServiceInterface may address the concepts associated with ownership, ownership domains, actions communicated between legal peers, trust, business transactions, authority, delegation, etc.
- A Service port or Request port or role may be typed by either a ServiceInterface or a simple UML2 Interface. In the latter case, there is no protocol associated with the Service. Consumers simply invoke the operations of the Interface. A ServiceInterface may also specify various protocols for using the functional capabilities defined by the servicer interface. This provides reusable protocol definitions in different Participants providing or consuming the same Service.
- A ServiceInterface may specify “parts” and “owned behaviors” to further define the responsibilities of participants in the service. The parts of a ServiceInterface are typed by the Interfaces realized (provided) and used (required) by the ServiceInterface and represent the potential consumers and providers of the functional capabilities defined in those interfaces. The owned behaviors of the ServiceInterface specify how the functional capabilities are to be used by consumers and implemented by providers. A ServiceInterface therefore represents a formal agreement between consumer Requests and providers that may be used to match needs and capabilities.
- A service interface may it self have service ports or request ports that define more granular services that serve to make up a larger composite service. This allows “enterprise scale” services to be composed from multiple, smaller services between the same parties. Internal to a participant connections can be made for the entire service or any one of the sub-services, allowing delegation of responsibility for specific parts of the service contract.
- One or more ServiceInterfaces may also be combined in a ServiceContract which can further specify and constrain related services provided and consumed by Participants.
- A ServiceInterface defines the following information:

Function	Metadata
An indication of what the service does or is about	The ServiceInterface name
The service defined by the ServiceInterface that will be provided by any Participant having a	The provided Interfaces containing Operations modeling the capabilities.

Service typed by the ServiceInterface, or used by a Participant having a Request typed by the ServiceInterface.	As in UML2, provided interfaces are designated using an InterfaceRealization between the ServiceInterface and other Interfaces.
Any service interaction consumers are expected to provide or consume in order to use or interact with a Service typed by this ServiceInterface.	Required Interfaces containing Operations modeling the needs. As in UML2, required interfaces are designated using a Usage between the ServiceInterface and other Interfaces.
The detailed specification of an interaction providing value as a service including: • Its name, often a verb phrase indicating what it does • Any required or optional service data inputs and outputs • Any preconditions consumers are expected to meet before using the capability • Any post conditions consumers participants can expect, and other providers must provide upon successful use of the service • Any exceptions or fault conditions that might be raised if the capability cannot be provided for some reason even though the preconditions have been met	Each atomic interaction of a ServiceInterface is modeled as an Operation or event reception in its provided or required Interfaces. From UML2, an Operation has Parameters defining its inputs and outputs, preconditions and post-conditions, and may raise Exceptions. Operation Parameters may also be typed by a MessageType.
Any communication protocol or rules that determine when a consumer can use the capabilities or in what order	An ownedBehavior of the ServiceInterface. This behavior expresses the expected interaction between the consumers and providers of services typed by this ServiceInterface. The ownedBehavior could be any Behavior including Activity, Interaction, StateMachine, ProtocolStateMachine, or OpaqueBehavior.
Requirements any implementer must meet when providing the service	This is the same ownedBehavior that defines the consumer protocol just viewed from an implementation rather than a usage perspective.
Constraints that reflect what successful use of the service is intended to accomplish and how it would be evaluated	UML2 Constraints in ownedRules of the ServiceInterface.
Policies for using the service such as security and transaction scopes for maintaining integrity or recovering from the inability to successfully perform the service or any required service	Policies may also be expressed as constraints.
Qualities of service consumers should expect and providers are expected to provide such as:	The OMG QoS specification may be used to model qualities of service constraints for a

cost, availability, performance, footprint, suitability to the task, competitive information, etc.	ServiceInterface.
A service composed of other services as a composite service.	Service ports or request ports on the service interface.

    - Participants specify their needs with Request ports and their capabilities with Service ports. Services and Requests, like any part, are described by their type which is either an Interface or a ServiceInterface. A request port may be connected to a compatible Service port in an assembly of Participants through a ServiceChannel. These connected participants are the parts of the internal structure of some other Participant where they are assembled in a context for some purpose, often to implement another service, and often adhering to some ServicesArchitecture. ServiceChannel specifies the rules for compatibility between a Request and Service. Essentially they are compatible if the needs of the Request are met by the capabilities of the Service and they are both structurally and behaviorally compatible.
    - A ServiceInterface specifies the receptions and operations it receives through InterfaceRealizations. A ServiceInterface can realize any number of Interfaces. Some platform specific models may restrict the number of realized interfaces to at most one. A ServiceInterface specifies its required needs through Usage dependences to Interfaces. These realizations and usages are used to derive the provided and required interfaces of Request and service ports typed by the ServiceInterface.
    - The parts of a ServiceInterface are typed by the interfaces realized or used by the ServiceInterface. These parts (or roles) may be used in the ownedBehaviors to indicate how potential consumers and providers of the service are expected to interact. A ServiceInterface may specify communication protocols or behavioral rules describing how its capabilities and needs must be used. These protocols may be specified using any UML2 Behavior.
    - A ServiceInterface may have ownedRules determine the successful accomplishment of its service goals. An ownedRule is a UML constraint within any namespace, such as a ServiceInterface.

5.32 Simple Interface

- Notes
    - The simple interface focuses attention on a one-way interaction provided by a participant on a port represented as a UML interface. The participant receives operations on this port and may provide results to the caller. This kind of one-way interface can be used with “anonymous” callers and the participant makes no assumptions about the caller or the choreography of the service. The one-way service corresponds most directly to simpler “RPC style web services” as well as many “OO” programming language objects. A simple interface used to type a service port can optionally be a ServiceInterface. Simple interfaces are often used to expose the “raw” capability of existing systems or to define simpler services that have no protocols. Simple interfaces are the degenerate case

of both the ServiceInterface and ServiceContract where the service is unidirectional - the consumer calls operations on the provider - the provider doesn't callback the consumer and may not even know who the consumer is.
    - Simple interfaces define one-way services that do not require a protocol. Such services may be defined with only a single UML interface and then provided on a "Service" port and consumed on a "Request" port.
    - When used in the «Service» port the [simple] interface is provided. When used in the «Request» port the service is used and the resulting ports are compatible.
    - In simple unidirectional services, the consumer interface may be missing or empty.
    - For a simple interface to be used in a ServicesArchitecture it must be put in the context of a ServiceContract, as shown in Figure 6.17. […] Note that the simple interface is used as the provider, the consumer's role type may be empty and the consumer will use a Request. Adding a ServiceContract for a simple interface is then equivalent to the ServiceContract defined with one interface [...]. The only difference being a top-down vs. bottom up approach to get to the same result. The consumer side of this service would use a "Request" port.

4.33 ServicesArchitecture

- Relationships
    - ServicesArchitecture **extends** metaclass Collaboration
- Structure
    - The services architecture of a community is modeled as collaboration stereotyped as «ServicesArchitecture».
- Constraints
    - The parts of a ServicesArchitecture must be typed by a Participant or capability. Each participant satisfying roles in a ServicesArchitecture shall have a port for each role binding attached to that participant. This port shall have a type compliant with the type of the role used in the ServiceContract.
- Notes
    - A ServicesArchitecture (see Figure 6.7) is defined using a UML Collaboration.
    - SoaML can also be used "in the large" where we are enabling an organization or community to work more effectively using an inter-related set of services. Such services are executed in the context of this enterprise, process, or community and so depend on agreements captured in the services architecture of that community. A SoaML ServicesArchitecture shows how multiple participants work together, providing and using services to enable business goals or processes.
    - A ServicesArchitecture (or SOA) is a network of participant roles providing and consuming services to fulfill a purpose. The services architecture defines the requirements for the types of participants and service realizations that fulfill those roles.
    - Within a ServicesArchitecture, participant roles provide and employ any number of services. The purpose of the services architecture is to specify the SOA of some organization, community, component or process to provide mutual value. The

participants specified in a ServicesArchitecture provide and consume services to achieve that value. The services architecture may also have a business process to define the tasks and orchestration of providing that value. The services architecture is a high-level view of how services work together for a purpose. The same services and participants may be used in many such architectures, providing reuse.

- A services architecture has components at two levels of granularity: The community services architecture is a “top level” view of how independent participants work together for some purpose. The services architecture of a community does not assume or require any one controlling entity or process.
- The services architecture serves to define the requirements of each of the participants. The participant roles are filled by participants with service ports required of the entities that fill these roles and are then bound by the services architectures in which they participate.
- A ServicesArchitecture can be used to specify the architecture for a particular Participant. Within a participant, where there is a concept of “management” exists, a services architecture illustrates how realizing participants and external collaborators work together and would often be accompanied by a business process. A ServicesArchitecture or specification class may be composed from other services architectures and service contracts.
- [...] there is a way to decompose a ServicesArchitecture and visualize how services can be implemented by using still other services. A participant can be further described by its internal services architecture or a composite component. Such participant can also use internal or external services, assemble other participants, business processes, and other forms of implementation. SoaML shows how the internal structure of a participant is described using other services. This is done by defining a ServicesArchitecture for participants in a more granular (larger scale) services architecture as is shown in Figure 6.7 and Figure 6.8.
- Capabilities, ServiceContracts, and ServicesArchitectures provide a means of bridging between business concerns and SOA solutions by tying the business requirements realized by the contracts to the services and service participants that fulfill the contracts.
- A Collaboration, ServiceContract, or ServicesArchitecture represents a pattern of interaction between roles. This interaction may be informal and loosely defined as in a requirements sketch.
- The high-level view of a Service Oriented Architecture that defines how a set of participants works together, forming a community, for some purpose by providing and using services.
- A ServicesArchitecture (a SOA) describes how participants work together for a purpose by providing and using services expressed as service contracts. By expressing the use of services, the ServicesArchitecture implies some degree of knowledge of the dependencies between the participants in some context. Each use of a service in a ServicesArchitecture is represented by the use of a ServiceContract bound to the roles of participants in that architecture.

- Note that use of a ServicesArchitecture is optional but is recommended to show a high level view of how a set of Participants work together for some purpose. Where as simple services may not have any dependencies or links to a business process, enterprise services can often only be understood in context. The services architecture provides that context, and may also contain a behavior, which is the business process. The participant's roles in a services architecture correspond to the swim lanes or pools in a business process.
- A ServicesArchitecture may be specified externally - in a "B2B" type collaboration where there is no controlling entity or as the ServicesArchitecture of a participant - under the control of a specific entity and/or business process. A "B2B" services architecture uses the "ServicesArchitecture" stereotype on a collaboration.
- Standard UML2 Collaboration semantics are augmented with the requirement that each participant used in a services architecture must have a port compliant with the ServiceContracts the participant provides or uses, which is modeled as a role binding to the use of a service contract.
- By specifying a ServicesArchitecture we can understand the services in our enterprise and communities in context and recognize the real (business) dependencies that exist between the participants. The purpose of the services architecture may also be specified as a comment.
- Each participant in a ServicesArchitecture must have a port that is compatible with the roles played in each ServiceContract role it is bound to.

5.34 UseCase

- Notes
  - A ServiceContract, which is also a classifier, can realize UseCases. In this case, the actors in the use case may also be used to type roles in the service contract, or the actors may realize the same Interfaces used to type the roles.

FW.6 Topology and Orchestration Specification for Cloud Applications (TOSCA) [6]

6.1 AppliesTo

- Relationships
  - AppliesTo has **1..*** NodeTypeReference
- Notes
  - This OPTIONAL element specifies the set of Node Types the Policy Type is applicable to, each defined as a separate, nested NodeTypeReference element.

6.2 ArtifactReference

- Relationships
  - ArtifactReference **has *** Include
    or Exclude
- Structure
  - reference: This attribute contains a URI pointing to an actual artifact.

6.3 ArtifactReferences

- Relationships
  - ArtifactReferences **has 1..*** ArtifactReference
- Notes
  - This OPTIONAL element contains one or more references to the actual artifact proper, each represented by a separate ArtifactReference element.

6.4 ArtifactTemplate

- Relationships
  - ArtifactTemplate **has 0..1** Documentation
  - ArtifactTemplate **has 0..1** Properties
  - ArtifactTemplate **has 0..1** PropertyConstraints
  - ArtifactTemplate **has 0..1** ArtifactReferences
- Structure
  - id: This attribute specifies the identifier of the Artifact Template. The identifier of the Artifact Template MUST be unique within the target namespace.
  - name: This OPTIONAL attribute specifies the name of the Artifact Template.
  - type: The QName value of this attribute refers to the Artifact Type providing the type of the Artifact Template.
- Notes
  - This element specifies a template describing an artifact referenced by parts of a Service Template. For example, the installable artifact for an application server node might be defined as an artifact template.

6.5 ArtifactType

- Relationships
  - ArtifactType **has 0..1** Documentation
  - ArtifactType **has 0..1** Tags
  - ArtifactType **has 0..1** DerivedFrom
  - ArtifactType **has 0..1** PropertiesDefinition
- Structure
  - name: This attribute specifies the name or identifier of the Artifact Type, which MUST be unique within the target namespace.
  - targetNamespace: This OPTIONAL attribute specifies the target namespace to which the definition of the Artifact Type will be added. If not specified, the Artifact Type definition will be added to the target namespace of the enclosing Definitions document.
  - abstract: This OPTIONAL attribute specifies that no instances can be created from Artifact Templates of that abstract Artifact Type, i.e. the respective artifacts cannot be used directly as deployment or implementation artifact in any context. Note: an abstract Artifact Type MUST NOT be declared as final and vice versa.
  - final: This OPTIONAL attribute specifies that other Artifact Types MUST NOT be derived from this Artifact Type.
- Notes
  - This element specifies a type of artifact used within a Service Template. Artifact Types might be, for example, application modules such as .war files or .ear files, operating system packages like RPMs, or virtual machine images like .ova files.

6.6 BoundaryDefinitions

- Relationships
  - BoundaryDefinitions **has 0..1** Properties
  - BoundaryDefinitions **has 0..1** PropertyConstraints
  - BoundaryDefinitions **has 0..1** Requirements
  - BoundaryDefinitions **has 0..1** Capabilities
  - BoundaryDefinitions **has 0..1** Policies
  - BoundaryDefinitions **has 0..1** Interfaces
- Notes
  - This OPTIONAL element specifies the properties the Service Template exposes beyond its boundaries, i.e. properties that can be observed from outside the Service Template. The BoundaryDefinitions element has the following properties.

6.7 Capabilities

- Relationships
  - Capabilities **has 1..*** Capability
- Notes
  - This OPTIONAL element specifies Capabilities exposed by the Service Template. Those Capabilities correspond to Capabilities of Node Templates within the Service Template

that are propagated beyond the boundaries of the Service Template. Each Capability is defined by a separate, nested Capability element.
    - This element contains a list of capabilities for the Node Template, according to the list of capability definitions of the Node Type specified in the type attribute of the Node Template. Each capability is specified in a separate nested Capability element.

6.8 Capability

- Relationships
    - Requirement **has 0..1** Properties
    - Requirement **has 0..1** PropertyConstraints
- Structure
    - id: This attribute specifies the identifier of the Capability. The identifier of the Capability MUST be unique within the target namespace.
    - name: (i) This OPTIONAL attribute allows for specifying a name of the Capability other than that specified by the referenced Capability of a Node Template. (ii) This attribute specifies the name of the Capability. The name and type of the Capability MUST match the name and type of a Capability Definition in the Node Type specified in the type attribute of the Node Template.
    - ref: This attribute references a Capability element of a Node Template within the Service Template.
    - type: The QName value of this attribute refers to the Capability Type definition of the Capability. This Capability Type denotes the semantics and well as potential properties of the Capability.

6.9 CapabilityDefinition

- Relationships
    - CapabilityDefinition **has 0..1** Constraints
- Structure
    - name: This attribute specifies the name of the defined capability and MUST be unique within the CapabilityDefinitions of the current Node Type. Note that one Node Type might define multiple capabilities of the same Capability Type, in which case each occurrence of a capability definition is uniquely identified by its name.
    - capabilityType: This attribute identifies by QName the Capability Type of capability that is being defined by the current CapabilityDefinition.
    - lowerBound: This OPTIONAL attribute specifies the lower boundary of requiring nodes that the defined capability can serve. The default value for this attribute is one. A value of zero is invalid, since this would mean that the capability cannot actually satisfy any requiring nodes.
    - upperBound: This OPTIONAL attribute specifies the upper boundary of client requirements the defined capability can serve. The default value for this attribute is one. A value of “unbounded” indicates that there is no upper boundary.

6.10 CapabilityDefinitions

- Notes
  - This OPTIONAL element specifies the capabilities that the Node Type exposes (see section 3.4 for details). Each capability is defined in a nested CapabilityDefinition element.

6.11 CapabilityType

- Relationships
  - CapabilityType **has 0..1** Documentation
  - CapabilityType **has 0..1** Tags
  - CapabilityType **has 0..1** DerivedFrom
  - CapabilityType **has 0..1** PropertiesDefinition
- Structure
  - name: This attribute specifies the name or identifier of the Capability Type, which MUST be unique within the target namespace.
  - targetNamespace: This OPTIONAL attribute specifies the target namespace to which the definition of the Capability Type will be added. If not specified, the Capability Type definition will be added to the target namespace of the enclosing Definitions document.
  - abstract: This OPTIONAL attribute specifies that no instances can be created from Node Templates of a Node Type that defines a capability of this Capability Type. Note: an abstract Capability Type MUST NOT be declared as final and vice versa.
  - final: This OPTIONAL attribute specifies that other Capability Types MUST NOT be derived from this Capability Type.
- Notes
  - This element specifies a type of Capability that can be exposed by Node Types used in a Service Template.

6.12 Constraint

- Structure
  - constraintType: This attribute specifies the type of constraint. According to this type, the body of the Constraint element will contain type specific content.

6.13 Constraints

- Relationships
  - Constraints **has 1..*** Constraint
- Notes
  - This OPTIONAL element contains a list of Constraint elements that specify additional constraints on the requirement definition. For example, if a database is needed a constraint on supported SQL features might be expressed.
  - This OPTIONAL element contains a list of Constraint elements that specify additional constraints on the capability definition.

6.14 Definitions

- Relationships
  - Definitions **has 0..1** Extensions
  - Definitions **has *** Import
  - Definitions **has 0..1** Types
  - Definitions **has at least one** ServiceTemplate **or** NodeType **or** NodeTypeImplementation **or** RelationshipType **or** RelationshipTypeImplementation **or** RequirementType **or** CapabilityType **or** ArtifactType **or** ArtifactTemplate **or** PolicyType **or** PolicyTemplate
  - Definitions **has 0..1** Documentation
- Structure
  - id: This attribute specifies the identifier of the Definitions document which MUST be unique within the target namespace.
  - name: This OPTIONAL attribute specifies a descriptive name of the Definitions document.
  - targetNamespace: The value of this attribute specifies the target namespace for the Definitions document. All elements defined within the Definitions document will be added to this namespace unless they override this attribute by means of their own targetNamespace attributes.
  - Extensions: This OPTIONAL element specifies namespaces of TOSCA extension attributes and extension elements. If present, the Extensions element MUST include at least one Extension element.
  - Import: This element declares a dependency on external TOSCA Definitions, XML Schema definitions, or WSDL definitions. Any number of Import elements MAY appear as children of the Definitions element.

6.15 DeploymentArtifact

- Structure
  - name: (i) This attribute specifies the name of the artifact. Uniqueness of the name within the scope of the encompassing Node Template SHOULD be guaranteed by the definition. (ii) This attribute specifies the name of the artifact, which SHOULD be unique within the scope of the encompassing Node Type Implementation.
  - artifactType: This attribute specifies the type of this artifact. The QName value of this attribute SHOULD correspond to the QName of an ArtifactType defined in the same Definitions document or in an imported document. The artifactType attribute specifies the artifact type specific content of the DeploymentArtifact element body and indicates the type of Artifact Template referenced by the Deployment Artifact via the artifactRef attribute. Note, that a deployment artifact specified with the Node Template under definition overrides any deployment artifact of the same name and the same artifactType (or any Artifact Type it is derived from) specified with the Node Type Implementation implementing the Node Type given as value of the type attribute of the Node Template under definition. Otherwise, the deployment artifacts of Node Type Implementations and

the deployment artifacts defined with the Node Template are combined.
    - artifactRef: This OPTIONAL attribute contains a QName that identifies an Artifact Template to be used as deployment artifact. This Artifact Template can be defined in the same Definitions document or in a separate, imported document. The type of Artifact Template referenced by the artifactRef attribute MUST be the same type or a sub-type of the type specified in the artifactType attribute.
- Notes
    - This element specifies one deployment artifact.
    - Note: Multiple deployment artifacts MAY be defined in a Node Type Implementation. One reason could be that multiple artifacts (maybe of different types) are needed to materialize a node as a whole. Another reason could be that alternative artifacts are provided for use in different contexts (e.g. different installables of a software for use in different operating systems).

6.16 DeploymentArtifacts

- Relationships
    - DeploymentArtifacts **has 1..*** DeploymentArtifact
- Notes
    - This element specifies the deployment artifacts relevant for the Node Template under definition. Its nested DeploymentArtifact elements specify details about individual deployment artifacts.
    - This element specifies a set of deployment artifacts relevant for materializing instances of nodes of the Node Type being implemented.

6.17 DerivedFrom

- Structure
    - typeRef: (i) The QName specifies the Node Type from which this Node Type derives its definitions. (ii) The QName specifies the Relationship Type from which this Relationship Type derives its definitions. (iii) The QName specifies the Requirement Type from which this Requirement Type derives its definitions and semantics. (iv) The QName specifies the Capability Type from which this Capability Type derives its definitions and semantics. (v) The QName specifies the Artifact Type from which this Artifact Type derives its definitions and semantics. (vi) nodeTypeImplementationRef: The QName specifies the Node Type Implementation from which this Node Type Implementation derives. (vii) relationshipTypeImplementationRef: The QName specifies the Relationship Type Implementation from which this Relationship Type Implementation derives.
- Notes
    - This is an OPTIONAL reference to another Node Type from which this Node Type derives. Conflicting definitions are resolved by the rule that local new definitions always override derived definitions. See section 6.3 Derivation Rules for details.
    - This is an OPTIONAL reference to another Node Type Implementation from which this

Node Type Implementation derives. See section 7.3 Derivation Rules for details.
    - This is an OPTIONAL reference to another Relationship Type from which this Relationship Type is derived. Conflicting definitions are resolved by the rule that local new definitions always override derived definitions. See section 8.3 Derivation Rules for details.
    - This is an OPTIONAL reference to another Relationship Type Implementation from which this Relationship Type Implementation derives. See section 9.3 Derivation Rules or details.
    - This is an OPTIONAL reference to another Requirement Type from which this Requirement Type derives. See section 10.3 Derivation Rules for details.
    - This is an OPTIONAL reference to another Capability Type from which this Capability Type derives. See section 11.3 Derivation Rules for details.
    - This is an OPTIONAL reference to another Artifact Type from which this Artifact Type derives. See section 12.3 Derivation Rules for details.
    - This is an OPTIONAL reference to another Policy Type from which this Policy Type derives. See section 14.3 Derivation Rules for details.

6.18 Documentation

- Notes
    - All TOSCA elements MAY use the documentation element to provide annnotation (sic!) for users. The content could be a plain text, HTML, and so on.

6.19 Exclude

- Structure
    - pattern: This attribute contains a pattern definition for files that are to be excluded in the overall artifact reference. For example, a pattern of “*.sh” would exclude all bash scripts contained in a directory.
- Notes
    - This OPTIONAL element can be used to define a pattern of files that are to be excluded from the artifact reference in case the reference points to a complete directory.

6.20 Extensions

- Relationships
    - Extensions **has 0..1** Documentation
- Structure
    - namespace: This attribute specifies the namespace of TOSCA extension attributes and extension elements.
    - mustUnderstand: This OPTIONAL attribute specifies whether the extension MUST be understood by a compliant implementation. If the mustUnderstand attribute has value “yes” (which is the default value for this attribute) the extension is mandatory. Otherwise, the extension is optional. If a TOSCA implementation does not support one or more of the mandatory extensions, then the Definitions document MUST be rejected. Optional extensions MAY be ignored. It is not necessary to declare optional extensions.

The same extension URI MAY be declared multiple times in the Extensions element. If an extension URI is identified as mandatory in one Extension element and optional in another, then the mandatory semantics have precedence and MUST be enforced. The extension declarations in an Extensions element MUST be treated as an unordered set.

6.21 ImplementationArtifact

- Structure
  - name: This attribute specifies the name of the artifact, which SHOULD be unique within the scope of the encompassing Node Type Implementation.
  - interfaceName: This OPTIONAL attribute specifies the name of the interface that is implemented by the actual implementation artifact. If not specified, the implementation artifact is assumed to provide the implementation for all interfaces defined by the Node Type referred to by the nodeType attribute of the containing NodeTypeImplementation.
  - operationName: This OPTIONAL attribute specifies the name of the operation that is implemented by the actual implementation artifact. If specified, the interfaceName MUST be specified and the specified operationName MUST refer to an operation of the specified interface. If not specified, the implementation artifact is assumed to provide the implementation for all operations defined within the specified interface.
  - artifactType: This attribute specifies the type of this artifact. The QName value of this attribute SHOULD correspond to the QName of an ArtifactType defined in the same Definitions document or in an imported document. The artifactType attribute specifies the artifact type specific content of the ImplementationArtifact element body and indicates the type of Artifact Template referenced by the Implementation Artifact via the artifactRef attribute.
  - artifactRef: This OPTIONAL attribute contains a QName that identifies an Artifact Template to be used as implementation artifact. This Artifact Template can be defined in the same Definitions document or in a separate, imported document. The type of Artifact Template referenced by the artifactRef attribute MUST be the same type or a sub-type of the type specified in the artifactType attribute.
- Notes
  - This element specifies one implementation artifact of an interface or an operation.
  - Note: if no Artifact Template is referenced, the artifact type specific content of the ImplementationArtifact element alone is assumed to represent the actual artifact. For example, a simple script could be defined in place within the ImplementationArtifact element.

6.22 ImplementationArtifacts

- Relationships
  - ImplementationArtifacts **has 1..*** ImplementationArtifact
- Notes
  - This element specifies a set of implementation artifacts for interfaces or operations of a Node Type.

- This element specifies a set of implementation artifacts for interfaces or operations of a Relationship Type.

6.23 Import

- Relationships
  - Import **has 0..1** Documentation
- Structure
  - namespace: This OPTIONAL attribute specifies an absolute URI that identifies the imported definitions. An Import element without a namespace attribute indicates that external definitions are in use, which are not namespace-qualified. If a namespace attribute is specified then the imported definitions MUST be in that namespace. If no namespace is specified then the imported definitions MUST NOT contain a targetNamespace specification. The namespace http://www.w3.org/2001/XMLSchema is imported implicitly. Note, however, that there is no implicit XML Namespace prefix defined for http://www.w3.org/2001/XMLSchema.
  - location: This OPTIONAL attribute contains a URI indicating the location of a document that contains relevant definitions. The location URI MAY be a relative URI, following the usual rules for resolution of the URI base [XML Base, RFC 2396]. An Import element without a location attribute indicates that external definitions are used but makes no statement about where those definitions might be found. The location attribute is a hint and a TOSCA compliant implementation is not obliged to retrieve the document being imported from the specified location.
  - importType: This REQUIRED attribute identifies the type of document being imported by providing an absolute URI that identifies the encoding language used in the document. The value of the importType attribute MUST be set to http://docs.oasis-open.org/tosca/ns/2011/12 when importing Service Template documents, to http://schemas.xmlsoap.org/wsdl/ when importing WSDL 1.1 documents, and to http://www.w3.org/2001/XMLSchema when importing an XSD document.

6.24 Include

- Structure
  - pattern: This attribute contains a pattern definition for files that are to be included in the overall artifact reference. For example, a pattern of “*.py” would include all python scripts contained in a directory.
- Notes
  - This OPTIONAL element can be used to define a pattern of files that are to be included in the artifact reference in case the reference points to a complete directory.

6.25 InputParameter

- Structure
  - name: This attribute specifies the name of the input parameter, which MUST be unique within the set of input parameters defined for the operation.
  - type: This attribute specifies the type of the input parameter.

- required: This OPTIONAL attribute specifies whether or not the input parameter is REQUIRED (required attribute with a value of “yes” – default) or OPTIONAL (required attribute with a value of “no”).

6.26 InputParameters

- Relationships
  - InputParameters has 1..* InputParameter
- Notes
  - This OPTIONAL property contains a list of one or more input parameter definitions for the Plan, each defined in a nested, separate InputParameter element.

6.27 InstanceState

- Structure
  - state: This attribute specifies a URI that identifies a potential state.

6.28 InstanceStates

- Notes
  - This OPTIONAL element lists the set of states an instance of this Node Type can occupy. Those states are defined in nested InstanceState elements.
  - This OPTIONAL element lists the set of states an instance of this Relationship Type can occupy at runtime. Those states are defined in nested InstanceState elements.

6.29 Interface

- Relationships
  - Interfaces has **1..*** Operation
- Structure
  - name: (i) This attribute specifies the name of the interfaces as either a URI or an NCName that MUST be unique in the scope of the Service Template’s boundary definitions. (ii) The name of the interface. This name is either a URI or it is an NCName that MUST be unique in the scope of the Node Type being defined.
- Notes
  - This element specifies one interfaces exposed by the Service Template.

6.30 Interfaces

- Relationships
  - Interfaces has **1..*** Interface
- Notes
  - This OPTIONAL element specifies the interfaces with operations that can be invoked on complete service instances created from the Service Template.
  - This element contains the definitions of the operations that can be performed on (instances of) this Node Type. Such operation definitions are given in the form of nested Interface elements.

6.31 NodeOperation

- Structure
  - nodeRef: specifies reference to Node Template.
  - interfaceName: specific interface to be mapped to the operation exposed by the Service Template
  - operationName: specific operation to be mapped to the operation exposed by the Service Template
- Notes
  - This element specifies a reference to an operation of a Node Template.

6.32 NodeTemplate

- Relationships
  - NodeTemplate **has 0..1** Properties
  - NodeTemplate **has 0..1** PropertyConstraints
  - NodeTemplate **has 0..1** Requirements
  - NodeTemplate **has 0..1** Capabilities
  - NodeTemplate **has 0..1** Policies
  - NodeTemplate **has 0..1** DeploymentArtifacts
- Structure
  - id: This attribute specifies the identifier of the Node Template. The identifier of the Node Template MUST be unique within the target namespace.
  - name: This OPTIONAL attribute specifies the name of the Node Template.
  - type: The QName value of this attribute refers to the Node Type providing the type of the Node Template.
  - minInstances: This integer attribute specifies the minimun number of instances to be created when instantiating the Node Template. The default value of this attribute is 1.The value of minInstances MUST NOT be less than 0.
  - maxInstances: This attribute specifies the maximum number of instances that can be created when instantiating the Node Template. The default value of this attribute is 1. If the string is set to “unbounded”, an unbouded number of instances can be created. The value of maxInstances MUST be 1 or greater and MUST NOT be less than the value specified for minInstances.
- Notes
  - This element specifies a kind of a component making up the cloud application.

6.33 NodeType

- Relationships
  - NodeType **has 0..1** Documentation
  - NodeType **has 0..1** Tags
  - NodeType **has 0..1** DerivedFrom
  - NodeType **has 0..1** PropertiesDefinition
  - NodeType **has 0..1** RequirementDefinitions

    - NodeType **has** CapabilityDefinitions
    - NodeType **has 0..1** InstanceStates
    - NodeType **has 0..1** Interfaces
- Structure
    - name: This attribute specifies the name or identifier of the Node Type, which MUST be unique within the target namespace.
    - targetNamespace: This OPTIONAL attribute specifies the target namespace to which the definition of the Node Type will be added. If not specified, the Node Type definition will be added to the target namespace of the enclosing Definitions document.
    - abstract: This OPTIONAL attribute specifies that no instances can be created from Node Templates that use this Node Type as their type. If a Node Type includes a Requirement Definition or Capability Definition of an abstract Requirement Type or Capability Type, respectively, the Node Type MUST be declared as abstract as well. As a consequence, the corresponding abstract Node Type referenced by any Node Template has to be substituted by a Node Type derived from the abstract Node Type at the latest during the instantiation time of a Node Template. Note: an abstract Node Type MUST NOT be declared as final and vice versa.
    - final: This OPTIONAL attribute specifies that other Node Types MUST NOT be derived from this Node Type.
- Notes
    - This element specifies a type of Node that can be referenced as a type for Node Templates of a Service Template.

6.34 NodeTypeImplementation

- Relationships
    - NodeTypeImplementation **has 0..1** Documentation
    - NodeTypeImplementation **has 0..1** Tags
    - NodeTypeImplementation **has 0..1** RequiredContainerFeatures
    - NodeTypeImplementation **has 0..1** ImplementationArtifacts
    - NodeTypeImplementation **has 0..1** DeploymentArtifacts
- Structure
    - name: This attribute specifies the name or identifier of the Node Type Implementation, which MUST be unique within the target namespace.
    - targetNamespace: This OPTIONAL attribute specifies the target namespace to which the definition of the Node Type Implementation will be added. If not specified, the Node Type Implementation will be added to the target namespace of the enclosing Definitions document.
    - nodeType: The QName value of this attribute specifies the Node Type implemented by this Node Type Implementation.
    - abstract: This OPTIONAL attribute specifies that this Node Type Implementation cannot be used directly as an implementation for the Node Type specified in the nodeType attribute. Note: an abstract Node Type Implementation MUST NOT be declared as final

and vice versa.
    - final: This OPTIONAL attribute specifies that other Node Type Implementations MUST NOT be derived from this Node Type Implementation.
- Notes
    - This element specifies the implementation of the manageability behavior of a type of Node that can be referenced as a type for Node Templates of a Service Template.

6.35 NodeTypeReference

- Structure
    - typeRef: The attribute provides the QName of a Node Type to which the Policy Type applies.

6.36 Operation

- Relationships
    - Operation **has** NodeOperation **or** RelationshipOperation **or** Plan
- Structure
    - name: (i) This attribute specifies the name of the operation, which MUST be unique within the containing interface. (ii) This attribute defines the name of the operation and MUST be unique within the containing Interface of the Node Type.
- Notes
    - This element specifies one exposed operation of an interface exposed by the Service Template.
    - An operation exposed by a Service Template maps to an internal component of the Service Template which actually provides the operation: it can be mapped to an operation provided by a Node Template (i.e. an operation defined by the Node Type specified in the type attribute of the Node Template), it can be mapped to an operation provided by a Relationship Template (i.e. an operation defined by the Relationship Type specified in the type attribute of the Relationship Template), or it can be mapped to a Plan of the Service Template.
    - When an exposed operation is invoked on a service instance created from the Service Template, the operation or Plan mapped to the exposed operation will actually be invoked.
    - This element defines an operation available to manage particular aspects of the Node Type.

6.37 OutputParameter

- Structure
    - name: This attribute specifies the name of the output parameter, which MUST be unique within the set of output parameters defined for the operation.
    - type: This attribute specifies the type of the output parameter.
    - required: This OPTIONAL attribute specifies whether or not the output parameter is REQUIRED (required attribute with a value of “yes” – default) or OPTIONAL (required attribute with a value of “no”).

6.38 OutputParameters

- Relationships
  - OutputParameters **has 1..*** OutputParameter
- Notes
  - This OPTIONAL property contains a list of one or more output parameter definitions for the Plan, each defined in a nested, separate OutputParameter element.

6.39 Plan

- Relationships
  - Plan **has 0..1** Precondition
  - Plan **has 0..1** InputParameters
  - Plan **has 0..1** OutputParameters
  - Plan **has** PlanModel **or** PlanModelReference
- Structure
  - id: This attribute specifies the identifier of the Plan. The identifier of the Plan MUST be unique within the target namespace.
  - name: This OPTIONAL attribute specifies the name of the Plan.
  - planType: The value of the attribute specifies the type of the plan as an indication on what the effect of executing the plan on a service will have. The plan type is specified by means of a URI, allowing for an extensibility mechanism for authors of service templates to define new plan types over time.
  - planLanguage: This attribute denotes the process modeling language (or metamodel) used to specify the plan.
  - planRef
- Notes
  - This element specifies by means of its planRef attribute a reference to a Plan that provides the implementation of the operation exposed by the Service Template. One of NodeOperation, RelationshipOperation or Plan MUST be specified within the Operation element.
  - An instance of the Plan element MUST either contain the actual plan as instance of the PlanModel element, or point to the model via the PlanModelReference element.

6.40 Plans

- Relationships
  - Plans **has 1..*** Plan
- Notes
  - This element specifies the operational behavior of the service. A Plan contained in the Plans element can specify how to create, terminate or manage the service.

6.41 PlanModel

- Notes
  - contains the actual model content

6.42 PlanModelReference

- Structure
  - reference
- Notes
  - This property points to the model content. Its reference attribute contains a URI of the model of the plan.

6.43 Policies

- Relationships
  - Policies **has 1..*** Policy
- Notes
  - This OPTIONAL element specifies global policies of the Service Template related to a particular management aspect. All Policies defined within the Policies element MUST be enforced by a TOSCA implementation, i.e. Policies are AND-combined. Each policy is defined by a separate, nested Policy element.
  - This OPTIONAL element specifies policies associated with the Node Template. All Policies defined within the Policies element MUST be enforced by a TOSCA implementation, i.e. Policies are AND-combined. Each policy is specified by means of a separate nested Policy element.

6.44 Policy

- Structure
  - name: (i) This OPTIONAL attribute allows for the definition of a name for the Policy. If specified, this name MUST be unique within the containing Policies element. (ii) This OPTIONAL attribute allows for the definition of a name for the Policy. If specified, this name MUST be unique within the containing Policies element.
  - policyType: (i) This attribute specifies the type of this Policy. The QName value of this attribute SHOULD correspond to the QName of a PolicyType defined in the same Definitions document or in an imported document. The policyType attribute specifies the artifact type specific content of the Policy element body and indicates the type of Policy Template referenced by the Policy via the policyRef attribute. (ii) This attribute specifies the type of this Policy. The QName value of this attribute SHOULD correspond to the QName of a PolicyType defined in the same Definitions document or in an imported document. The policyType attribute specifies the artifact type specific content of the Policy element body and indicates the type of Policy Template referenced by the Policy via the policyRef attribute.
  - policyRef: (i) The QName value of this OPTIONAL attribute references a Policy Template that is associated to the Service Template. This Policy Template can be defined in the same TOSCA Definitions document, or it can be defined in a separate document that is imported into the current Definitions document. The type of Policy Template referenced by the policyRef attribute MUST be the same type or a sub-type of the type specified in the policyType attribute. (ii) The QName value of this OPTIONAL attribute

references a Policy Template that is associated to the Node Template. This Policy Template can be defined in the same TOSCA Definitions document, or it can be defined in a separate document that is imported into the current Definitions document. The type of Policy Template referenced by the policyRef attribute MUST be the same type or a sub-type of the type specified in the policyType attribute.

6.45 PolicyTemplate

- Relationships
  - PolicyTemplate **has 0..1** Documentation
  - PolicyTemplate **has 0..1** Properties
  - PolicyTemplate **has 0..1** PropertyConstraints
- Structure
  - id: This attribute specifies the identifier of the Policy Template which MUST be unique within the target namespace.
  - name: This OPTIONAL attribute specifies the name of the Policy Template.
  - type: The QName value of this attribute refers to the Policy Type providing the type of the Policy Template.
- Notes
  - This element specifies a template of a Policy that can be associated to Node Templates defined within a Service Template. Other than a Policy Type, a Policy Template can define concrete values for a policy according to the set of attributes specified by the Policy Type the Policy Template refers to.

6.46 PolicyType

- Relationships
  - PolicyType **has 0..1** Documentation
  - PolicyType **has 0..1** Tags
  - PolicyType **has 0..1** DerivedFrom
  - PolicyType **has 0..1** PropertiesDefinition
  - PolicyType **has 0..1** AppliesTo
- Structure
  - name: This attribute specifies the name or identifier of the Policy Type, which MUST be unique within the target namespace.
  - policyLanguage: This OPTIONAL attribute specifies the language used to specify the details of the Policy Type. These details can be defined as policy type specific content of the PolicyType element.
  - abstract: This OPTIONAL attribute specifies that no instances can be created from Policy Templates of that abstract Policy Type, i.e. the respective policies cannot be used directly during the instantiation of a Service Template.
  - final: This OPTIONAL attribute specifies that other Policy Types MUST NOT be derived from this Policy Type. Note: a final Policy Type MUST NOT be declared as abstract.

    - targetNamespace: This OPTIONAL attribute specifies the target namespace to which the definition of the Policy Type will be added. If not specified, the Policy Type definition will be added to the target namespace of the enclosing Definitions document.
- Notes
    - This element specifies a type of Policy that can be associated to Node Templates defined within a Service Template. For example, a scaling policy for nodes in a web server tier might be defined as a Policy Type, which specifies the attributes the scaling policy can have.

6.47 Precondition

- Structure
    - expressionLanguage
- Notes
    - This OPTIONAL element specifies a condition that needs to be satisfied in order for the plan to be executed. The expressionLanguage attribute of this element specifies the expression language the nested condition is provided in.

6.48 Properties

- Relationships
    - Properties **has 0..1** PropertyMappings
- Notes
    - This OPTIONAL element specifies global properties of the Service Template in the form of an XML fragment contained in the body of the Properties element. Those properties MAY be mapped to properties of components within the Service Template to make them visible to the outside.
    - Specifies initial values for one or more of the Node TypeProperties of the Node Type providing the property definitions in the concrete context of the Node Template. The initial values are specified by providing an instance document of the XML schema of the corresponding Node Type Properties. This instance document considers the inheritance structure deduced by the DerivedFrom property of the Node Type referenced by the type attribute of the Node Template. The instance document of the XML schema might not validate against the existence constraints of the corresponding schema: not all Node Type properties might have an initial value assigned, i.e. mandatory elements or attributes might be missing in the instance provided by the Properties element. Once the defined Node Template has been instantiated, any XML representation of the Node Type properties MUST validate according to the associated XML schema definition.
    - This element specifies initial values for one or more of the Requirement Properties according to the Requirement Type providing the property definitions. Properties are provided in the form of an XML fragment. The same rules as outlined for the Properties element of the Node Template apply.
    - This element specifies initial values for one or more of the Capability Properties according to the Capability Type providing the property definitions. Properties are

provided in the form of an XML fragment. The same rules as outlined for the Properties element of the Node Template apply.
    - Specifies initial values for one or more of the Relationship Type Properties of the Relationship Type providing the property definitions in the concrete context of the Relationship Template. The initial values are specified by providing an instance document of the XML schema of the corresponding Relationship Type Properties. This instance document considers the inheritance structure deduced by the DerivedFrom property of the Relationship Type referenced by the type attribute of the Relationship Template. The instance document of the XML schema might not validate against the existence constraints of the corresponding schema: not all Relationship Type properties might have an initial value assigned, i.e. mandatory elements or attributes might be missing in the instance provided by the Properties element. Once the defined Relationship Template has been instantiated, any XML representation of the Relationship Type properties MUST validate according to the associated XML schema definition.
    - This OPTIONAL element specifies the invariant properties of the Artifact Template, i.e. those properties that will be commonly used across different contexts in which the Artifact Template is used. The initial values are specified by providing an instance document of the XML schema of the corresponding Artifact Type Properties. This instance document considers the inheritance structure deduced by the DerivedFrom property of the Artifact Type referenced by the type attribute of the Artifact Template.
    - This OPTIONAL element specifies the invariant properties of the Policy Template, i.e. those properties that will be commonly used across different contexts in which the Policy Template is used. The initial values are specified by providing an instance document of the XML schema of the corresponding Policy Type Properties. This instance document considers the inheritance structure deduced by the DerivedFrom property of the Policy Type referenced by the type attribute of the Policy Template.

6.49 PropertiesDefinition

- Structure
    - element: (i) This attribute provides the QName of an XML element defining the structure of the Node Type Properties. (ii) This attribute provides the QName of an XML element defining the structure of the Relationship Type Properties. (iii) This attribute provides the QName of an XML element defining the structure of the Requirement Type Properties. (iv) This attribute provides the QName of an XML element defining the structure of the Capability Type Properties. (v) This attribute provides the QName of an XML element defining the structure of the Artifact Type Properties. (vi) This attribute provides the QName of an XML element defining the structure of the Policy Type Properties.
    - type: (i) This attribute provides the QName of an XML (complex) type defining the structure of the Node Type Properties. (ii) This attribute provides the QName of an XML (complex) type defining the structure of the Relationship Type Properties. (iii) This

element specifies the structure of the observable properties of the Requirement Type, such as its configuration and state, by means of XML schema. (iv) This attribute provides the QName of an XML (complex) type defining the structure of the Requirement Type Properties. (v) This attribute provides the QName of an XML (complex) type defining the structure of the Capability Type Properties. (vi) This attribute provides the QName of an XML (complex) type defining the structure of the Artifact Type Properties. (vii) This attribute provides the QName of an XML (complex) type defining the structure of the Policy Type Properties.

- Notes
  - This element specifies the structure of the observable properties of the Node Type, such as its configuration and state, by means of XML schema.
  - This element specifies the structure of the observable properties of the Relationship Type, such as its configuration and state, by means of XML schema.
  - This element specifies the structure of the observable properties of the Capability Type, such as its configuration and state, by means of XML schema.
  - This element specifies the structure of the observable properties of the Artifact Type, such as its configuration and state, by means of XML schema.
  - This element specifies the structure of the observable properties of the Policy Type by means of XML schema.

6.50 PropertyConstraint

- Structure
  - property: (i) This attribute identifies a property by means of an XPath expression to be evaluated on the XML fragment defining the Service Template's properties. (ii) The string value of this property is an XPath expression pointing to the property within the Node Type Properties document that is constrained within the context of the Node Template. More than one constraint MUST NOT be defined for each property. (iii) The string value of this property is an XPath expression pointing to the property within the Relationship Type Properties document that is constrained within the context of the Relationship Template. More than one constraint MUST NOT be defined for each property. (iv) The string value of this property is an XPath expression pointing to the property within the Artifact Type Properties document that is constrained within the context of the Artifact Template. More than one constraint MUST NOT be defined for each property. (v) The string value of this property is an XPath expression pointing to the property within the Policy Type Properties document that is constrained within the context of the Policy Template. More than one constraint MUST NOT be defined for each property.
  - constraintType: (i) This attribute specifies the type of constraint by means of a URI, which defines both the semantic meaning of the constraint as well as the format of the content. (ii) The constraint type is specified by means of a URI, which defines both the semantic meaning of the constraint as well as the format of the content. (iii) The constraint type is specified by means of a URI, which defines both the semantic meaning

of the constraint as well as the format of the content. (iv) The constraint type is specified by means of a URI, which defines both the semantic meaning of the constraint as well as the format of the content. (v) The constraint type is specified by means of a URI, which defines both the semantic meaning of the constraint as well as the format of the content.
    - The body of the PropertyConstraint element provides the actual constraint. Note: The body MAY be empty in case the constraintType URI already specifies the constraint appropriately. For example, a “read-only” constraint could be expressed solely by the constraintType URI.

6.51 PropertyConstraints

- Relationships
    - PropertyConstraints **has 1..*** PropertyConstraint
- Notes
    - This OPTIONAL element specifies constraints on one or more of the Service Template’s properties. Each constraint is specified by means of a separate, nested PropertyConstraint element.
    - Specifies constraints on the use of one or more of the Node Type Properties of the Node Type providing the property definitions for the Node Template. Each constraint is specified by means of a separate nested PropertyConstraint element.
    - This element specifies constraints on the use of one or more of the Properties of the Requirement Type providing the property definitions for the Requirement. Each constraint is specified by means of a separate nested PropertyConstraint element. The same rules as outlined for the PropertyConstraints element of the Node Template apply.
    - This element specifies constraints on the use of one or more of the Properties of the Capability Type providing the property definitions for the Capability. Each constraint is specified by means of a separate nested PropertyConstraint element. The same rules as outlined for the PropertyConstraints element of the Node Template apply.
    - Specifies constraints on the use of one or more of the Relationship Type Properties of the Relationship Type providing the property definitions for the Relationship Template. Each constraint is specified by means of a separate nested PropertyConstraint element.
    - This OPTIONAL element specifies constraints on the use of one or more of the Artifact Type Properties of the Artifact Type providing the property definitions for the Artifact Template. Each constraint is specified by means of a separate nested PropertyConstraint element.
- This OPTIONAL element specifies constraints on the use of one or more of the Policy Type Properties of the Policy Type providing the property definitions for the Policy Template. Each constraint is specified by means of a separate nested PropertyConstraint element.

6.52 PropertyMapping

- Structure
    - serviceTemplatePropertyRef: This attribute identifies a property of the Service Template by means of an XPath expression to be evaluated on the XML fragment defining the

Service Template's properties.
    - targetObjectRef: This attribute specifies the object that provides the property to which the respective Service Template property is mapped. The referenced target object MUST be one of Node Template, Requirement of a Node Template, Capability of a Node Template, or Relationship Template.
    - targetPropertyRef: This attribute identifies a property of the target object by means of an XPath expression to be evaluated on the XML fragment defining the target object's properties.

6.53 PropertyMappings

- Relationships
    - PropertyMappings **has 1..*** PropertyMapping
- Notes
    - This OPTIONAL element specifies mappings of one or more of the Service Template's properties to properties of components within the Service Template (e.g. Node Templates, Relationship Templates, etc.). Each property mapping is defined by a separate, nested PropertyMapping element.

6.54 RelationshipConstraint

- Structure
    - constraintType: This attribute specifies the type of relationship constraint by means of a URI. Depending on the type, the body of the RelationshipConstraint element might contain type specific content that further details the actual constraint.

6.55 RelationshipConstraints

- Relationships
    - RelationshipConstraints **has 1..*** RelationshipConstraint
- Notes
    - This element specifies a list of constraints on the use of the relationship in separate nested RelationshipConstraint elements.

6.56 RelationshipOperation

- Structure
    - relationshipRef: specifies a reference to the respective Relationship Template
    - interfaceName: specific interface to be mapped to the operation exposed by the Service Template
    - operationName: specific operation to be mapped to the operation exposed by the Service Template
- Notes
    - This element specifies a reference to an operation of a Relationship Template.

6.57 RelationshipTemplate

- Relationships

    - RelationshipTemplate **has 0..1** Properties
    - RelationshipTemplate **has** SourceElement
    - RelationshipTemplate **has** TargetElement
    - RelationshipTemplate **has 0..1** PropertyConstraints
    - RelationshipTemplate **has 0..1** RelationshipConstraints
- Structure
    - id: This attribute specifies the identifier of the Relationship Template. The identifier of the Relationship Template MUST be unique within the target namespace.
    - name: This OPTIONAL attribute specifies the name of the Relationship Template.
    - type: The QName value of this property refers to the Relationship Type providing the type of the Relationship Template.
- Notes
    - This element specifies a kind of relationship between the components of the cloud application. For each specified Relationship Template the source element and target element MUST be specified in the Topology Template.

6.58 RelationshipType

- Relationships
    - RelationshipType **has 0..1** Documentation
    - RelationshipType **has 0..1** Tags
    - RelationshipType **has 0..1** DerivedFrom
    - RelationshipType **has 0..1** PropertiesDefinition
    - RelationshipType **has 0..1** InstanceStates
    - RelationshipType **has 0..1** SourceInterfaces
    - RelationshipType **has 0..1** TargetInterfaces
    - RelationshipType **has 0..1** ValidSource
    - RelationshipType **has 0..1** ValidTarget
- Structure
    - name: This attribute specifies the name or identifier of the Relationship Type, which MUST be unique within the target namespace.
    - targetNamespace: This OPTIONAL attribute specifies the target namespace to which the definition of the Relationship Type will be added. If not specified, the Relationship Type definition will be added to the target namespace of the enclosing Definitions document.
    - abstract: This OPTIONAL attribute specifies that no instances can be created from Relationship Templates that use this Relationship Type as their type. Note: an abstract Relationship Type MUST NOT be declared as final and vice versa.
    - final: This OPTIONAL attribute specifies that other Relationship Types MUST NOT be derived from this Relationship Type.
- Notes
    - This element specifies a type of Relationship that can be referenced as a type for Relationship Templates of a Service Template.

6.59 RelationshipTypeImplementation

- Relationships
  - RelationshipTypeImplementation **has 0..1** Documentation
  - RelationshipTypeImplementation **has 0..1** Tags
  - RelationshipTypeImplementation **has 0..1** DerivedFrom
  - RelationshipTypeImplementation **has 0..1** RequiredContainerFeatures
  - RelationshipTypeImplementation **has 0..1** ImplementationArtifacts
- Structure
  - name: This attribute specifies the name or identifier of the Relationship Type Implementation, which MUST be unique within the target namespace.
  - targetNamespace: This OPTIONAL attribute specifies the target namespace to which the definition of the Relationship Type Implementation will be added. If not specified, the Relationship Type Implementation will be added to the target namespace of the enclosing Definitions document.
  - relationshipType: The QName value of this attribute specifies the Relationship Type implemented by this Relationship Type Implementation.
  - abstract: This OPTIONAL attribute specifies that this Relationship Type Implementation cannot be used directly as an implementation for the Relationship Type specified in the relationshipType attribute. Note: an abstract Relationship Type Implementation MUST NOT be declared as final and vice versa.
  - final : This OPTIONAL attribute specifies that other Relationship Type Implementations MUST NOT be derived from this Relationship Type Implementation.
- Notes
  - This element specifies the implementation of the manageability behavior of a type of Relationship that can be referenced as a type for Relationship Templates of a Service Template.

6.60 RequiredContainerFeature

- Structure
  - feature: The value of this attribute is a URI that denotes the corresponding needed feature of the environment.

6.61 RequiredContainerFeatures

- Relationships
  - RequiredContainerFeatures has 1..* RequiredContainerFeature
- Notes
  - An implementation of a Node Type might depend on certain features of the environment it is executed in, such as specific (potentially proprietary) APIs of the TOSCA container. For example, an implementation to deploy a virtual machine based on an image could require access to some API provided by a public cloud, while another implementation could require an API of a vendor-specific virtual image library. Thus, the contents of the RequiredContainerFeatures element provide “hints” to the TOSCA container allowing it

to select the appropriate Node Type Implementation if multiple alternatives are provided. Each such dependency is defined by a separate RequiredContainerFeature element.
    - An implementation of a Relationship Type might depend on certain features of the environment it is executed in, such as specific (potentially proprietary) APIs of the TOSCA container. Thus, the contents of the RequiredContainerFeatures element provide “hints” to the TOSCA container allowing it to select the appropriate Relationship Type Implementation if multiple alternatives are provided. Each such dependency is defined by a separate RequiredContainerFeature element.

6.62 Requirement

- Relationships
    - Requirement **has 0..1** Properties
    - Requirement **has 0..1** PropertyConstraints
- Structure
    - id: This attribute specifies the identifier of the Requirement. The identifier of the Requirement MUST be unique within the target namespace.
    - name: (i) This OPTIONAL attribute allows for specifying a name of the Requirement other than that specified by the referenced Requirement of a Node Template. (ii) This attribute specifies the name of the Requirement. The name and type of the Requirement MUST match the name and type of a Requirement Definition in the Node Type specified in the type attribute of the Node Template.
    - ref: This attribute references a Requirement element of a Node Template within the Service Template.
    - type: The QName value of this attribute refers to the Requirement Type definition of the Requirement. This Requirement Type denotes the semantics and well as potential properties of the Requirement.

6.63 RequirementDefinition

- Relationships
    - RequirementDefinition **has 0..1** Constraints
- Structure
    - name: This attribute specifies the name of the defined requirement and MUST be unique within the RequirementsDefinitions of the current Node Type.
    - requirementType: This attribute identifies by QName the Requirement Type that is being defined by the current RequirementDefinition.
    - lowerBound: This OPTIONAL attribute specifies the lower boundary by which a requirement MUST be matched for Node Templates according to the current Node Type, or for instances created for those Node Templates. The default value for this attribute is one. A value of zero would indicate that matching of the requirement is optional.
    - upperBound: This OPTIONAL attribute specifies the upper boundary by which a requirement MUST be matched for Node Templates according to the current Node Type, or for instances created for those Node Templates. The default value for this attribute is

one. A value of “unbounded” indicates that there is no upper boundary.

6.64 RequirementDefinitions

- Relationships
  - RequirementDefinitions **has 1..*** RequirementDefinition
- Notes
  - This OPTIONAL element specifies the requirements that the Node Type exposes (see section 3.4 for details). Each requirement is defined in a nested RequirementDefinition element.

6.65 RequirementType

- Relationships
  - RequirementType **has 0..1** Documentation
  - RequirementType **has 0..1** Tags
  - RequirementType **has 0..1** DerivedFrom
  - RequirementType **has 0..1** PropertiesDefinition
- Structure
  - name: This attribute specifies the name or identifier of the Requirement Type, which MUST be unique within the target namespace.
  - targetNamespace: This OPTIONAL attribute specifies the target namespace to which the definition of the Requirement Type will be added. If not specified, the Requirement Type definition will be added to the target namespace of the enclosing Definitions document.
  - abstract: This OPTIONAL attribute specifies that no instances can be created from Node Templates of a Node Type that defines a requirement of this Requirement Type. Note: an abstract Requirement Type MUST NOT be declared as final and vice versa.
  - final: This OPTIONAL attribute specifies that other Requirement Types MUST NOT be derived from this Requirement Type.
  - requiredCapabilityType: This OPTIONAL attribute specifies the type of capability needed to match the defined Requirement Type. The QName value of this attribute refers to the QName of a CapabilityType element defined in the same Definitions document or in a separate, imported document.
- Notes
  - This element specifies a type of Requirement that can be exposed by Node Types used in a Service Template.

6.66 Requirements

- Relationships
  - Requirements **has 1..*** Requirement
- Notes
  - This OPTIONAL element specifies Requirements exposed by the Service Template. Those Requirements correspond to Requirements of Node Templates within the Service Template that are propagated beyond the boundaries of the Service Template. Each Requirement is defined by a separate, nested Requirement element.

    - This element contains a list of requirements for the Node Template, according to the list of requirement definitions of the Node Type specified in the type attribute of the Node Template. Each requirement is specified in a separate nested Requirement element.

6.67 ServiceTemplate

- Relationships
    - ServiceTemplate **has 0..1** Documentation
    - ServiceTemplate **has 0..1** Tags
    - ServiceTemplate **has 0..1** BoundaryDefinitions
    - ServiceTemplate **has** TopologyTemplate
- Structure
    - id: This attribute specifies the identifier of the Service Template which MUST be unique within the target namespace.
    - name: This OPTIONAL attribute specifies a descriptive name of the Service Template.
    - targetNamespace: The value of this OPTIONAL attribute specifies the target namespace for the Service Template. If not specified, the Service Template will be added to the namespace declared by the targetNamespace attribute of the enclosing Definitions element.
    - substitutableNodeType: This OPTIONAL attribute specifies a Node Type that can be substituted by this Service Template. If another Service Template contains a Node Template of the specified Node Type (or any Node Type this Node Type is derived from), this Node Template can be substituted by an instance of this Service Template that then provides the functionality of the substituted node. See section 3.5 for more details.
- Notes
    - This element specifies a complete Service Template for a cloud application. A Service Template contains a definition of the Topology Template of the cloud application, as well as any number of Plans. Within the Service Template, any type definitions (e.g. Node Types, Relationship Types, etc.) defined in the same Definitions document or in imported Definitions document can be used.

6.68 SourceInterfaces

- Relationships
    - SourceInterfaces **has 1..*** Interface
- Notes
    - This OPTIONAL element contains definitions of manageability interfaces that can be performed on the source of a relationship of this Relationship Type to actually establish the relationship between the source and the target in the deployed service. Those interface definitions are contained in nested Interface elements, the content of which is that described for Node Type interfaces (see section 6.2).

6.69 SourceElement

- Structure

    - ref: This attribute references by ID a Node Template or a Requirement of a Node Template within the same Service Template document that is the source of the Relationship Template.
- Notes
    - This element specifies the origin of the relationship represented by the current Relationship Template.

6.70 Tag

- Structure
    - name: This attribute specifies the name of the tag.
    - value: This attribute specifies the value of the tag.

6.71 Tags

- Relationships
    - Tags **has 1..*** Tag
- Notes
    - This OPTIONAL element allows the definition of any number of tags which can be used by the author to describe the Service Template. Each tag is defined by a separate, nested Tag element.
    - This OPTIONAL element allows the definition of any number of tags which can be used by the author to describe the Node Type. Each tag is defined by a separate, nested Tag element.
    - This OPTIONAL element allows the definition of any number of tags which can be used by the author to describe the Node Type Implementation. Each tag is defined by a separate, nested Tag element.
    - This OPTIONAL element allows the definition of any number of tags which can be used by the author to describe the Relationship Type. Each tag is defined by a separate, nested Tag element.
    - This OPTIONAL element allows the definition of any number of tags which can be used by the author to describe the Relationship Type Implementation. Each tag is defined by a separate, nested Tag element.
    - This OPTIONAL element allows the definition of any number of tags which can be used by the author to describe the Requirement Type. Each tag is defined by a separate, nested Tag element.
    - This OPTIONAL element allows the definition of any number of tags which can be used by the author to describe the Capability Type. Each tag is defined by a separate, nested Tag element.
    - This OPTIONAL element allows the definition of any number of tags which can be used by the author to describe the Artifact Type. Each tag is defined by a separate, nested Tag element.
    - This OPTIONAL element allows the definition of any number of tags which can be used by the author to describe the Policy Type. Each tag is defined by a separate, nested Tag

element.

6.72 TargetElement

- Structure
  - ref: This attribute references by ID a Node Template or a Capability of a Node Template within the same Service Template document that is the target of the Relationship Template.
- Notes
  - This element specifies the target of the relationship represented by the current Relationship Template.

6.73 TargetInterfaces

- Relationships
  - TargetInterfaces **has 1..*** Interface
- Notes
  - This OPTIONAL element contains definitions of manageability interfaces that can be performed on the target of a relationship of this Relationship Type to actually establish the relationship between the source and the target in the deployed service. Those interface definitions are contained in nested Interface elements, the content of which is that described for Node Type interfaces (see section 6.2).

6.74 TopologyTemplate

- Relationships
  - TopologyTemplate **has at least one** NodeTemplate **or** RelationshipTemplate
  - TopologyTemplate **has 0..1** Plans
- Notes
  - This element specifies the overall structure of the cloud application defined by the Service Template, i.e. the components it consists of, and the relations between those components. The components of a service are referred to as Node Templates, the relations between the components are referred to as Relationship Templates.

6.75 Types

- Relationships
  - Types **has 0..1** Documentation
- Notes
  - This element specifies XML definitions introduced within the Definitions document. Such definitions are provided within one or more separate Schema definitions (usually xs:schema elements). The Types element defines XML definitions within a Definitions document without having to define these XML definitions in separate files and importing them. Note, that an xs:schema element nested in the Types element MUST be a valid XML schema definition. In case the targetNamespace attribute of a nested xs:schema element is not specified, all definitions within this element become part of the target namespace of the encompassing Definitions element.

6.76 ValidSource

- Structure
  - typeRef: This attribute specifies the QName of a Node Type or Requirement Type that is allowed as a valid source for relationships defined using the Relationship Type under definition. Node Types or Requirements Types derived from the specified Node Type or Requirement Type, respectively, MUST also be accepted as valid relationship source.
- Notes
  - This OPTIONAL element specifies the type of object that is allowed as a valid origin for relationships defined using the Relationship Type under definition. If not specified, any Node Type is allowed to be the origin of the relationship.

6.77 ValidTarget

- Structure
  - typeRef: This attribute specifies the QName of a Node Type or Capability Type that is allowed as a valid target for relationships defined using the Relationship Type under definition. Node Types or Capability Types derived from the specified Node Type or Capability Type, respectively, MUST also be accepted as valid targets of relationships.
- Notes
  - This OPTIONAL element specifies the type of object that is allowed as a valid target for relationships defined using the Relationship Type under definition. If not specified, any Node Type is allowed to be the origin of the relationship.

References

[1] Gorka Benguria, Xabier Larrucea, Brian Elvesæter, Tor Neple, Anthony Beardsmore, and Michael Friess. 2007. A platform independent model for service oriented architectures. In *Enterprise Interoperability*. Guy Doumeingts, Jörg Müller, Gérard Morel, and Bruno Vallespir, (Eds.) Springer, 23–32

[2] Methodologies Corporation. 2011. Service-oriented modeling framework (SOMF) Version 2.1.

[3] The Open Group. 2011. SOA Reference Architecture. C119

[4] OASIS. 2012. Reference Architecture Foundation for Service Oriented Architecture Version 1.0. Standard OASIS Committee Specification 01. Organization for the Advancement of Structured Information Standards

[5] OMG. 2012. Service oriented architecture Modeling Language (SoaML) Specification Version 1.0.1. Standard. Object Management Group

[6] OASIS. 2013. Topology and Orchestration Specification for Cloud Applications Version 1.0. Standard. Organization for the Advancement of Structured Information Standards