INTERNATIONAL STANDARD

ISO/IEC 18384-3

First edition
2016-07-01

Information technology — Reference Architecture for Service Oriented Architecture (SOA RA) —

Part 3: **Service Oriented Architecture ontology**

Technologie de l'information — Architecture de référence pour l'architecture orientée service (SOA RA) —

Partie 3: Ontologie de l'architecture orientée service

Reference number
ISO/IEC 18384-3:2016(E)

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation on the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the WTO principles in the Technical Barriers to Trade (TBT) see the following URL: Foreword — Supplementary information

The committee responsible for this document is ISO/IEC JTC 1, *Information technology*, Subcommittee SC 38, *Cloud Computing and Distributed Platforms*.

ISO/IEC 18384 consists of the following parts, under the general title *Reference Architecture for Service Oriented Architecture (SOA RA)*:

— *Part 1: Terminology and concepts for SOA*

— *Part 2: Reference Architecture for SOA Solutions*

— *Part 3: Service Oriented Architecture Ontology*

Introduction

Service oriented architecture (SOA) is an architectural style in which business and IT systems are designed in terms of services available at an interface and the outcomes of these services. A service is a logical representation of a set of activities that has specified outcomes, is self-contained, it may be composed of other services but consumers of the service need not be aware of any internal structure.

SOA takes “service” as its basic element to constitute and integrate information systems so that they are suitable for a variety of solution requirements. SOA enables interactions between businesses without needing to specify aspects of any particular business domain. Using the SOA architectural style can improve the efficiency of developing information systems and integrating and reusing IT resources. In addition, using the SOA architectural style can help enable rapid response of information systems to ever-changing business needs.

This International Standard is intended to be a single set of SOA technical principles, specific norms, and standards for the world-wide market to help remove confusion about SOA and improve the standardization and quality of solutions.

This International Standard defines the terminology, technical principles, reference architecture and the ontology for SOA. ISO/IEC 18384 can be used to introduce SOA concepts, as a guide to the development and management of SOA solutions, as well as be referenced by business and industry standards.

This International Standard contains three parts:

1) ISO/IEC 18384-1 which defines the terminology, basic technical principles and concepts for SOA.

2) ISO/IEC 18384-2 which defines the detailed SOA reference architecture layers, including a metamodel, capabilities, architectural building blocks, as well as types of services in SOA solutions.

3) ISO/IEC 18384-3 which defines the core concepts of SOA and their relationships in the Ontology.

The targeted audience of this International Standard includes, but is not limited to, standards organizations, architects, architecture methodologists, system and software designers, business people, SOA service providers, SOA solution and service developers, and SOA service consumers who are interested in adopting and developing SOA.

Users of this International Standard will find it useful to read ISO/IEC 18384-1 for an understanding of SOA basics. ISO/IEC 18384-1 should be read before reading or applying ISO/IEC 18384-2. For those new to the SOA reference architecture in ISO/IEC 18384-2:2016, Clause 4 provides a high level understanding of the reference architecture for SOA solutions. The remaining clauses provide comprehensive details of the architectural building blocks and tradeoffs needed for a SOA Solution. This part of ISO/IEC 18384 contains the SOA Ontology, which is a formalism of the core concepts and terminology of SOA, with mappings to both UML and OWL. The SOA Ontology can be used independent of or in conjunction with ISO/IEC 18384-1 and ISO/IEC 18384-2.

The purpose of this part of ISO/IEC 18384 is to contribute to developing and fostering common understanding of service-oriented architecture (SOA) in order to improve alignment between the business and information technology communities and facilitate SOA adoption.

The SOA Ontology defines the concepts, terminology, and semantics of SOA in both business and technical terms, in order to

— create a foundation for further work in domain-specific areas,

— enable communications between business and technical people,

— enhance the understanding of SOA concepts in the business and technical communities,

— provide a means to state problems and opportunities clearly and unambiguously to promote mutual understanding, and

— provide a starting point for model-driven development of SOA solutions.

Information technology — Reference Architecture for Service Oriented Architecture (SOA RA) —

Part 3: Service Oriented Architecture ontology

1 Scope

This part of ISO/IEC 18384 defines a formal ontology for service-oriented architecture (SOA), an architectural style that supports service orientation. The terms defined in this ontology are key terms from the vocabulary in ISO/IEC 18384-1.

2 Normative references

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 18384-1, *Information technology — Reference Architecture for Service Oriented Architecture (SOA RA) — Part 1 Terminology and concepts for SOA*

3 Terms, definitions and abbreviated terms

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 18384-1 and the following apply.

3.1.1
opaque
having no internal structure that is visible to an external observer

3.1.2
ontology
model that represents a domain and is used to reason about the objects in that domain and the relations between them

Note 1 to entry: This part of ISO/IEC 18384 is high level and not meant to be used for formal reasoning.

[SOURCE: ISO/IEC/TR 24800-1:2007, 2.1.9]

3.2 Abbreviated terms

For the purposes of this document, the following abbreviated terms apply.

ABB	Architecture Building Block
BPMN	Business Process Model and Notation
EA	Enterprise Architecture
ESB	Enterprise Service Bus
IT	Information Technology

OWL	Web Ontology Language
RA	Reference Architecture
RDF	Resource Definition Framework
SLA	Service Level Agreement
SOA	Service Oriented Architecture
UML	Unified Modeling Language

4 Notations

The ontology is represented in the web ontology language (OWL) defined by the World Wide Web Consortium. OWL has three increasingly expressive sub-languages: OWL-Lite, OWL-DL, and OWL-Full (see Reference [10] for a definition of these three dialects of OWL). This ontology uses OWL-DL, the sub-language that provides the greatest expressiveness possible while retaining computational completeness and decidability.

The ontology contains classes and properties corresponding to the concepts of SOA. The formal OWL definitions are supplemented by natural language descriptions of the concepts, with graphic illustrations of the relations between them, and with examples of their use. For purposes of exposition, the ontology also includes UML (see Reference [8]) diagrams that graphically illustrate its classes and properties of the ontology. The natural language and OWL definitions contained in this part of ISO/IEC 18384 constitute the authoritative definition of the ontology; the diagrams are for explanatory purposes only. Some of the natural language terms used to describe the concepts are not formally represented in the ontology; those terms are meant in their natural language sense.

The availability of an OWL expression a standard RDF format allows easy loading into tools for architects and developers and allows validation.

This part of ISO/IEC 18384 uses examples to illustrate the ontology. One of these, the car-wash example, is used consistently throughout to illustrate the main concepts (see Annex A for the complete example). Other examples are used ad hoc in individual clauses to illustrate particular points.

5 Conventions

Bold font is used for OWL class, property, and instance names where they appear in clause text.

Italic strings are used for emphasis and to identify the first instance of a word requiring definition.

OWL definitions and syntax are shown in fixed-width font.

An unlabeled arrow in the illustrative UML diagrams means subclass.

The examples in this part of ISO/IEC 18384 are strictly informative and are for illustrative purposes.

6 Conformance

ISO/IEC 18384 contains three parts which have different conformance requirements:

1. terminology and concepts — conformance only to terms and adherence to the semantics in the definitions;
2. reference architecture for SOA solutions — conformance only to semantics of the metamodel and any Layers, ABBs, or capabilities that are used;
3. SOA Ontology — conformance for OWL or non-OWL applications.

Conformance to this part of ISO/IEC 18384 is defined as follows.

There are two kinds of applications that may conform to this ontology. One is the OWL-based ontologies (typically extensions of the SOA ontology); the other is a non-OWL application, such as a meta-model or a piece of software (see Clause 2 for the OWL version that is required).

A conforming OWL application (derived OWL-based ontology)

— shall conform to the OWL standard specified in Clause 2,

— shall include the whole of the ontology contained in Annex C,

— may add other OWL constructs, including class and property definitions, and

— may import other ontologies in addition to the SOA ontology.

This part of ISO/IEC 18384 does not use any OWL 2 (see Reference [15]) constructs; however, conforming applications may choose to use OWL or OWL 2.

A conforming non-OWL application

— shall include a defined and consistent transformation (at least semantic mapping) to a non-trivial subset of the ontology contained in Annex C,

— may add other constructs, including class and property definitions, and

— may import and/or use other ontologies in addition to the SOA ontology.

7 SOA Ontology Overview

7.1 At a Glance

A graphically compressed visualization of the entire ontology is shown in Figure 1.

The concepts illustrated in Figure 1 are described in the body.

This part of ISO/IEC 18384 starts by explaining the most basic foundational concept of elements and systems followed by explaining the elements of SOA human actor and task and then service concepts and descriptions and contracts for services and building on that to explain compositions of services. Finally, this part of ISO/IEC 18384 wraps up with Policies and Events which are relevant to all of the elements of SOA.

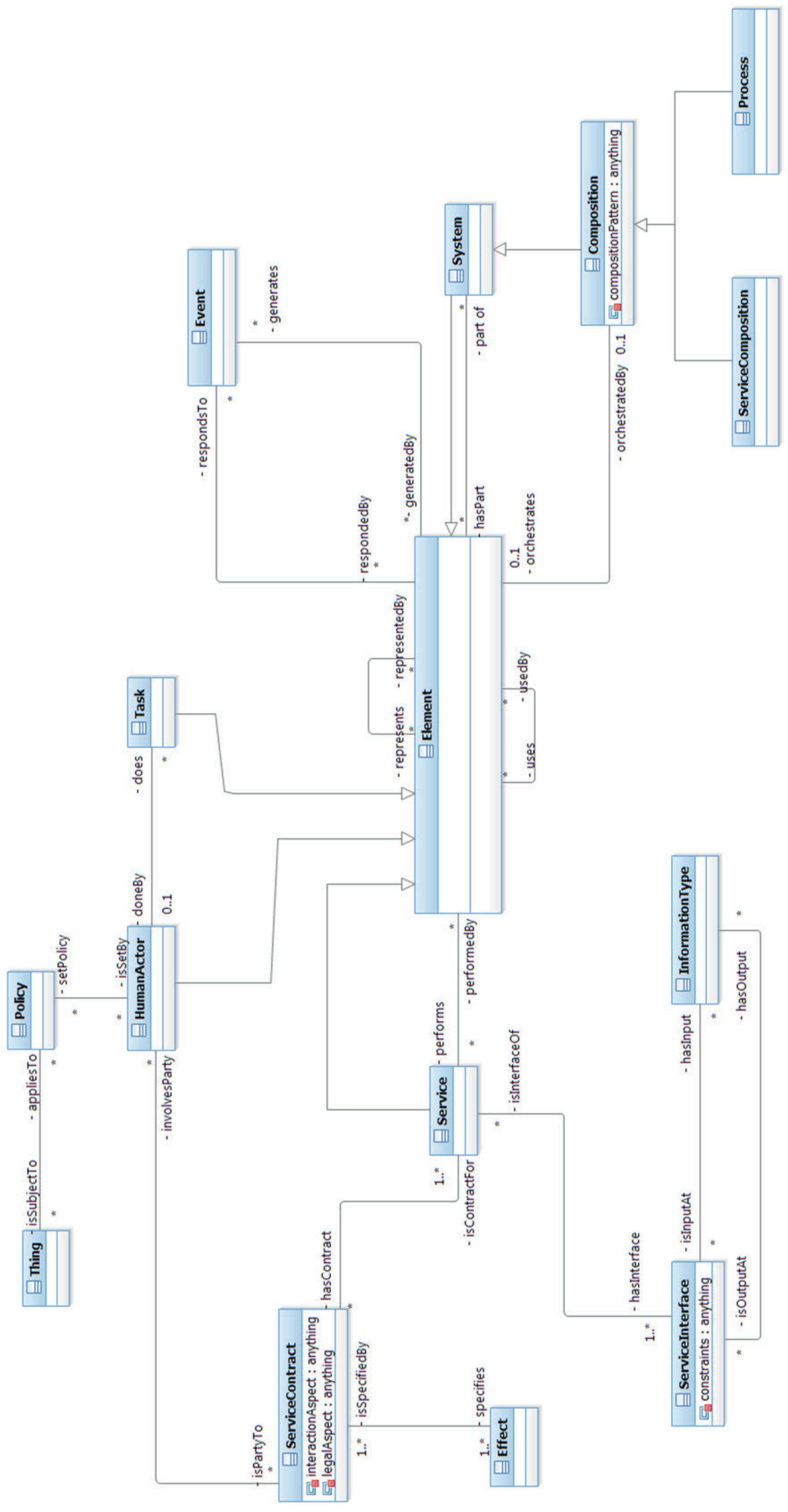

Figure 1 — SOA Ontology — Graphical Overview

7.2 Intended Use

This Clause describes caveats and assumptions for how this ontology should be interpreted.

— This ontology is intended for high level representation of concepts and is not intended for formal reasoning.

— This part of ISO/IEC 18384 is designed for use by business people, architects and systems and software designers to enable communications between business and technical people.

— This part of ISO/IEC 18384 focuses on a minimal set of SOA terms, modelling those terms in detail.

— This part of ISO/IEC 18384 explains relationships to other important concepts, but not at the same level of detail as the SOA terms. For example, policy is modelled, but not in great detail.

— This part of ISO/IEC 18384 restricts itself to OWL constructs, not using those introduced in OWL 2 (see Reference [15]), because the OWL constructs are sufficient for the scope of this part of ISO/IEC 18384. It is consistent with OWL 2 and does not preclude others from using it with OWL 2.

— This part of ISO/IEC 18384 elaborates on the SOA terms and relationships in ISO/IEC 18384-1 and ISO/IEC 18384-2. A separate metamodel in ISO/IEC 18384-2 provides the basis for the modeling in ISO/IEC 18384-2 and is used to describe and understand the reference architecture.

— This part of ISO/IEC 18384 defines the concepts, terminology, and semantics of SOA in both business and technical terms, in order to create a foundation for further work in domain-specific areas.

— This part of ISO/IEC 18384 provides a means to state problems and opportunities clearly and unambiguously to promote mutual understanding.

— This part of ISO/IEC 18384 may provide a starting point for model-driven development of SOA solutions.

7.3 Applications

The SOA ontology was developed in order to aid understanding and can simply be read.

It can also be used as a starting point for model-driven development, by applying it to particular usage domains and applications.

The ontology is applied to a particular usage domain by adding SOA OWL class instances of things in that domain. This is sometimes referred to as “populating the ontology.” In addition, an application can add definitions of new classes and properties, can import other ontologies, and can import the ontology OWL representation into other ontologies.

The ontology defines the relations between terms, but does not prescribe exactly how they should be applied. For explanations of what ontologies are and why they are needed, see References [11] and [14]. The examples provided in this part of ISO/IEC 18384 are describing one way in which the ontology could be applied in practical situations. Different applications of the ontology to the same situations would nevertheless be possible. The precise instantiation of the ontology in particular practical situations is a matter for users of the ontology, as long as the concepts and constraints defined by the ontology are correctly applied, the instantiation is valid.

8 System and Element

8.1 Overview

System and *element* are two of the concepts of this ontology. Both are concepts that are often used by practitioners, including the notion that systems have elements and that systems can be hierarchically combined (systems of systems). What differs from domain to domain is the specific nature of systems and elements, for instance, an electrical system has very different kinds of elements than an SOA system.

In the ontology, only elements and systems within the SOA domain are considered. Some SOA sub-domains use the term *component* rather than the term element. This is not contradictory, as any component of an SOA system is also an element of that (composite) system.

This Clause describes the following classes of the ontology:

Element

System

In addition, it defines the following properties:

uses and **usedBy**

represents and **representedBy**

8.2 The Element Class

```
<owl:Class rdf:about="#Element">
</owl:Class>
```

An *element* is an entity that is opaque and indivisible at a given level of abstraction. The element has a clearly defined boundary. The concept of element is captured by the **Element** OWL class, which is illustrated in Figure 2.

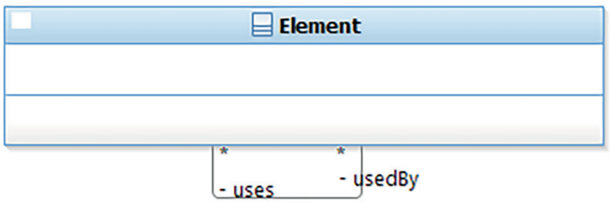

Figure 2 — The Element Class

In the context of the SOA ontology, only functional elements that belong to the SOA domain are considered in detail. There are other kinds of Elements than members of the four named subclasses (System, HumanActor, Task, and Service) described later in this ontology. Examples of such other kinds of Elements are things like software components or technology components (such as Enterprise Service Bus (ESB) implementations, etc.).

8.3 The uses and usedBy Properties

```
<owl:ObjectProperty rdf:about="#uses">
  <rdfs:domain rdf:resource="#Element"/>
  <rdfs:range rdf:resource="#Element"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#usedBy">
  <owl:inverseOf>
   <owl:ObjectProperty rdf:about="#uses"/>
  </owl:inverseOf>
</owl:ObjectProperty>
```

Elements may use other elements in various ways. In general, the notion of some element using another element is applied by practitioners for all of models, executables, and physical objects. What differs from domain to domain is the way in which such use is perceived.

An element uses another element if it interacts with it in some fashion. Interacts here is interpreted very broadly ranging through, for example, an element simply being a member of (used by) some system (see later for a formal definition of the **System** class), an element interacting with (using) another element (such as a service; see later for a formal definition of the **Service** class) in an *ad hoc* fashion, or even a strongly coupled dependency in a composition (see later for a formal definition of the **Composition** class). The **uses** property, and its inverse **usedBy**, capture the abstract notion of an element using another. These properties capture not just transient relations. Instantiations of the property can include “uses at this instant”, “has used”, and “may in future use”.

For the purposes of this ontology, the multitude of different possible semantics of a *uses* relationship is not enumerated and formally defined .The semantic interpretations are left to a particular sub-domain, application or even design approach.

8.4 Element — Organizational Example

Using an organizational example, typical instances of **Element** are organizational units and people. Whether to perceive a given part of an organization as an organizational unit or as the set of people within that organizational unit is an important choice of abstraction level.

Inside the boundary of the organizational unit, as the organizational unit can in fact use the people that are members of it. Note that the same person can in fact be a member of (be used by) multiple organizational units.

Outside the boundary the internal structure of an organizational unit remains opaque to an external observer, as the enterprise wants to be able to change the people within the organizational unit without having to change the definition of the organizational unit itself.

This simple example expresses that some elements have an internal structure. In fact, from an internal perspective they are an organized collection of other simpler things (captured by the System class defined in 8.5).

8.5 The System Class

```
<owl:Class rdf:about="#System">
  <owl:disjointWith>
   <owl:Class rdf:about="#Task"/>
  </owl:disjointWith>
  <owl:disjointWith>
   <owl:Class rdf:about="#Service"/>
  </owl:disjointWith>
  <rdfs:subClassOf>
   <owl:Class rdf:about="#Element"/>
  </rdfs:subClassOf>
</owl:Class>
```

A *system* is an organized collection of other things. Specifically, things in a system collection are instances of **Element**, each such instance being used by the system. The concept of *system* is captured by the **System** OWL class, which is illustrated in Figure 3.

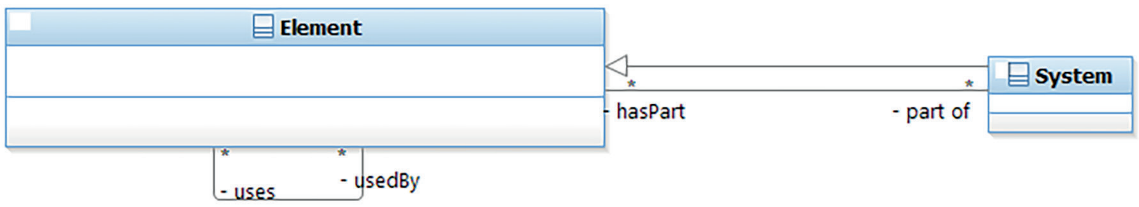

Figure 3 — The System Class

This definition of System is heavily influenced by ISO/IEC 42010:2011 (see Reference [13]).

In the context of the SOA ontology, only functional systems that belong to the SOA domain are considered in detail. Note that a fully described instance of **System** should have by its nature (as a collection) a *part of* relationship to at least one instance of **Element**.

Since System is a subclass of **Element**, all systems have a boundary and are opaque to an external observer (black box view). This excludes from the **System** class structures that have no defined boundary. From an SOA perspective, this is not really a loss since all interesting SOA systems do have the characteristic of being possible to perceive from an outside (consumer) perspective. Furthermore, having **System** as a subclass of **Element** allows us to naturally express the notion of systems of systems — the lower-level systems are simply elements used by the higher level system.

At the same time as supporting an external view point (black box view), all systems also support an internal view point (white box view) expressing how they are an organized collection. As an example, for the notion of a service this would typically correspond to a service specification view *versus* a service realization view (similar to the way that SoaML[9] defines services as having both a black box/specification part and a white box/realization part).

It is important to realize that even though systems using elements express an important aspect of the **uses** property, it is not necessary to "invent" a system just to express that some element uses another. In fact, even for systems it may be necessary to to express that they can use elements outside their own boundary — though this in many cases will preferably be expressed not at the system level, but rather by an element of the system using that external **Element** instance.

System is defined as disjoint with the *Service* and *Task* classes. Instances of these classes are considered not to be collections of other things. **System** is specifically not defined as disjoint with the *HumanActor* class since an organization is many cases in fact just a particular kind of system. A special intersection class to represent this fact is not defined.

8.6 System — Examples

8.6.1 Organizational Example

Continuing the organizational example from 8.5, an organizational unit can now be expressed as an instance of **System** has the people in it as members (and instances of element).

8.6.2 Service composition Example

Using a service composition example, services A and B are instances of **Element** and the composition of A and B is an instance of **System** (that uses A and B). It is important to realize that the act of composing is different than composition as a thing — it is in the latter sense that the term composition is used here.

See also Clause 11 for a formal definition of the concepts of service and service composition (and a repeat of the example in that more precise context).

8.6.3 Car wash Example

Consider a car wash business. The company as a whole is an organizational unit and can be instantiated in the ontology in the following way:

— *CarWashBusiness* is an instance of **System**.

— *Joe* (the owner) is an instance of **Element** and used by (owner of) *CarWashBusiness*.

— *Mary* (the secretary) is an instance of **Element** and used by (employee of) *CarWashBusiness*.

— *John* (the pre-wash guy) is an instance of **Element** and used by (employee of) *CarWashBusiness*.

— *Jack* (the washing manager and operator) is an instance of **Element** and used by (employee of) *CarWashBusiness*.

8.7 The represents and representedBy Properties

```
<owl:ObjectProperty rdf:about="#represents">
  <rdfs:domain rdf:resource="#Element"/>
  <rdfs:range rdf:resource="#Element"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#representedBy">
  <owl:inverseOf>
   <owl:ObjectProperty rdf:about="#represents"/>
  </owl:inverseOf>
</owl:ObjectProperty>
```

The environment described by an SOA is intrinsically hierarchically composite (see also 11.2 for a definition of the *Composition* class); in other words, the elements of SOA systems can be repeatedly composed to ever higher levels of abstraction. One aspect of this has already been addressed by the **uses** and **usedBy** properties in that the notion of systems of systems can be expressed. This is still a very concrete relationship though, and does not express the concept of architectural abstraction. The need for architectural abstraction is found in various places, such as a role representing the people playing that role, an organizational unit representing the people within it (subtly different from that same organizational unit using the people within it, as the **represents** relationship indicates the organizational unit as a substitute interaction point), an architectural building block representing an underlying construct (for instance, important to enterprise architects wanting to explicitly distinguish between constructs and building blocks), and an enterprise service bus (ESB) representing the services that are accessible through it (for instance, relevant when explicitly modelling operational interaction and dependencies). The concept of such an explicitly changing view point, or level of abstraction, is captured by the *represents* and *representedBy* properties illustrated in Figure 4.

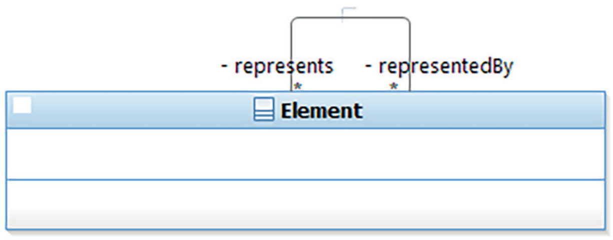

Figure 4 — The represents and representedBy Properties

It is important to understand the exact nature of the distinction between using an element (E1) and using another element (E2) that represents E1. If E1 changes, then anyone using E1 directly would experience a change, but someone using E2 would not experience any change.

When applying the architectural abstraction via the **represents** property there are three different architectural choices that can be made:

An element represents another element in a very literal way, simply by hiding the existence of that element and any changes to it. There will be a one-to-one relationship between the instance of **Element** and the (different) instance of **Element** that it represents. A simple real-world example is the notion of a broker acting as an intermediary between a seller (that does not wish to be known) and a buyer.

An element represents a particular aspect of another element. There will be a many-to-one relationship between many instances of **Element** (each of which represents a different aspect), and one (different) instance of **Element**. A simple real-world example is the notion that the same person can play (be represented by) many different roles.

An element is an abstraction that can represent many other elements. There will be a one-to-many relationship between one instance of **Element** (as an abstraction) and many other instances of **Element**. A simple real-world example is the notion of an architectural blueprint representing an abstraction of many different buildings being built according to that blueprint.

Note that in most cases, an instance of **Element** will represent only one kind of thing. Specifically, an instance of **Element** will typically represent instances of at most one of the classes **System**, **Service**, Human **Actor**, and **Task** (with the exception of the case where the same thing is both an instance of **System** and an instance of **Actor**). See later clauses for the definitions of **Service**, Human **Actor**, and **Task**.

8.8 The represents and representedBy Examples

8.8.1 Organizational Example

Expanding further on the organizational example, assume that a company desires to form a new organizational unit O1. There are two ways of doing this.

Define the new organization directly as a collection of people P1, P2, P3, and P4. This means that the new organization is perceived to be a leaf in the organizational hierarchy, and that any exchange of personnel means that its definition needs to change.

Define the new organization as a higher-level organizational construct, joining together two existing organizations O3 and O4. Coincidentally, O3 and O4 between them may have the same four people P1, P2, P3, and P4, but the new organization really doesn't know, and any member of O3 or O4 can be changed without needing to change the definition of the new organization. Furthermore, any member of O3 is intrinsically *not* working in the same organization as the members of O4 (in fact need not even be aware of them) — contrary to the first option where P1, P2, P3, and P4 are all colleagues in the same new organization.

In this way, the abstraction aspect of the **represents** property induces an important difference in the semantics of the collection defining the new organization. Any instantiation of the ontology can and should use the **represents** and **representedBy** properties to crisply define the implied semantics and lines of visibility/change.

8.8.2 Car Wash Example

Joe chooses to organize his business into two organizational units, one for the administration and one for the actual washing of cars. This can be instantiated in the ontology in the following way:

— *CarWashBusiness* is an instance of **System**.

— *AdministrativeSystem* is an instance of **System**.

— *Administration* is an instance of **Element** that represents *AdministrativeSystem* (the organizational unit aspect is opaque, *aka* ignoring anything else about *AdministrativeSystem*).

— *CarwashBusiness* uses (has organizational unit) *Administration*.

— *CarWashSystem* is an instance of **System**.

— *CarWash* is an instance of **Element** that represents *CarWashSystem* (the organizational unit aspect is opaque, *aka* ignoring anything else about *CarWashSystem*).

— *CarWash* is a member of *CarWashBusiness*.

— *Joe* (the owner) is an instance of **Element** and now used by *AdministrationSystem*.

— *Mary* (the secretary) is an instance of **Element** and now used by *AdministrationSystem*.

— *John* (the pre-wash guy) is an instance of **Element** and now used by *CarWashSystem*.

— *Jack* (the wash manager and operator) is an instance of **Element** and now used by *CarWashSystem*.

9 HumanActor and Task

9.1 Overview

People, organizations, and the things they do are important aspects of SOA systems. *HumanActor* and *Task* capture this as another set of core concepts of the ontology. Both are concepts that are generic and have relevance outside the domain of SOA. For the purposes of this SOA ontology, specific scope is given in that tasks are intrinsically atomic [corresponding to, for instance, the business process modeling notation (BPMN) 2.0 definition of Task (see Reference [4])] and human actors are restricted to people and organizations.

This Clause describes the following classes of the ontology:

HumanActor

Task

In addition, it defines the following properties:

does and **doneBy**

9.2 The HumanActor Class

```
<owl:Class rdf:about="#HumanActor">
  <rdfs:subClassOf>
   <owl:Class rdf:about="#Element"/>
  </rdfs:subClassOf>
  <owl:disjointWith>
   <owl:Class rdf:about="#Task"/>
  </owl:disjointWith>
  <owl:disjointWith>
  <owl:Class rdf:about="#Service"/>
   </owl:disjointWith>
  </owl:Class>

<owl:Class rdf:about="#ServiceContract"/>
  </owl:disjointWith>
  <owl:disjointWith>
   <owl:Class rdf:about="#ServiceInterface"/>
  </owl:disjointWith>
</owl:Class>
```

A *human actor* is a person or an organization. The concept of *human actor* is captured by the **HumanActor** OWL class, which is illustrated in Figure 5.

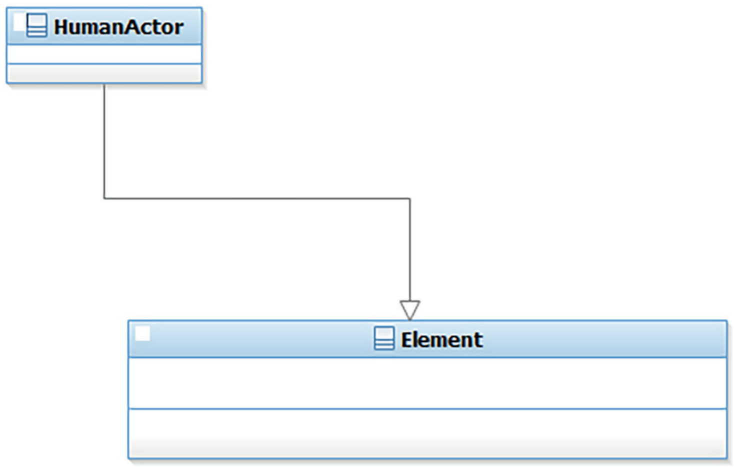

Figure 5 — The HumanActor Class

HumanActor is defined as disjoint with the *Service* and *Task* classes. Instances of these classes are considered not to be people or organizations. **HumanActor** is specifically not defined as disjoint with **System** since an organization in many cases is in fact just a particular kind of system. A special intersection class to represent this fact is not defined.

9.3 HumanActor — Examples

9.3.1 The uses and usedBy Properties Applied to HumanActor

In one direction, a human actor can itself use things such as services, systems, and other human actors. In the other direction, a human actor can, for instance, be used by another human actor or by a system (as an element within that system such as a human actor in a process).

9.3.2 The represents and representedBy Properties Applied to HumanActor

As mentioned in the introduction to this Clause, human actors are intrinsically part of systems that instantiate service oriented architectures. Yet in many cases as an element of an SOA system, the specific person or organization is not discussed, rather an abstract representation of them that participates in processes, provides services, etc. In other words, elements representing human actors are discussed

As examples, a broker (instance of **HumanActor**) may represent a seller (instance of **HumanActor**) that wishes to remain anonymous, a role (instance of **Element**) may represent (the role aspect of) multiple instances of **HumanActor**, and an organizational unit (instance of **HumanActor**) may represent the many people (all instances of **HumanActor**) that are part of it.

Note that a "role class" has not been defined, as using **Element** with the **represents** property is a more general approach which does not limit the ability to also define role-based systems. For all practical purposes there is simply a "role subclass" of **Element**, a subclass that is not defined explicitly.

9.3.3 Organizational Example

Continuing the organizational example from 8.8.1, P1 (*John*), P2 (*Jack*), P3 (*Joe*), and P4 (*Mary*) can now be expressed as instances of **Element** which are in fact (people) instances of **HumanActor**. All of O1 (*CarWashBusiness*), O3 (*CarWash*), and O4 (*Administration*) can also be expressed as (organization) human actors from an action perspective at the same time that they are systems from a collection/composition perspective.

9.3.4 Car Wash Example

See Annex A for the complete organizational aspect of the car wash example.

9.4 The Task Class

```
<owl:Class rdf:about="#Task">
  <owl:disjointWith>
   <owl:Class rdf:about="#System"/>
  </owl:disjointWith>
  <owl:disjointWith>
   <owl:Class rdf:about="#HumanActor"/>
  </owl:disjointWith>
  <owl:disjointWith>
   <owl:Class rdf:about="#Service"/>
  </owl:disjointWith>
  <rdfs:subClassOf>
   <owl:Class rdf:about="#Element"/>
  </rdfs:subClassOf>
</owl:Class>
```

A *task* is an atomic action which accomplishes a defined result. Tasks are done by people or organizations, specifically by instances of **HumanActor**.

The Business Process Modeling Notation (BPMN) 2.0 defines task as follows: "A Task is an atomic Activity within a Process flow (see Reference [4]). A Task is used when the work in the Process cannot be broken down to a finer level of detail. Generally, an end-user and/or applications are used to perform the Task when it is executed." For the purposes of the ontology, precision has been added by formally separating the notion of doing from the notion of performing. Tasks are (optionally) done by human actors, furthermore (as instances of Element) tasks can use services that are performed by technology components (see details in 10.3; see also the example in Annex A).

The concept of *task* is captured by the **Task** OWL class, which is illustrated in Figure 6.

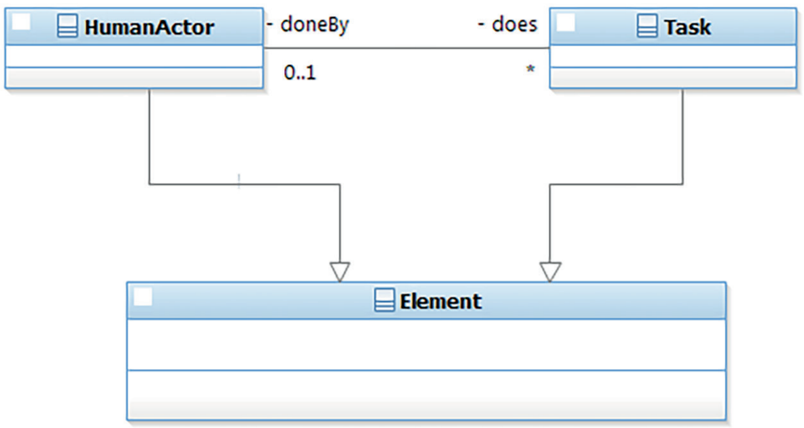

Figure 6 — The Task Class

Task is defined as disjoint with the *System*, *Service*, and *HumanActor* classes. Instances of these classes are considered not to be atomic actions.

9.5 The does and doneBy Properties

```
<owl:ObjectProperty rdf:about="#doneBy">
  <rdfs:domain rdf:resource="#Task"/>
  <rdfs:range rdf:resource="#HumanActor"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#does">
```

```
  <owl:inverseOf>
   <owl:ObjectProperty rdf:about="#doneBy"/>
  </owl:inverseOf>
</owl:ObjectProperty>

<owl:Class rdf:about="#Task">
  <rdfs:subClassOf>
   <owl:Restriction>
    <owl:onProperty>
     <owl:ObjectProperty rdf:about="#doneBy"/>
    </owl:onProperty>
    <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
>0</owl:minCardinality>
   </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
   <owl:Restriction>
    <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
>1</owl:maxCardinality>
    <owl:onProperty>
     <owl:ObjectProperty rdf:about="#doneBy"/>
    </owl:onProperty>
   </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Tasks are naturally thought of as being done by people or organizations. If thinking of tasks as being the actual things done, then the natural cardinality is that each instance of **Task** is done by at most one instance of **HumanActor**. Due to the atomic nature of instances of **Task** the case is ruled out where such an instance is done jointly by multiple instances of Human**Actor**. The cardinality can be zero if someone chooses not to instantiate all possible human actors. On the other hand, the same instance of **HumanActor** can (over time) easily do more than one instance of **Task**. The **does** property, and its inverse **doneBy**, capture the relation between a human actor and the tasks it does.

9.6 Task — Examples

9.6.1 The uses and usedBy Properties Applied to Task

In one direction, the most common case of a task using another element is where an automated task (in an orchestrated process; see Clause 11 for the definition of *process* and *orchestration*) uses a service as its realization. In the other direction, a task can, for instance, be used by a system (as an element within that system, such as a task in a process).

9.6.2 The represents and representedBy Properties Applied to Task

As mentioned in the introduction to this Clause, tasks are intrinsically part of SOA systems. Yet in many cases as an element of an SOA system, the actual thing being done is not discussed, rather an abstract representation of it that is used as an element in systems, processes, etc. In other words, discuss elements representing tasks.

As a simple example, an abstract activity in a process model (associated with a role) may represent a concrete task (done by a person fulfilling that role). Note that due to the atomic nature of a task it does not make sense to talk about many elements representing different aspects of it.

9.6.3 Organizational Example

Continuing the organizational example from 8.8.1, the tasks that are done by human actors (people) P1, P2, P3, and P4 can now be expressed, and how those tasks can be elements in bigger systems that describe things such as organizational processes. Clause 11 will deal formally with the concept of *composition*, including properly defining the concept of a *process* as one particular kind of *composition*.

9.6.4 Car Wash Example

As an important part of the car wash system, John and Jack perform certain manual tasks required for washing a car properly.

— *Jack* and *John* are instances of **HumanActor**.

— *WashWindows* is an instance of **Task** and is done by *John*.

— *PushWashButton* is an instance of **Task** and is done by *Jack*.

10 Service, ServiceContract, and ServiceInterface

10.1 Overview

Service is another core concept of this ontology. It is a concept that is fundamental to SOA and always used in practice when describing or engineering SOA systems, yet it is not easy to define formally. The ontology is based on the following definition of *service*:

A service is a "logical representation of a set of activities that has specified outcomes, is self-contained, may be composed of other services, and is a "black box" to consumers of the service"

This corresponds to the existing official definition of the term in the Reference architecture for SOA, ISO/IEC 18384-1.

The word activity in the definition of service is used in the general English language sense of the word, not in the process-specific sense of that same word (i.e. activities are not necessarily process activities). The ontology purposefully omits "business" as an intrinsic part of the definition of *service*. The reason for this is that the notion of business is relative to a person's viewpoint, as an example, one person's notion of IT is another person's notion of business (the business of IT). *Service* as defined by the ontology is agnostic to whether the concept is applied to the classical notion of a business domain or the classical notion of an IT domain.

Other current SOA-specific definitions of the term service include the following:

— *"A mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description."* (see Reference [3])

— *"A capability offered by one entity or entities to others using well-defined 'terms and conditions' and interfaces."* (see Reference [9])

Within the normal degree of precision of the English language, these definitions are not contradictory; they are stressing different aspects of the same concept. All three definitions are SOA-specific though, and represent a particular interpretation of the generic English language term service.

This Clause describes the following classes of the ontology:

— **Service**;

— **ServiceContract**;

— **ServiceInterface**;

— **InformationType.**

In addition, it defines the following properties:

— **performs** and **performedBy**;

— **hasContract** and **isContractFor**;

— **involvesParty** and **isPartyTo**;

— **specifies** and **isSpecifiedBy**;

— **hasInterface** and **isInterfaceOf**;

— **hasInput** and **isInputAt**;

— **hasOutput** and **isOutputAt**.

10.2 The Service Class

```
<owl:Class rdf:about="#Service">
  <owl:disjointWith>
    <owl:Class rdf:about="#System"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:about="#Task"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:about="#HumanActor"/>
  </owl:disjointWith>
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Element"/>
  </rdfs:subClassOf>
</owl:Class>
```

A *service* is a logical representation of a set of activities that has specified outcomes, is self-contained, may be composed of other services, and is a "black box" to consumers of the service. The concept of *service* is captured by the **Service** OWL class, which is illustrated in Figure 7.

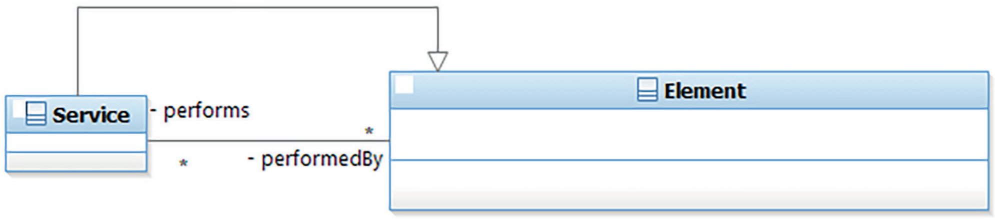

Figure 7 — The Service Class

In the context of the SOA ontology, only SOA-based services are considered. Other domains, such as integrated service management, can have services that are not SOA-based hence are outside the intended scope of the SOA ontology.

Service is defined as disjoint with the *System*, *Task*, and *HumanActor* classes. Instances of these classes are considered not to be services themselves, even though they may provide capabilities that can be offered as services.

10.3 The performs and performedBy Properties

```
<owl:ObjectProperty rdf:about="#performs">
  <rdfs:domain rdf:resource="#Element"/>
  <rdfs:range rdf:resource="#Service"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#performedBy">
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#performs"/>
  </owl:inverseOf>
</owl:ObjectProperty>
```

As a service itself is only a logical representation, any service is *performed* by something. The something that *performs* a service is opaque to anyone interacting with it, an opaqueness which is the exact nature of the **Element** class. This concept is captured by the *performs* and *performedBy* properties as illustrated in the Service Class in Figure 7.

This also captures the fact that services can be performed by elements of other types than systems. This includes elements such as software components, human actors, and tasks.

Note that the same instance of **Service** can be performed by many different instances of **Element**. As long as the service performed is the same, an external observer cannot tell the difference (for contractual obligations, SLAs, etc. see the definition of the *ServiceContract* class in 10.6). Conversely, any instance of **Element** may perform more than one service or none at all.

While a service can be performed by other elements, the service itself (as a purely logical representation) does not perform other services. See the Simple Service Composition Example (11.7.1) for an example of how to represent service compositions formally in the ontology.

10.4 Service Consumers and Service Providers

Terminology used in an SOA environment often includes the notions of service providers and service consumers. There are two challenges with this terminology:

— It does not distinguish between the contractual obligation aspect of consume/provide and the interaction aspect of consume/provide. A contractual obligation does not necessarily translate to an interaction dependency, if for no other reason than because the realization of the contractual obligation may have been sourced to a third party.

— Consuming or providing a service is a statement that only makes sense in context, either a contractual context or an interaction context. These terms are consequently not well suited for making statements about elements and services in isolation.

These are the reasons why the ontology has chosen not to adopt consume and provide as core concepts, rather instead allows consume or provide terms used with contractual obligations and/or interaction rules described by service contracts; see the definition of the *ServiceContract* class in 10.6. In its simplest form, outside the context of a formal service contract, the interaction aspect of consuming and providing services may even be expressed simply by saying that some element uses (consumes) a service or that some element performs (provides) a service; see also the examples in 10.5.

10.5 Service — Examples

10.5.1 The uses and usedBy properties Applied to Service

In one direction, it does not really make sense to talk about a service that uses another element. While the thing that performs the service might very well include the use of other elements (and certainly will in the case of *Service Composition*), the service itself (as a purely logical representation) does not use other elements.

In the other direction, the most common of all interactions is found in an SOA environment: the notion that some element uses a service by interacting with it. Note that from an operational perspective this interaction actually reaches somewhat beyond the service itself by involving the following typical steps:

— picking the service to interact with (this statement is agnostic as to whether this is done dynamically at runtime or statically at design and/or construct time);

— picking an element that performs that service [in a typical SOA environment, this is most often done “inside” an enterprise service bus (ESB)];

— interacting with the chosen element (that performs the chosen) service (often also facilitated by an ESB).

10.5.2 The represents and representedBy Properties Applied to Service

Concepts such as service mediations, service proxies, ESBs, etc. are natural to those practitioners that describe and implement the operational aspects of SOA systems. From an ontology perspective, all of these can be captured by some other element representing the service, a level of indirection that is critical when not wanting to bind operationally to a particular service endpoint, rather preserving loose coupling and the ability to switch embodiments as needed. Note that by leveraging the *represents* and *representedBy* properties in this fashion the relatively complex operational interaction pattern that was described in the 9.6.2 (picking the service, picking an element that performs the service, and interacting with that chosen element) is additionally encapsulated.

While a service being represented by something else is quite natural, it is harder to imagine what the service itself might represent. To some degree the fact that a service represents any embodiment of it has already been captured, only the *performs* and *performedBy* properties have been chosen to describe this rather than the generic *represents* and *representedBy* properties. As a consequence, practical applications of the ontology to have services represent anything is not expected.

10.5.3 Exemplifying the Difference Between Doing a Task and Performing a Service

The distinction between a human actor performing a task and an element (technology, human actor, or other) performing a service is important. The human actor doing the task has the responsibility that it gets done, yet may in fact in many cases leverage some service to achieve that outcome:

— *John* is an instance of **HumanActor**.

— *WashWindows* is an instance of **Task** and is done by *John*.

— *SoapWater* is an instance of **Service**.

— *WaterTap* is an instance of **Element**.

— *WaterTap* performs *SoapWater*.

— *John* uses *SoapWater* (to do *WashWindows*).

Note how clearly *SoapWater* does not do *WashWindows*, nor does *WaterTap* do *WashWindows.*

10.5.4 Car Wash Example

Joe offers two different services to his customers: a basic wash and a gold wash. This can be instantiated in the ontology in the following way (subset to the part relevant for these two services):

— *GoldWash* is an instance of **Service**.

— *BasicWash* is an instance of **Service**.

— *CarWash* performs both *BasicWash* and *GoldWash*.

— *WashManager* represents both BasicWash and *GoldWash* (i.e. is the interaction point where customers can order services as well as pay for them).

Note the purposeful use of *WashManager* representing both services. This is due to Joe deciding that in his car wash customers are not to interact with the washing machinery directly, rather instead interact with whomever (human actor) is fulfilling the role of wash manager.

10.6 The ServiceContract Class

```
<owl:Class rdf:about="#ServiceContract">
  <owl:disjointWith>
    <owl:Class rdf:about="#HumanActor"/>
  </owl:disjointWith>
  <owl:disjointWith>
```

```
    <owl:Class rdf:about="#Task"/>
  </owl:disjointWith>
</owl:Class>
```

In many cases, specific agreements are needed in order to define how to use a service. This can either be because of a desire to regulate such use or can simply be because the service will not function properly unless interaction with it is done in a certain sequence. A *service contract* defines the terms, conditions, and interaction rules that interacting participants agree to (directly or indirectly). A *service contract* is binding on all participants in the interaction, including the service itself and the element that provides it for the particular interaction in question. The concept of *service contract* is captured by the **ServiceContract** OWL class, which is illustrated in Figure 8.

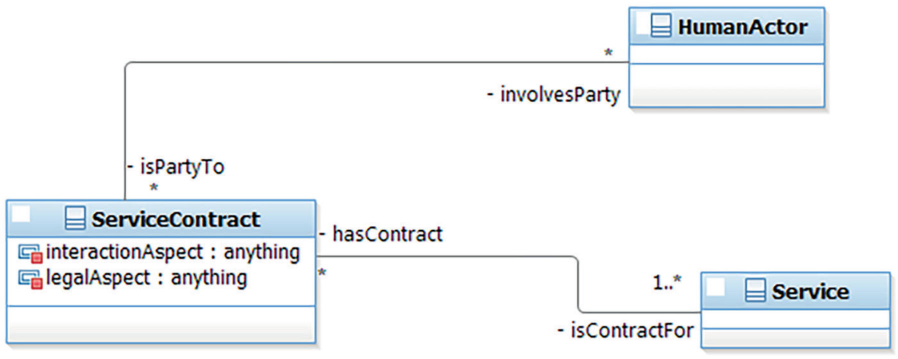

Figure 8 — The ServiceContract Class

10.7 The interactionAspect and legalAspect Datatype Properties

```
<owl:DatatypeProperty rdf:about="#interactionAspect">
  <rdfs:domain rdf:resource="#ServiceContract"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="#legalAspect">
  <rdfs:domain rdf:resource="#ServiceContract"/>
</owl:DatatypeProperty>

<owl:Class rdf:about="#ServiceContract">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:DatatypeProperty rdf:about="#legalAspect"/>
      </owl:onProperty>
      <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:maxCardinality>
      <owl:onProperty>
        <owl:DatatypeProperty rdf:about="#legalAspect"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:DatatypeProperty rdf:about="#interactionAspect"/>
      </owl:onProperty>
      <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:DatatypeProperty rdf:about="#interactionAspect"/>
      </owl:onProperty>
      <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Service contracts explicitly regulate both the interaction aspects (see the *hasContract* and *isContractFor* properties) and the legal agreement aspects (see the *involvedParty* and *isPartyTo* properties) of using a service. The two types of aspects are formally captured by defining the **interactionAspect** and **legalAspect** datatype properties on the **ServiceContract** class. Note that the second of these attributes, the legal agreement aspects, includes concepts such as service-level agreements (SLAs).

If desired, it is possible as an architectural convention to split the interaction and legal aspects into two different service contracts. Such choices will be up to any application using this ontology.

10.8 The hasContract and isContractFor Properties

```
<owl:ObjectProperty rdf:about="#isContractFor">
  <rdfs:domain rdf:resource="#ServiceContract"/>
  <rdfs:range rdf:resource="#Service"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#hasContract">
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#isContractFor"/>
  </owl:inverseOf>
</owl:ObjectProperty>

<owl:Class rdf:about="#ServiceContract">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#isContractFor"/>
      </owl:onProperty>
      <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

The **hasContract** property, and its inverse **isContractFor**, capture the abstract notion of a service having a service contract. Anyone wanting to use a service obeys to the interaction aspects (as defined in the **interactionAspect** datatype property) of any service contract applying to that interaction. In that fashion, the interaction aspects of a service contract are context-independent; they capture the defined or intrinsic ways in which a service may be used.

By definition, any service contract is a contract for at least one service. It is possible that the same service contract can be a contract for more than one service; for instance, in cases where a group of services share the same interaction pattern or where a service contract (legally – see the *involvesParty* and *isPartyTo* properties in 10.9) regulates the providing and consuming of multiple services.

10.9 The involvesParty and isPartyTo Properties

```
<owl:ObjectProperty rdf:about="#isPartyTo">
  <rdfs:domain rdf:resource="#HumanActor"/>
  <rdfs:range rdf:resource="#ServiceContract"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#involvesParty">
```

```
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#isPartyTo"/>
  </owl:inverseOf>
</owl:ObjectProperty>
```

In addition to the rules and regulations that intrinsically apply to any interaction with a service (the interaction aspect of service contracts captured in the **interactionAspect** datatype property) there may be additional legal agreements that apply to certain human actors and their use of services. The **involvesParty** property, and its inverse **isPartyTo**, captures the abstract notion of a service contract specifying legal obligations between human actors in the context of using the one or more services for which the service contract is a contract.

While the **involvesParty** and **isPartyTo** properties define the relationships to human actors involved in the service contract, the actual legal obligations on each of these human actors is defined in the **legalAspect** datatype property on the service contract. This includes the ability to define who is the provider and who is the consumer from a legal obligation perspective.

There is a many-to-many relationship between service contracts and human actors. A given human actor may be party to none, one, or many service contracts. Similarly, a given service contract may involve none, one, or multiple human actors (none in the case where that particular service contract only specifies the **interactionAspect** datatype property). Note that it is important to allow for sourcing contracts where there is a legal agreement between human actor A and human actor B (both of which are party to a service contract), yet human actor B has sourced the performing of the service to human actor C (*aka* human actor C performs the service in question, not human actor B).

The **involvesParty** property together with the **legalAspect** datatype property on **ServiceContract** capture not just transient obligations. They include the ability to express “is obliged to at this instant”, “was obliged to”, and “may in future be obliged to”.

10.10 The Effect Class

```
<owl:Class rdf:about="#Effect">
  <owl:disjointWith>
    <owl:Class rdf:about="#ServiceInterface"/>
  </owl:disjointWith>
</owl:Class>
```

Interacting with something performing a service has *effects*. These comprise the outcome of that interaction, and are how a service (through the element that performs it) delivers value to its consumers. The concept of *effect* is captured by the **Effect** OWL class, which is illustrated in Figure 9.

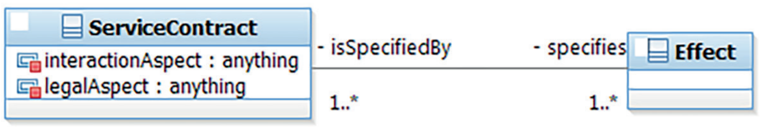

Figure 9 — The Effect Class

Note that the **Effect** class purely represents how results or value is delivered to someone interacting with a service. Any possible internal side-effects are explicitly not covered by the **Effect** class.

Effect is defined as disjoint with the *ServiceInterface* class. (The *ServiceInterface* class is defined later in this part of ISO/IEC 18384). Interacting with a service through its service interface can have an outcome or provide a value (an instance of **Effect**) but the service interface itself does not constitute that outcome or value.

10.11 The specifies and isSpecifiedBy Properties

```
<owl:ObjectProperty rdf:about="#specifies">
  <rdfs:domain rdf:resource="#ServiceContract"/>
  <rdfs:range rdf:resource="#Effect"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#isSpecifiedBy">
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#specifies"/>
  </owl:inverseOf>
</owl:ObjectProperty>

<owl:Class rdf: about="#Effect">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:minCardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf: about="#isSpecifiedBy"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:about="#ServiceContract">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf: about="#specifies"/>
      </owl:onProperty>
      <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

While a service intrinsically has an effect every time someone interacts with it, in order to trust the effect to be something in particular, the effect needs to be specified as part of a service contract. The **specifies** property, and its inverse **isSpecifiedBy**, capture the abstract notion of a service contract specifying a particular effect as part of the agreement for using a service. Note that the specified effect can apply to both the **interactionAspect** datatype property (simply specifying what will happen when interacting with the service according to the service contract) and the **legalAspect** datatype property (specifying a contractually promised effect).

Anyone wanting a guaranteed effect of the interaction with a given service ensures that the desired effect is specified in a service contract applying to that interaction. By definition, any service contract specifies at least one effect. In the other direction, an effect is an effect of at least one service contract; this represents that fact that those effects that are specified by service contracts are only formalized (and not all intrinsic effects of all services).

10.12 ServiceContract — Examples

10.12.1Service-level Agreements

A service-level agreement (SLA) on a service has been agreed by organizations A and B. It is important to realize that an SLA always has a context of the parties that have agreed to it, involving at a minimum one legal “consumer” and one legal “provider”. This can be represented in the ontology as follows:

— *A* and *B* are instances of **HumanActor**;

— *Service* is an instance of **Service**;

— *ServiceContract* is an instance of **ServiceContract**;

— *ServiceContract* isContractFor *Service*;

— *ServiceContract* involvesParty *A*;

— *ServiceContract* involvesParty *B*;

— The **legalAspect** datatype property on *ServiceContract* describes the SLA.

10.12.2 Service Sourcing

Organizations A and B have agreed on B providing certain services for A, yet B wants to source the actual delivery of those services to third party C. This can be represented in the ontology as follows:

— *A*, *B*, and *C* are instances of **HumanActor**;

— *Service* is an instance of **Service**;

— *C* provides *Service*;

— *ServiceContract* is an instance of **ServiceContract**;

— *ServiceContract* is ContractFor Service;

— *ServiceContract* involvesParty *A*;

— *ServiceContract* involvesParty *B*;

— The **legalAspect** datatype property on ServiceContract describes the legal obligation of B to provide Service for A.

10.12.3 Car Wash Example

See Annex A for the complete **Service** and **ServiceContract** aspects of the car wash example.

10.13 The ServiceInterface Class

```
<owl:Class rdf:about="#ServiceInterface">
  <owl:disjointWith>
    <owl:Class rdf: about="#Service"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf: about="#ServiceContract"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf: about="#Effect"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf: about="#HumanActor"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf: about="#Task"/>
  </owl:disjointWith>
</owl:Class>
```

An important characteristic of services is that they have simple, well-defined interfaces. This makes it easy to interact with them, and enables other elements to use them in a structured manner. A *service interface* defines the way in which other elements can interact and exchange information with a service. This concept is captured by the **ServiceInterface** class which is illustrated in Figure 10.

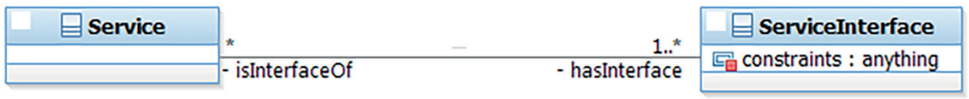

Figure 10 — The ServiceInterface Class

The concept of an interface is in general well understood by practitioners, including the notion that interfaces define the parameters for information going in and out of them when invoked. What differs from domain to domain is the specific nature of how an interface is invoked and how information is passed back and forth. Service interfaces are typically, but not necessarily, message-based (to support loose coupling). Furthermore, service interfaces are always defined independently from any service implementing them (to support loose coupling and service mediation).

From a design perspective interfaces may have more granular operations or may be composed of other interfaces, however, this part of ISO/IEC 18384 has been kept at the concept level and does not include such design aspects in the ontology.

ServiceInterface is defined as disjoint with the **Service**, **ServiceContract**, and **Effect** classes. Instances of these classes are considered not to define (by themselves) the way in which other elements can interact and exchange information with a service. Note that that there is a natural synergy between **ServiceInterface** and the **interactionAspect** datatype property on **ServiceContract**, as the latter defines any multi-interaction and/or sequencing constraints on how to use a service through interaction with its service interfaces.

10.14 The Constraints Datatype Property

```
<owl:DatatypeProperty rdf:about="#constraints">
  <rdfs:domain rdf:resource="#ServiceInterface"/>
</owl:DatatypeProperty>

<owl:Class rdf:about="#ServiceInterface">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:DatatypeProperty rdf: about="#constraints"/>
      </owl:onProperty>
      <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:DatatypeProperty rdf:about="#constraints"/>
      </owl:onProperty>
      <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

The **Constraints** datatype property on **ServiceInterface** captures the notion that there can be constraints on the allowed interaction such as only certain value ranges allowed on given parameters. Depending on the nature of the service and the service interface in question these constraints may be defined either formally or informally (the informal case being relevant at a minimum for certain types of real-world services).

10.15 The hasInterface and isInterfaceOf Properties

```
<owl:ObjectProperty rdf:about="#hasInterface">
  <rdfs:domain rdf:resource="#Service"/>
  <rdfs:range rdf:resource="#ServiceInterface"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf: about="#isInterfaceOf">
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#hasInterface"/>
  </owl:inverseOf>
</owl:ObjectProperty>

<owl:Class rdf:about="#Service">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:minCardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf: about="#hasInterface"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

The **hasInterface** property, and its inverse **isInterfaceOf**, capture the abstract notion of a service having a particular service interface.

In one direction, any service has at least one service interface; anything else would be contrary to the definition of a service as a representation of a set of activities that has a specified outcome and is a “black box” to its consumers. In the other direction, there can be service interfaces that are not yet interfaces of any defined services. Also, the same service interface can be an interface of multiple services. The latter does not mean that these services are the same, nor even that they have the same effect, it only means that it is possible to interact with all these services in the manner defined by the service interface in question.

10.16 The InformationType Class

```
<owl:Class rdf: about="#InformationType">
  <owl:disjointWith>
    <owl:Class rdf: about="#Effect"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf: about="#ServiceContract"/>
  </owl:disjointWith>
</owl:Class>
```

A service interface can enable another element to give information to or receive information from a service (when it uses that service), specifically the types of information given or received. The concept of *information type* is captured by the **InformationType** OWL class, which is illustrated in Figure 11.

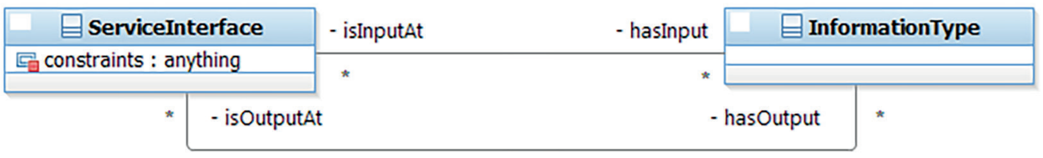

Figure 11 — The InformationType Class

In any concrete interaction through a service interface, the information types on that interface are instantiated by information items, yet for the service interface itself it is the types that are important.

Note that the **constraints** datatype property on **ServiceInterface**, if necessary, can be used to express constraints on allowed values for certain information types.

10.17 The hasInput and isInputAt Properties

```
<owl:ObjectProperty rdf: about="#hasInput">
  <rdfs:domain rdf:resource="#ServiceInterface"/>
  <rdfs:range rdf:resource="#InformationType"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf: about="#isInputAt">
  <owl:inverseOf>
    <owl:ObjectProperty rdf: about="#hasInput"/>
  </owl:inverseOf>
</owl:ObjectProperty>
```

The **hasInput** property, and its inverse **isInputAt**, capture the abstract notion of a particular type of information being given when interacting with a service through a service interface.

Note that there is a many-to-many relationship between service interfaces and input information types. A given information type may be input at many service interfaces or none at all. Similarly, a given service interface may have many information types as input or none at all. It is important to realize that some services may have only inputs (triggering an asynchronous action without a defined response) and other services may have only outputs (elements performing these services execute independently yet may provide output that is used by other elements).

10.18 The hasOutput and isOutputAt Properties

```
<owl:ObjectProperty rdf: about="#hasOutput">
  <rdfs:domain rdf:resource="#ServiceInterface"/>
  <rdfs:range rdf:resource="#InformationType"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf: about="#isOutputAt">
  <owl:inverseOf>
    <owl:ObjectProperty rdf: about="#hasOutput"/>
  </owl:inverseOf>
</owl:ObjectProperty>
```

The **hasOutput** property, and its inverse **isOutputAt**, capture the abstract notion of a particular type of information being received when interacting with a service through a service interface.

Note that there is a many-to-many relationship between service interfaces and output information types. A given information type may be output at many service interfaces or none at all. Similarly, a given service interface may have many information types as output or none at all. It is important to realize that some services may have only inputs (triggering an asynchronous action without a defined response) and other services may have only outputs (elements performing these services execute independently yet may provide output that is used by other elements).

10.19 Examples

10.19.1Interaction Sequencing

A service contract on a service expresses that the services interfaces on that services are used in a certain order.

— *Service* is an instance of **Service**.

— *ServiceContract* is an instance of **ServiceContract**.

— *ServiceContract* isContractFor *Service*.

— *X* is an instance of *ServiceInterface*.

— *X* isInterfaceOf *Service*.

— *Y* is an instance of *ServiceInterface*.

— *Y* isInterfaceOf *Service*.

— The **interactionAspect** datatype property on *ServiceContract* describes that *X* is used before *Y* may be used.

10.19.2 Car wash example

See Annex A for the complete **ServiceInterface** aspect of the car wash example.

11 Composition and its Subclasses

11.1 Overview

The notion of Composition is a core concept of SOA. Services can be composed of other services. Processes are composed of human actors, tasks, and possibly services. Experienced SOA practitioners intuitively apply composition as an integral part of architecting, designing, and realizing SOA systems; in fact, any well-structured SOA environment is intrinsically composite in the way services and processes support business capabilities. What differs from practitioner to practitioner is the exact nature of the composition, the composition pattern being applied.

This Clause describes the following classes of the ontology:

Composition (as a subclass of **System**)

ServiceComposition (as a subclass of **Composition**)

Process (as a subclass of **Composition**)

In addition, it defines the following datatype property:

compositionPattern

11.2 The Composition Class

```
<owl:Class rdf:about="#Composition">
  <rdfs:subClassOf>
   <owl:Class rdf: about="#System"/>
  </rdfs:subClassOf>
  <owl:disjointWith>
   <owl:Class rdf: about="#Task"/>
  </owl:disjointWith>
</owl:Class>
```

A *composition* is the result of assembling a collection of things for a particular purpose. Note in particular the act of composing has been purposefully distinguished from the resulting composition as a thing, and that it is in the latter sense the concept of *composition* is used here. The concept of *composition* is captured by the **Composition** OWL class, which is illustrated in Figure 12.

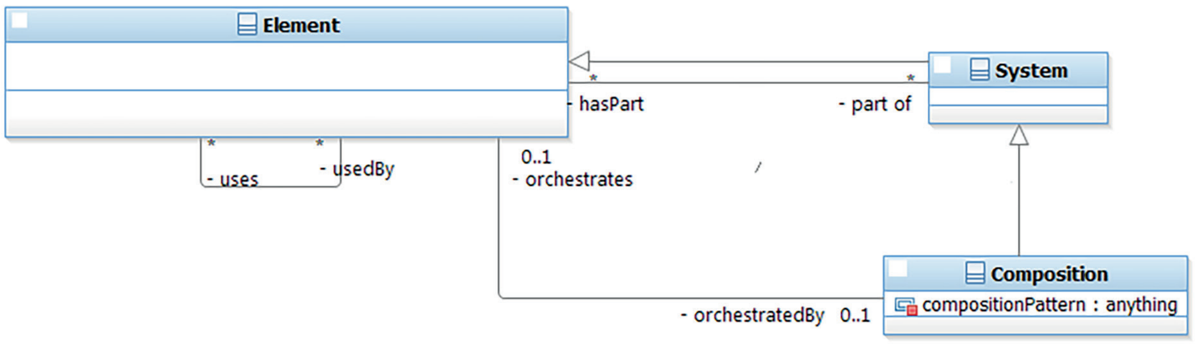

Figure 12 — The Composition Class

Being intrinsically (also) an organized collection of other, simpler things, the **Composition** class is a subclass of the **System** class. While a composition is always also a system, a system is not necessarily a composition in that it is not necessarily a result of anything, note here the difference between a system producing a result and the system itself being a result. A perhaps more tangible difference between a system and a composition is that the latter has associated with it a specific composition pattern that renders the composition (as a whole) as the result when that composition pattern is applied to the elements used in the composition. One implication of this is that there is not a single member of a composition that represents (as an element) that composition as a whole; in other words, the composition itself is not one of the things being assembled. On the other hand, *composition* is in fact a recursive concept (as are all subclasses of **System**), being a system, a composition is also an element which means that it can be used by a higher-level composition.

In the context of the SOA ontology, only functional compositions that belong to the SOA domain are considered in detail. Note that a fully described instance of **Composition** is by its nature a *uses* relationship to at least one instance of **Element.** (It need not necessarily have more than one as the composition pattern applied may be, for instance, simply a transformation.) Again (as for **System**) it is important to realize that a composition can use elements outside its own boundary.

Since **Composition** is a subclass of **Element**, all compositions have a boundary and are opaque to an external observer (black box view). The composition pattern in turn is the internal view point (white box view) of a composition. As an example, for the notion of a service composition this would correspond to the difference between seeing the service composition as an element providing a (higher-level) service or seeing the service composition as a composite structure of (lower-level) services.

11.3 The compositionPattern Datatype Property

11.3.1 Overview

```
<owl:DatatypeProperty rdf:about="#compositionPattern">
    <rdfs:domain rdf:resource="#Composition"/>
</owl:DatatypeProperty>

<owl:Class rdf:about="#Composition">
<rdfs:subClassOf>
  <owl:Restriction>
   <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</
owl:maxCardinality>
   <owl:onProperty>
    <owl:DatatypeProperty rdf: about="#compositionPattern"/>
   </owl:onProperty>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
   <owl:Restriction>
```

```
    <owl:onProperty>
     <owl:DatatypeProperty rdf: about="#compositionPattern"/>
    </owl:onProperty>
    <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</
owl:minCardinality>
   </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

As discussed in 11.2, any composition has associated with it a specific composition pattern, that pattern describing the way in which a collection of elements is assembled to a result. The concept of a composition pattern is captured by the **compositionPattern** datatype property. Note that even though certain kinds of composition patterns are of special interest within SOA (see 11.3.2), the **compositionPattern** data type property may take any value as long as that value describes how to assemble the elements used by the composition with which it is associated.

11.3.2 The Orchestration Composition Pattern

One kind of composition pattern that has special interest within SOA is an *Orchestration*. In an orchestration (a composition whose composition pattern is an orchestration), there is one particular element used by the composition that oversees and directs the other elements. Note that the element that directs an orchestration by definition is different than the orchestration (**Composition** instance) itself.

Think of an orchestrated executable workflow as an example of an orchestration. The workflow construct itself is one of the elements being used in the composition, yet it is different from the composition itself, the composition itself is the result of applying (executing) the workflow on the processes, human actors, services, etc. that are orchestrated by the workflow construct.

A non-IT example is the foreman of a road repair crew. If the foreman chooses to exert direct control over the tasks done by his crew, then the resulting composition becomes an orchestration (with the foreman as the director and provider of the composition pattern). Note that under other circumstances, with a different team composition model, a road repair crew can also act as a collaboration or a choreography (see 11.3.3 and 11.3.4 for definitions of collaboration and choreography).

As the last example clearly shows, using an orchestration composition pattern is not a guarantee that “nothing can go wrong”. That would, in fact, depend on the orchestration director’s ability to handle exceptions.

11.3.3 The Choreography Composition Pattern

Another kind of composition pattern that has special interest within SOA is a *Choreography*. In a choreography (a composition whose composition pattern is a choreography) the elements used by the composition interact in a non-directed fashion, yet with each autonomous member knowing and following a predefined pattern of behaviour for the entire composition.

Think of a process model as an example of choreography. The process model does not direct the elements within it, yet does provide a predefined pattern of behaviour that each such element is expected to conform to when “executing”.

11.3.4 The Collaboration Composition Pattern

A third kind of composition pattern that has special interest within SOA is a *Collaboration*. In collaboration (a composition whose composition pattern is a collaboration) the elements used by the composition interact in a non-directed fashion, each according to their own plans and purposes without a predefined pattern of behaviour. Each element simply knows what it has to do and does it independently, initiating interaction with the other members of the composition as applicable on its own initiative. This means that there is no overall predefined “flow” of the collaboration, though there may be a run-time “observed flow of interactions”.

A good example of collaboration is a work meeting. There is no script for how the meeting will unfold and only after the meeting has concluded can the sequence of interactions that actually occurred be described.

11.4 The orchestrates and orchestratedBy Properties

```
<owl:ObjectProperty rdf:about="#orchestratedBy">
    <rdfs:domain rdf:resource="#Composition"/>
    <rdfs:range rdf:resource="#Element"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#orchestrates">
  <owl:inverseOf>
   <owl:ObjectProperty rdf: about="#orchestratedBy"/>
  </owl:inverseOf>
</owl:ObjectProperty>

<owl:Class rdf:about="#Composition">
  <rdfs:subClassOf>
   <owl:Restriction>
    <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</
owl:maxCardinality>
    <owl:onProperty>
     <owl:ObjectProperty rdf: about="#orchestratedBy"/>
    </owl:onProperty>
   </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
   <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">0</
owl:minCardinality>
   <owl:onProperty>
    <owl:ObjectProperty rdf: about="#orchestratedBy"/>
   </owl:onProperty>
  </owl:Restriction>
</rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:about="#Element">
  <rdfs:subClassOf>
   <owl:Restriction>
    <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">0</
owl:minCardinality>
    <owl:onProperty>
     <owl:ObjectProperty rdf: about="#orchestrates"/>
    </owl:onProperty>
   </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
   <owl:Restriction>
    <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</
owl:maxCardinality>
    <owl:onProperty>
     <owl:ObjectProperty rdf:about="#orchestrates"/>
    </owl:onProperty>
   </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

An orchestration has one particular element that oversees and directs the other elements used by the composition. This type of relationship is important enough that the abstract notion is captured in the **orchestrates** property and its inverse **orchestratedBy**.

In one direction, a composition has at most one element that orchestrates it, and the cardinality can only be one if in fact the composition pattern of that composition is an orchestration. In the other direction, an element can orchestrate at most one composition which then has an orchestration as its composition pattern.

Note that in practical applications of the ontology, even though **Service** is a subclass of **Element**, a service (as a purely logical representation) is not expected to orchestrate a composition.

11.5 The ServiceComposition Class

```
<owl:Class rdf: about="#ServiceComposition">
   <rdfs:subClassOf>
    <owl:Class rdf: about="#Composition"/>
   </rdfs:subClassOf>
   <owl:disjointWith>
    <owl:Class rdf: about="#ServiceContract"/>
   </owl:disjointWith>
   <owl:disjointWith>
    <owl:Class rdf:about="#ServiceInterface"/>
   </owl:disjointWith>
</owl:Class>
```

A key SOA concept is the notion of *service composition*, the result of assembling a collection of services in order to perform a new higher-level service. The concept of *service composition* is captured by the **ServiceComposition** OWL class, which is illustrated in Figure 13.

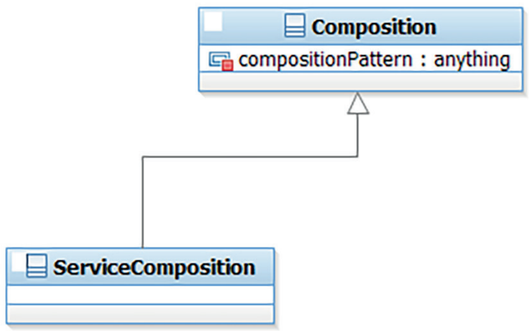

Figure 13 — The ServiceComposition Class

As a *service composition* is the result of assembling a collection of services, **ServiceComposition** is naturally a subclass of **Composition**.

A service composition may, and typically will, add logic (or even “code”) via the composition pattern. Note that a service composition is *not* the new higher-level service itself (due to the **System** and **Service** classes being disjoint); rather it performs (as an element) that higher-level service.

11.6 The Process Class

```
<owl:Class rdf: about="#Process">
   <rdfs:subClassOf>
    <owl:Class rdf: about="#Composition"/>
   </rdfs:subClassOf>
   <owl:disjointWith>
    <owl:Class rdf: about="#ServiceContract"/>
   </owl:disjointWith>
   <owl:disjointWith>
    <owl:Class rdf: about="#ServiceInterface"/>
   </owl:disjointWith>
</owl:Class>
```

Another key SOA concept is the notion of *process*. A *process* is a composition whose elements are composed into a sequence or flow of activities and interactions with the objective of carrying out certain work. This definition is consistent with, for instance, the Business Process Modeling Notation (BPMN) 2.0 definition of a process. (see Reference [4]). The concept of *process* is captured by the **Process** OWL class, which is illustrated in Figure 14.

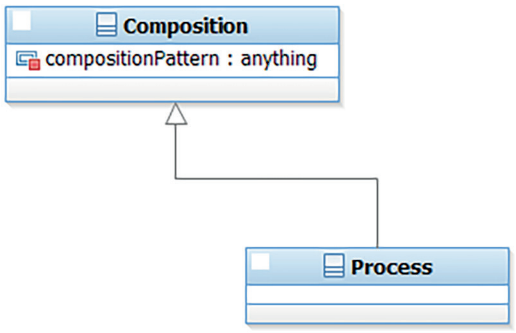

Figure 14 — The Process Class

Elements in process compositions can be things like human actors, tasks, services, other processes, etc. A process always adds logic via the composition pattern, the result is more than the parts. According to their collaboration pattern, processes can be as follows:

— **Orchestrated**: When a process is orchestrated in a Business Process Management System, then the resulting IT artifact is in fact an orchestration; i.e. it has an orchestration collaboration pattern. This type of process is often called a “Process Orchestration”.

— **Choreographed**: A process model representing a defined pattern of behaviour is often called a “Process Choreography”.

— **Collaborative**: No (pre)defined pattern of behaviour (model); the process represents observed (executed) behaviour.

11.7 Service Composition and Process Examples

11.7.1 Simple Service Composition Example

Using a service composition example, services A and B are instances of **Service** and the composition of A and B is an instance of **ServiceComposition** (that uses A and B):

— *A* and *B* are instances of **Service**,

— *X* is an instance of **ServiceComposition**, and

— *X* uses both *A* and *B* (composes them according to its service composition pattern).

Note that there are various ways in which the service composition pattern can compose A and B, all of which are relevant in one situation or another. For example, interfaces of X may or may not include some subset of the interfaces of A and B. Furthermore, the interfaces of A and B may or may not also be (directly) invocable without going through X, that is, a matter of the service contracts and/or access policies apply to the A and B. Finally, X may also use other elements that are not services at all (examples are composition code, adaptors, etc.).

11.7.2 Process Example

Using a process example, tasks T1 and T2 are instances of **Task**, roles R1 and R2 are instances of **Element**, and the composition of T1, T2, R1, and R2 is an instance of **Process** (that uses T1, T2, R1, and R2):

— *T1* and *T2* are instances of **Task**,

— *R1* and *R2* are instances of **Element**,

— *Y* is an instance of **Process**, and

— *Y* uses all of *T1, T2, R1*, and *R2* (composes them according to its process composition pattern).

11.7.3 Process and Service Composition Example

Elaborating on the process example in 11.7.2, if T1 is done using service S then:

— *S* is an instance of **Service**, and

— *T1* uses *S*.

Note that depending on the particular design approach chosen (and the resulting composition pattern), Y may or may not use S directly. This depends on whether Y carries the binding between T1 and S or whether that binding is encapsulated in T1.

11.7.4 Car Wash Example

See Annex A for the **Process** aspect of the car wash example.

12 Policy

12.1 Overview

Policies, the human actors defining them, and the things that they apply to are important aspects of any system, certainly also SOA systems with their many different interacting elements. Policies can apply to any element in a system. The concept of *Policy* is captured by the **Policy** class and its relationships to the **HumanActor** and **Thing** classes.

This Clause describes the following classes of the ontology:

Policy

In addition, it defines the following properties:

appliesTo and **isSubjectTo**

setsPolicy and **isSetBy**

12.2 The Policy Class

```
<owl:Class rdf:about="#Policy">
   <owl:disjointWith>
    <owl:Class rdf: about="#InformationType"/>
   </owl:disjointWith>
   <owl:disjointWith>
    <owl:Class rdf: about="#ServiceInterface"/>
   </owl:disjointWith>
   <owl:disjointWith>
    <owl:Class rdf: about="#Element"/>
   </owl:disjointWith>
   <owl:disjointWith>
    <owl:Class rdf: about="#Effect"/>
   </owl:disjointWith>
   <owl:disjointWith>
    <owl:Class rdf: about="#Event"/>
   </owl:disjointWith>
   <owl:disjointWith>
    <owl:Class rdf: about="#ServiceContract"/>
   </owl:disjointWith>
</owl:Class>
```

A *policy* is a statement of direction that a human actor may intend to follow or may intend that another human actor should follow. Knowing the policies that apply to something makes it easier and more transparent to interact with that something. The concept of *policy* is captured by the **Policy** OWL class, which is illustrated in Figure 15.

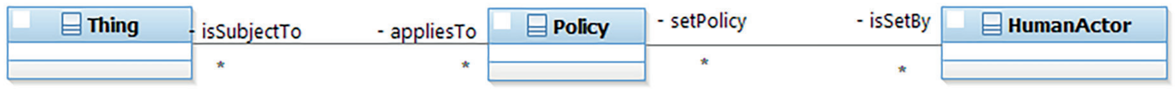

Figure 15 — The Policy Class

Policy as a concept is generic and has relevance outside the domain of SOA. For the purposes of this SOA ontology, it has not been necessary or relevant to restrict the generic nature of the **Policy** class itself. The relationships between **Policy** and **HumanActor** are of course bound by the SOA-specific restrictions that have been applied on the definition of **HumanActor**.

From a design perspective, policies may have more granular parts or may be expressed and made operational through specific rules. This part of ISO/IEC 18384 stays at the concept level and does not include such design aspects in the ontology.

Policy is distinct from all other concepts in this ontology; hence the **Policy** class is defined as disjoint with all other defined classes. In particular, **Policy** is disjoint with **ServiceContract**. While policies may apply to service contracts, such as security policies on who may change a given service contract, or conversely be referred to by service contracts as part of the terms, conditions, and interaction rules that interacting participants agree to, service contracts are themselves not policies as they do not describe an intended course of action.

12.3 The appliesTo and isSubjectTo Properties

```
<owl:ObjectProperty rdf: about="#appliesTo">
   <rdfs:domain rdf:resource="#Policy"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf: about="#isSubjectTo">
   <owl:inverseOf>
    <owl:ObjectProperty rdf: about="#appliesTo"/>
   </owl:inverseOf>
</owl:ObjectProperty>
```

Policies can apply to things other than elements; in fact, policies can apply to anything at all, including other policies. For instance, a security policy might specify which actors have the authority to change some other policy. The **appliesTo** property, and its inverse **isSubjectTo**, capture the abstract notion that a policy can apply to any instance of **Thing**. Note specifically that **Element** is a subclass of **Thing**, hence policies by inference can apply to any instance of **Element**.

In one direction, a policy can apply to zero (in the case where a policy has been formulated but not yet explicitly applied to anything), one, or more instances of **Thing**. Note that having a policy apply to multiple things does not mean that these things are the same, only that they are (partly) regulated by the same intent. In the other direction, an instance of **Thing** may be subject to zero, one, or more policies. Note that where multiple policies apply to the same instance of **Thing** this is often because the multiple policies are from multiple different policy domains (such as security and governance).

The SOA ontology does not attempt to enumerate different policy domains; such policy-focused details are deemed more appropriate for a policy ontology. It is worth pointing out that a particular policy ontology may also restrict (if desired) the kinds of things that policies can apply to.

12.4 The setsPolicy and isSetBy Properties

```
<owl:ObjectProperty rdf:about="#setsPolicy">
<rdfs:domain rdf:resource="#HumanActor"/>
```

```
<rdfs:range rdf:resource="#Policy"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf: about="#isSetBy">
<owl:inverseOf>
<owl:ObjectProperty rdf: about="#setsPolicy"/>
</owl:inverseOf>
</owl:ObjectProperty>
```

The **setsPolicy** property, and its inverse **isSetBy**, capture the abstract notion that a policy can be set by one or more human actors.

In one direction, a policy can be set by zero (in the case where actors setting the policy by choice are not defined or captured), one, or more human actors. Note specifically that some policies are set by multiple human actors in conjunction, meaning that all these human actors need to discuss and agree on the policy before it can take effect. A real-world example would be two parents in conjunction setting policies for acceptable child behaviour. In the other direction, a human actor may set (or be part of setting) multiple policies.

The SOA ontology purposefully separates the setting of the policy itself and the application of the policy to one or more instances of **Thing**. In some cases, these two acts may be inseparably bound together, yet in other cases, they are definitely not. One such example is an overall compliance policy that is formulated at the corporate level yet applied by the compliance officer in each line of business.

Also, while a particular case of interest for this ontology is that where the provider of a service has a policy for the service, a policy for a service is not necessarily owned by the provider. For example, government food and hygiene regulations (a policy that is law) cover restaurant services independently of anything desired or defined by the restaurant owner.

12.5 Examples

12.5.1 Car Wash Example

See A.5 for the **Policy** aspect of the car wash example.

13 Event

13.1 Overview

Events and the elements that generate or respond to them are important aspects of any event emitting system. SOA systems are in fact often event emitting, hence *event* is defined as a concept in the SOA ontology.

This Clause describes the following classes of the ontology:

Event

In addition, it defines the following properties:

generates and **generatedBy**

respondsTo and **respondedToBy**

13.2 The Event Class

```
<owl:Class rdf:about="#Event">
   <owl:disjointWith>
    <owl:Class rdf: about="#Policy"/>
   </owl:disjointWith>
   <owl:disjointWith>
    <owl:Class rdf: about="#ServiceContract"/>
   </owl:disjointWith>
   <owl:disjointWith>
    <owl:Class rdf: about="#ServiceInterface"/>
   </owl:disjointWith>
</owl:Class>
```

An *event* is something that happens, to which an element may choose to respond. Events can be responded to by any element. Similarly, events may be generated (emitted) by any element. Knowing the events generated or responded to by an element makes it easier and more transparent to interact with that element. Note that some events may occur whether generated or responded to by an element or not. The concept of an *event* is captured by the **Event** OWL class which is illustrated in Figure 16.

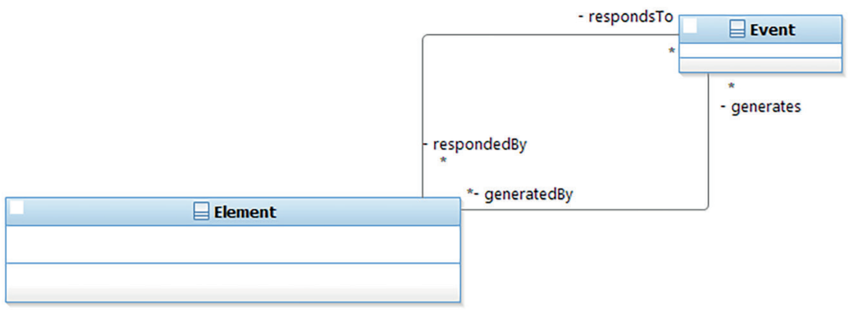

Figure 16 — The Event Class

Event as a concept is generic and has relevance to the domain of SOA as well as many other domains. For the purposes of this ontology, *Event* is used in its generic sense.

From a design perspective, events may have more granular parts or may be expressed and made operational through specific syntax or semantics. This part of ISO/IEC 18384 stays at the concept level and does not include such design aspects in the ontology.

13.3 The generates and generatedBy Properties

```
<owl:ObjectProperty rdf: about="#generates">
   <rdfs:domain rdf:resource="#Element"/>
   <rdfs:range rdf:resource="#Event"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf: about="#generatedBy">
   <owl:inverseOf>
    <owl:ObjectProperty rdf: about="#generates"/>
   </owl:inverseOf>
</owl:ObjectProperty>
```

Events can, but need not necessarily, be generated by elements. The **generates** property, and its inverse **generatedBy**, captures the abstract notion that an element generates an event.

Note that the same event may be generated by many different elements. Similarly, the same element may generate many different events.

13.4 The respondsTo and respondedToBy Properties

```
<owl:ObjectProperty rdf: about="#respondsTo">
   <rdfs:domain rdf:resource="#Element"/>
   <rdfs:range rdf:resource="#Event"/>
</owl:ObjectProperty>
```

```
<owl:ObjectProperty rdf: about="#respondedToBy">
   <owl:inverseOf>
    <owl:ObjectProperty rdf: about="#respondsTo"/>
   </owl:inverseOf>
</owl:ObjectProperty>
```

Events can, but need not necessarily, be responded to by elements. The **respondsTo** property and its inverse **respondedToBy**, capture the abstract notion that an element responds to an event.

The same event may be responded to by many different elements. Similarly, the same element may respond to many different events.

Annex A
(informative)

Complete Car Wash Example

A.1 General

This Annex contains the complete car wash example that has been used in parts throughout the definitional clauses of the ontology.

A.2 The Organizational Aspect

Joe the owner chooses to organize his business into two organizational units: *Administration* and *CarWash*:

— *CarWashBusiness* is an instance of both **HumanActor** and **System**,

— *Administration* is an instance of **HumanActor** (organizational unit),

— *CarWash* is an instance of **HumanActor** (organizational unit),

— *CarWashBusiness* uses (has organizational units) *Administration* and *CarWash*,

— *AdministrativeSystem* is an instance of **System**,

— *Administration* represents *AdministrativeSystem*,

— *CarWashSystem* is an instance of **System**, and

— *CarWash* represents *CarWashSystem*.

And using well-defined roles within each organization:

— *Owner* (role) is an instance of **Element** and is used by *AdministrativeSystem*,

— *Joe* is an instance of **HumanActor** and is represented by (has role) *Owner*,

— *Secretary* (role) is an instance of **Element** and is used by *AdministrativeSystem*,

— *Mary* is an instance of **HumanActor** and is represented by (has role) *Secretary*,

— *PreWashGuy* (role) is an instance of **Element** and is used by *CarWashSystem*,

— *John* is an instance of **HumanActor** and is represented by (has role) *PreWashGuy*,

— *WashManager* (role) is an instance of **Element** and is used by *CarWashSystem*,

— *WashOperator* (role) is an instance of **Element** and is used by *CarWashSystem*, and

— *Jack* is an instance of **HumanActor** and is represented by (has roles) both *WashManager* and *WashOperator*.

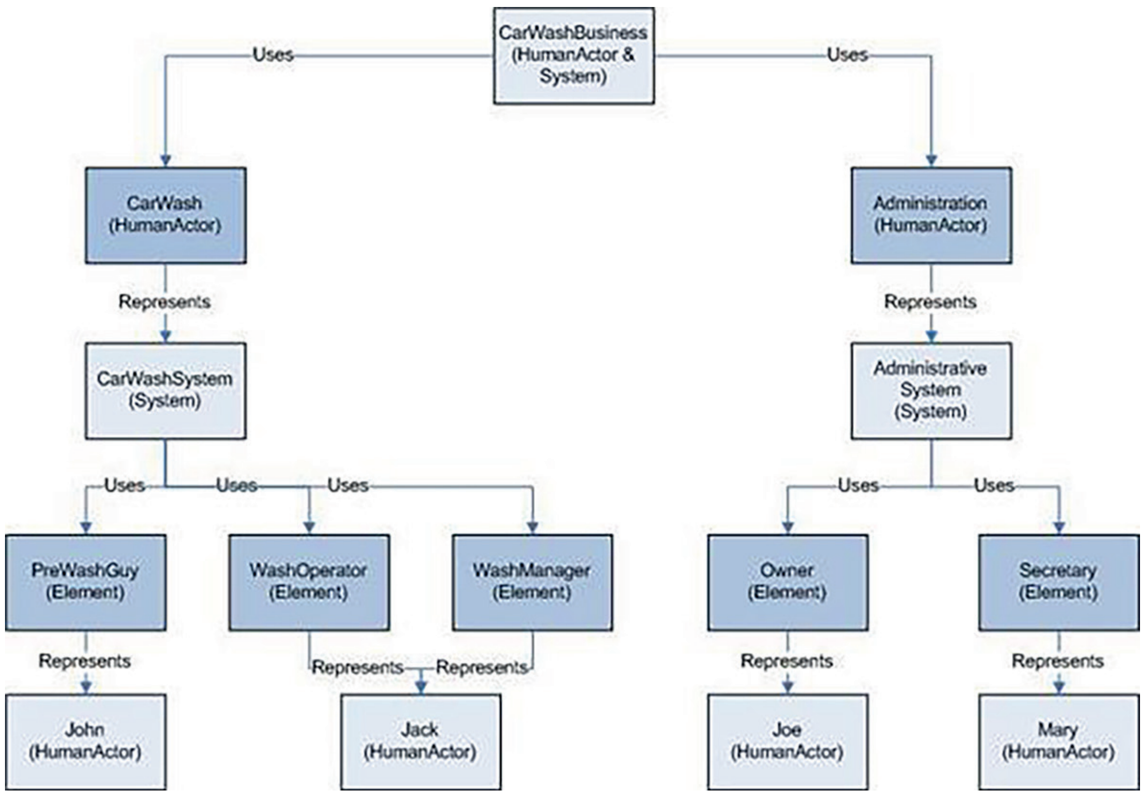

Figure A.1 — Car Wash Example — The Organizational Aspect

A.3 The Washing Services

Joe offers two different services to his customers: a basic wash and a gold wash:

— *GoldWash* is an instance of **Service**,

— *BasicWash* is an instance of **Service**,

— *CarWash* performs both *BasicWash* and *GoldWash*, and

— *WashManager* represents both BasicWash and *GoldWash* (i.e. it is the interaction point where customers can order services as well as pay for them).

In return for payment, Joe's *BasicWash* service cleans the car of customer Judy:

— *Judy* is an instance of **HumanActor** (the customer),

— *BasicWashContract* is an instance of **ServiceContract**,

— *BasicWash* has contract *BasicWashContract,*

— *CleanCar* is an instance of **Effect**,

— *BasicWashContract* specifies *CleanCar* as its effect,

— *BasicWashContract* involves parties *CarWashBusiness* and *Judy* and specifies that *Judy* (as the legal consumer) pays *CarWashBusiness* (as the legal provider) $10 for the one consumption of *BasicWash* with the effect of (one) *CleanCar*. Note that *BasicWash* is actually performed by *CarWash* and not by the legal provider *CarWashBusiness*, in this particular example *CarWash* happens to be a member of *CarWashBusiness* but such need not always be the case, *CarWash* could have been some third party provider, and

— *Judy* uses *WashManager* (in order to invoke the *BasicWash* service).

Note that in this example Judy does not interact with the (abstract) *BasicWash* service directly, rather she interacts with the *WashManager* that represents the service. This is due to Joe deciding that in his car wash customers are not to interact with the washing machinery directly.

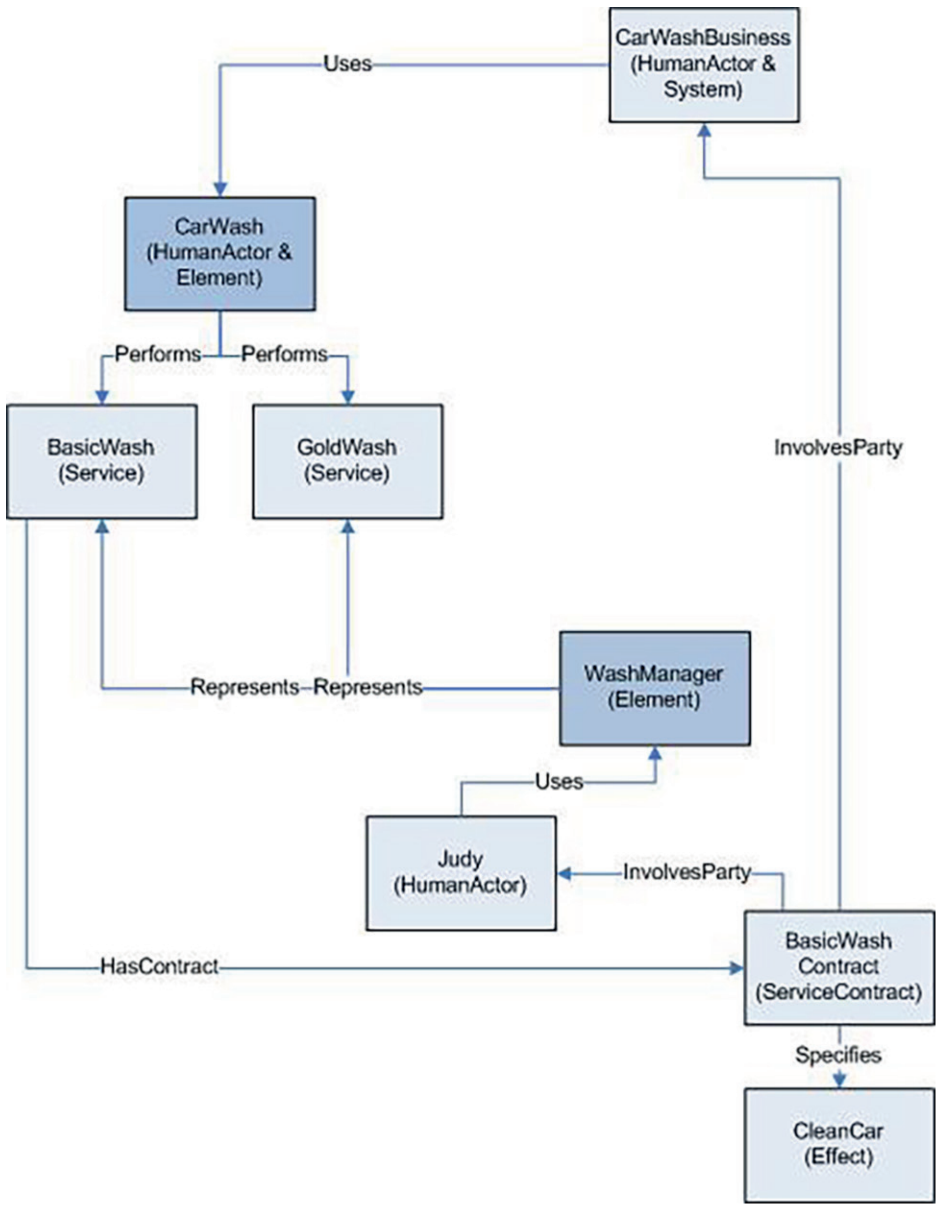

Figure A.2 — Car Wash Example — The Washing Services

A.4 Interfaces to the Washing Services

The way to interact with the car wash services is simple for the customer; he or she simply gives money to the wash manager and asks to have to the car washed using one of the two available wash services. Due to the fact that Joe has decided to interpose the wash manager between the customer and the washing machine, the customer actually never interacts with the wash services themselves. A proxy service provided by the wash manager could have been formally defined, but that level of formality in this real-world example has been omitted.

The wash manager in turn does interact with the wash services through their interfaces defined as follows:

— *WashingMachineInterface* is an instance of **ServiceInterface**;

— *TypeOfWash* is an instance of **InformationType**;

— *WashingMachineInterface* has input *TypeOfWash*;

— *BasicWash* has interface *WashingMachineInterface*;

— *GoldWash* has interface *WashingMachineInterface*.

Note how both washing services in fact have the same service interface. Even though Joe has chosen to offer basic wash and gold wash as two different services, both are in effect done by the same washing machine (one simply has to choose the type of wash when initializing the washing machine).

A.5 The Washing Processes

An important part of the car wash system is the car washing processes itself are as follows:

— *AutomatedCarWashProcess* is an instance of both **Process** and **Orchestration**;

— *Wash* is an instance of **Task** and is used by *AutomatedCarWashProcess*;

— *Dry* is an instance of **Task** and is used by *AutomatedCarWashProcess*;

— *AutomatedCarWash* is an instance of **Element** (the automated washing machine) and represents *AutomatedCarWashProcess* (encapsulates the process) as well as directs *AutomatedCarWashProcess*;

— *CarWashProcess* is an instance of **Process** and is used by (part of) *CarWashSystem* (no need to create an explicit building block that is opaque);

— *AutomatedCarWash* is used by *CarWashProcess* (automated activity in the process);

— *WashWindows* is an instance of **Task** and is done by *John*;

— *PreWash* is an instance of **Element**, represents *WashWindows*, and is used by *CarWashProcess* (logical activity in the process);

— *PrewashGuy* is a member of *CarWashProcess* (role in the process);

— *PushWashButton* is an instance of **Task** and is done by *Jack*;

— *InitiateAutomatedWash* is an instance of **Element**, represents *PushWashButton*, and is used by *CarWashProcess* (logical activity in the process);

— *WashOperator* is a member of *CarWashProcess* (role in the process).

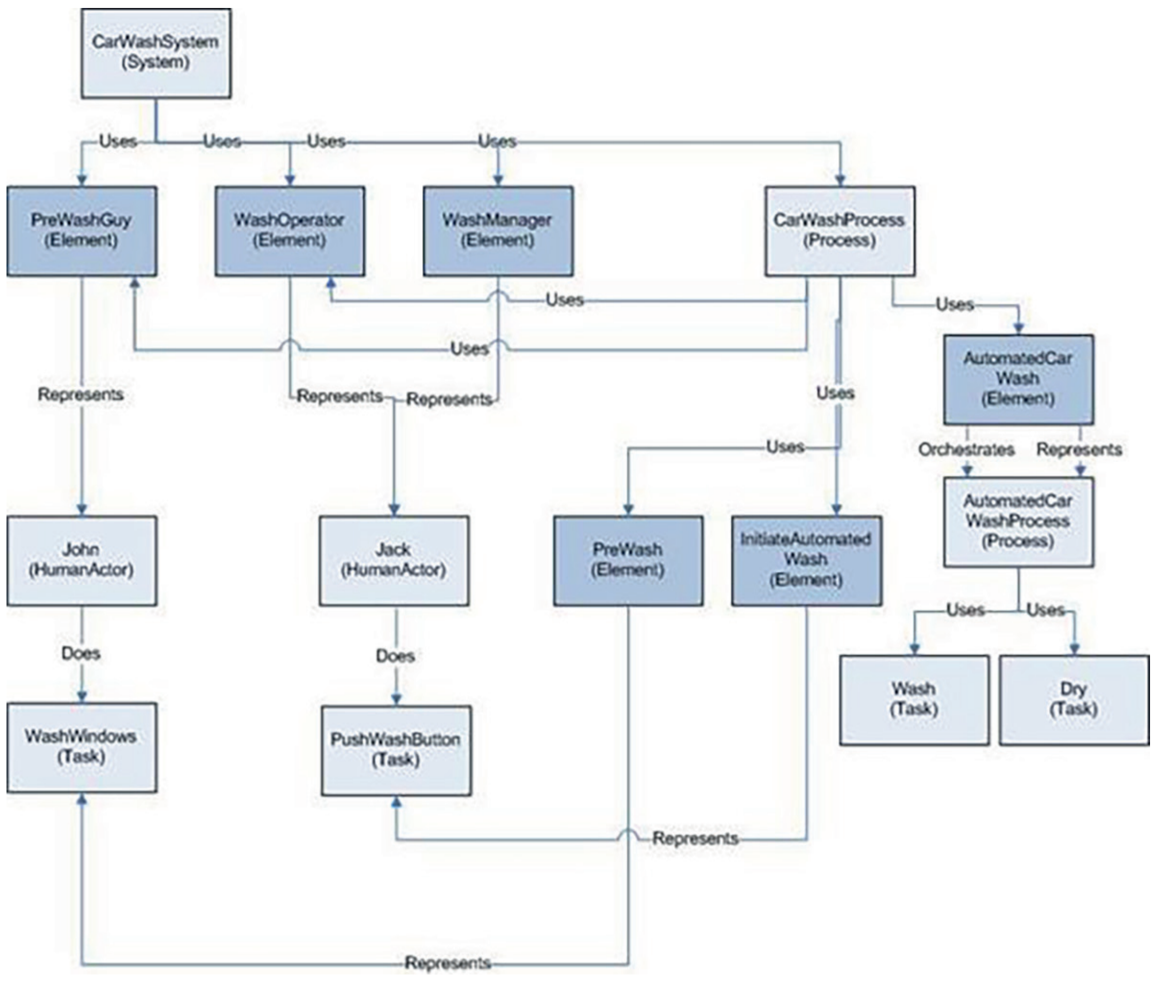

Figure A.3 — Car Wash Example — The Washing Processes

A.5.1 The Washing Policies

Joe sets a payment up-front policy for the washing services defines as follows:

— *PaymentUpFront* is an instance of both **Policy**;

— *PaymentUpFront* is set by *Joe*;

— *PaymentUpFront* applies to both *GoldWash* and *BasicWash*.

Note how the *PaymentUpFront* policy enhances the service contract *BasicWashContract*. While *BasicWashContract* only specifies that *Judy* has to pay $10 for one consumption of the *BasicWash* service, the *PaymentUpFront* policy makes it specific that payment has to happen up-front. One of the advantages of separating policy from service contract is that the payment policy can be changed independently of the service contract. For instance, at some later point in time Joe may decide that recurring customers need not pay up-front, and can institute this change in policy without changing anything else related to *CarWashBusiness*.

Annex B
(informative)

Internet Purchase Example

Jill is purchasing a new TV on the Internet through an online sales site:

— *Jill* is an instance of **Actor** (person).

— *PurchaseTV* is an instance of **Task**.

— *Jill* does *PurchaseTV*.

— *BuyTVOnline* is an instance of **Service**.

— *PurchaseTV* uses *BuyTVOnline*.

OnlineTVSales is the company that is selling TVs:

— *OnlineTVSales* is an instance of **Actor** (organization).

— *BuyTVOnlineContract* is an instance of **ServiceContract** (and describes how to interact with *BuyTVOnline* as well as the legal contract between TV buyer and OnlineTVSales).

— *BuyTVOnline* has contract *BuyTVOnlineContract*.

— *OnlineTVSales* is party to *BuyTVOnlineContract*.

— *Jill* is party to *BuyTVOnlineContract*.

The online site is implemented using web site software:

— *OnlineSalesComponent* is an instance of **Element**.

— *OnlineSalesComponent* performs *OnlineTVSales*.

— *SelectWhatToBuyComponent* is an instance of **Element**.

— *SelectWhatToBuyService* is an instance of **Service**.

— *SelectWhatToBuyComponent* performs *SelectWhatToBuyService*.

— *PayComponent* is an instance of **Element**.

— *PayService* is an instance of **Service**.

— *PayComponent* performs *PayService*.

— *OnlineSalesComponent* is also an instance of **ServiceComposition**.

— *OnlineSalesComponent* uses *SelectWhatToBuyService* and *PayService*.

To complete the purchase transaction, Jill needs to pay for the purchase and then the TV will be delivered:

— *PayForTV* is an instance of **Task**.

— *Jill* does *PayForTV*.

— *PayForTV* uses *BuyTVOnline*.

— *DeliverTV* is an instance of **Task**.
— *OnlineTVSales* does *DeliverTV*.
— *OnlineTVSalesProcess* is an instance of **Process**.
— *OnlineTVSalesProcess* uses *Jill, OnlineTVSales, PurchaseTV, PayForTV*, and *DeliverTV*.

Annex C
(normative)

The OWL Definition of the SOA Ontology

The OWL ontology is available online at: http://www.w3.org/2007/OWL/wiki/OWL_Working_Group

The Ontology is reproduced below.

```
<?xml version="1.0"?>

<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://www.semanticweb.org/ontologies/2010/01/core-soa.owl#"
    xml:base="http://www.semanticweb.org/ontologies/2010/01/core-soa.owl"
>

  <!-- ontology -->
  <owl:Ontology rdf:about=""/>
  <!-- classes -->

  <owl:Class rdf:about="#Event">
    <owl:disjointWith>
      <owl:Class rdf:about="#Policy"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#ServiceContract"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#ServiceInterface"/>
    </owl:disjointWith>
  </owl:Class>

  <owl:Class rdf:about="#InformationType">
    <owl:disjointWith>
      <owl:Class rdf:about="#Policy"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#ServiceContract"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#Effect"/>
    </owl:disjointWith>
  </owl:Class>

  <owl:Class rdf:about="#ServiceComposition">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#Composition"/>
    </rdfs:subClassOf>
    <owl:disjointWith>
      <owl:Class rdf:about="#ServiceContract"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#ServiceInterface"/>
    </owl:disjointWith>
  </owl:Class>

  <owl:Class rdf:about="#Effect">
    <owl:disjointWith>
      <owl:Class rdf:about="#Policy"/>
    </owl:disjointWith>
    <owl:disjointWith>
    <owl:Class rdf:about="#ServiceInterface"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:about="#InformationType"/>
  </owl:disjointWith>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:minCardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#isSpecifiedBy"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:about="#Task">
  <owl:disjointWith>
    <owl:Class rdf:about="#Policy"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:about="#System"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:about="#HumanActor"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:about="#Service"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:about="#ServiceContract"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:about="#ServiceInterface"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:about="#Composition"/>
  </owl:disjointWith>
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Element"/>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#doneBy"/>
      </owl:onProperty>
      <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >0</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:maxCardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#doneBy"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:about="#System">
  <owl:disjointWith>
    <owl:Class rdf:about="#Task"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:about="#Service"/>
  </owl:disjointWith>
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Element"/>
  </rdfs:subClassOf>
```

```
</owl:Class>

<owl:Class rdf:about="#Service">
  <owl:disjointWith>
    <owl:Class rdf:about="#System"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:about="#Task"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:about="#HumanActor"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:about="#ServiceInterface"/>
  </owl:disjointWith>
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Element"/>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:minCardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#hasInterface"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:about="#Policy">
  <owl:disjointWith>
    <owl:Class rdf:about="#InformationType"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:about="#ServiceInterface"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:about="#Element"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:about="#Effect"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:about="#Event"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:about="#ServiceContract"/>
  </owl:disjointWith>
</owl:Class>

<owl:Class rdf:about="#HumanActor">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Element"/>
  </rdfs:subClassOf>
  <owl:disjointWith>
    <owl:Class rdf:about="#Task"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:about="#Service"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:about="#ServiceContract"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:about="#ServiceInterface"/>
  </owl:disjointWith>
</owl:Class>

<owl:Class rdf:about="#Composition">
  <owl:disjointWith>
    <owl:Class rdf:about="#Task"/>
    </owl:disjointWith>
    <rdfs:subClassOf>
      <owl:Class rdf:about="#System"/>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:maxCardinality>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:about="#compositionPattern"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:about="#compositionPattern"/>
        </owl:onProperty>
        <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:minCardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:maxCardinality>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="#orchestratedBy"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >0</owl:minCardinality>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="#orchestratedBy"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>

  <owl:Class rdf:about="#ServiceInterface">
    <owl:disjointWith>
      <owl:Class rdf:about="#Service"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#ServiceContract"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#Effect"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#Policy"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#HumanActor"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#Task"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#ServiceComposition"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#Process"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#Event"/>
    </owl:disjointWith>
    <rdfs:subClassOf>
```

```
      <owl:Restriction>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:about="#constraints"/>
        </owl:onProperty>
        <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:maxCardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:minCardinality>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:about="#constraints"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>

  <owl:Class rdf:about="#Element">
    <owl:disjointWith>
      <owl:Class rdf:about="#Policy"/>
    </owl:disjointWith>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >0</owl:minCardinality>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="#orchestrates"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:maxCardinality>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="#orchestrates"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>

  <owl:Class rdf:about="#ServiceContract">
    <owl:disjointWith>
      <owl:Class rdf:about="#ServiceInterface"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#Policy"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#HumanActor"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#Task"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#ServiceComposition"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#Process"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#Event"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#InformationType"/>
    </owl:disjointWith>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
        <owl:DatatypeProperty rdf:about="#legalAspect"/>
      </owl:onProperty>
      <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:maxCardinality>
      <owl:onProperty>
        <owl:DatatypeProperty rdf:about="#legalAspect"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:DatatypeProperty rdf:about="#interactionAspect"/>
      </owl:onProperty>
      <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:DatatypeProperty rdf:about="#interactionAspect"/>
      </owl:onProperty>
      <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#isContractFor"/>
      </owl:onProperty>
      <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#specifies"/>
      </owl:onProperty>
      <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:about="#Process">
  <owl:disjointWith>
    <owl:Class rdf:about="#ServiceContract"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:about="#ServiceInterface"/>
  </owl:disjointWith>
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Composition"/>
  </rdfs:subClassOf>
</owl:Class>

<!-- object properties -->

<owl:ObjectProperty rdf:about="#isPartyTo">
  <rdfs:domain rdf:resource="#HumanActor"/>
  <rdfs:range rdf:resource="#ServiceContract"/>
</owl:ObjectProperty>
```

```
  <owl:ObjectProperty rdf:about="#involvesParty">
    <owl:inverseOf>
      <owl:ObjectProperty rdf:about="#isPartyTo"/>
    </owl:inverseOf>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#orchestratedBy">
    <rdfs:domain rdf:resource="#Composition"/>
    <rdfs:range rdf:resource="#Element"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#orchestrates">
    <owl:inverseOf>
      <owl:ObjectProperty rdf:about="#orchestratedBy"/>
    </owl:inverseOf>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#isContractFor">
    <rdfs:domain rdf:resource="#ServiceContract"/>
    <rdfs:range rdf:resource="#Service"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#hasContract">
    <owl:inverseOf>
      <owl:ObjectProperty rdf:about="#isContractFor"/>
    </owl:inverseOf>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#setsPolicy">
    <rdfs:domain rdf:resource="#HumanActor"/>
    <rdfs:range rdf:resource="#Policy"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#isSetBy">
    <owl:inverseOf>
      <owl:ObjectProperty rdf:about="#setsPolicy"/>
    </owl:inverseOf>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#generates">
    <rdfs:domain rdf:resource="#Element"/>
    <rdfs:range rdf:resource="#Event"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#generatedBy">
    <owl:inverseOf>
      <owl:ObjectProperty rdf:about="#generates"/>
    </owl:inverseOf>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#represents">
    <rdfs:domain rdf:resource="#Element"/>
    <rdfs:range rdf:resource="#Element"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#representedBy">
    <owl:inverseOf>
      <owl:ObjectProperty rdf:about="#represents"/>
    </owl:inverseOf>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#hasInput">
    <rdfs:domain rdf:resource="#ServiceInterface"/>
    <rdfs:range rdf:resource="#InformationType"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#isInputAt">
    <owl:inverseOf>
      <owl:ObjectProperty rdf:about="#hasInput"/>
    </owl:inverseOf>
```

```
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#doneBy">
  <rdfs:domain rdf:resource="#Task"/>
  <rdfs:range rdf:resource="#HumanActor"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#does">
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#doneBy"/>
  </owl:inverseOf>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#specifies">
  <rdfs:domain rdf:resource="#ServiceContract"/>
  <rdfs:range rdf:resource="#Effect"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#isSpecifiedBy">
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#specifies"/>
  </owl:inverseOf>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#appliesTo">
  <rdfs:domain rdf:resource="#Policy"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#isSubjectTo">
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#appliesTo"/>
  </owl:inverseOf>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#hasInterface">
  <rdfs:domain rdf:resource="#Service"/>
  <rdfs:range rdf:resource="#ServiceInterface"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#isInterfaceOf">
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#hasInterface"/>
  </owl:inverseOf>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#respondsTo">
  <rdfs:domain rdf:resource="#Element"/>
  <rdfs:range rdf:resource="#Event"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#respondedToBy">
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#respondsTo"/>
  </owl:inverseOf>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#performs">
  <rdfs:domain rdf:resource="#Element"/>
  <rdfs:range rdf:resource="#Service"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#performedBy">
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#performs"/>
  </owl:inverseOf>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#uses">
  <rdfs:domain rdf:resource="#Element"/>
  <rdfs:range rdf:resource="#Element"/>
</owl:ObjectProperty>
```

```
  <owl:ObjectProperty rdf:about="#usedBy">
    <owl:inverseOf>
      <owl:ObjectProperty rdf:about="#uses"/>
    </owl:inverseOf>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#hasOutput">
    <rdfs:domain rdf:resource="#ServiceInterface"/>
    <rdfs:range rdf:resource="#InformationType"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#isOutputAt">
    <owl:inverseOf>
      <owl:ObjectProperty rdf:about="#hasOutput"/>
    </owl:inverseOf>
  </owl:ObjectProperty>

  <!-- datatype properties -->

  <owl:DatatypeProperty rdf:about="#legalAspect">
    <rdfs:domain rdf:resource="#ServiceContract"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="#constraints">
    <rdfs:domain rdf:resource="#ServiceInterface"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="#compositionPattern">
    <rdfs:domain rdf:resource="#Composition"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="#interactionAspect">
    <rdfs:domain rdf:resource="#ServiceContract"/>
  </owl:DatatypeProperty>

</rdf:RDF>
```

Annex D
(informative)

Class Relationship Matrix

This Annex contains a class relationship matrix that illustrates the class-to-class relationships intrinsic in the OWL definitions of the SOA ontology. The matrix is deterministically derived from the ontology OWL definitions. Each row X and each column Y corresponds to an OWL class. A relation appears in cell (X,Y), if and only if, class X is part of the domain and class Y is part of the range of the corresponding OWL property. Note that this means that datatype properties (which do not have a range) are not included in the class relationship matrix.

As outlined in the body of the document, there are four relationships in the table (plus their inverses and sub-classed derivatives) that are technically allowed according to the OWL definitions, but would not be expected to occur in a practical application of the ontology. Specifically, services are not expected to perform services, services are not expected to use elements (directly), services are not expected to represent elements, and services are not expected to orchestrate compositions, all due to the **Service** class being defined as a logical representation of a set of activitiesy; see 10.3, 10.5.1, 10.5.2 and 11.3.2 for details.

Table D.1

	Element	System	Service	Human Actor	Task	Composition	Process	Service composition	Service contract	Effect	Service interface	Information type	Event	Policy	Thing
Element	uses usedBy represents representedBy	uses usedBy represents representedBy	uses usedBy represents representedBy performs	uses usedBy represents representedBy	uses usedBy represents representedBy	uses usedBy represents representedBy orchestrates	uses usedBy represents representedBy orchestrates	uses usedBy represents representedBy orchestrates					generates respondsTo	isSubjectTo	
System	uses usedBy represents representedBy	uses usedBy represents representedBy	uses usedBy represents representedBy performs	uses usedBy represents representedBy	uses usedBy represents representedBy	uses usedBy represents representedBy orchestrates	uses usedBy represents representedBy orchestrates	uses usedBy represents representedBy orchestrates					generates respondsTo	isSubjectTo	
Service	Uses usedBy represents representedBy performedBy	uses usedBy represents representedBy performedBy	uses usedBy represents representedBy performs performedBy	Uses usedBy represents representedBy performedBy	uses usedBy represents representedBy performedBy	uses usedBy represents representedBy performedBy orchestrates	uses usedBy represents representedBy performedBy orchestrates	uses usedBy represents representedBy performedBy orchestrates	hasContract		hasInterface		generates respondsTo	isSubjectTo	
Human Actor	Uses usedBy represents representedBy	uses usedBy represents representedBy	uses usedBy represents representedBy performs	uses usedBy represents representedBy	uses usedBy represents representedBy does	uses usedBy represents representedBy orchestrates	uses usedBy represents representedBy orchestrates	uses usedBy represents representedBy orchestrates	isPartyTo				generates respondsTo	setsPolicy isSubjectTo	
Task	Uses usedBy represents representedBy	uses usedBy represents representedBy	uses usedBy represents representedBy performs	uses usedBy represents representedBy doneBy	uses usedBy represents representedBy	uses usedBy represents representedBy orchestrates	uses usedBy represents representedBy orchestrates	uses usedBy represents representedBy orchestrates					generates respondsTo	isSubjectTo	
Composition	uses usedBy represents representedBy orchestratedBy	uses usedBy represents representedBy orchestratedBy	uses usedBy represents representedBy performs orchestratedBy	uses usedBy represents representedBy orchestratedBy	uses usedBy represents representedBy orchestratedBy	uses usedBy represents representedBy orchestrates orchestratedBy	uses usedBy represents representedBy orchestrates orchestratedBy	uses usedBy represents representedBy orchestrates orchestratedBy					generates respondsTo	isSubjectTo	

Table D.1 *(continued)*

	Element	System	Service	Human Actor	Task	Composition	Process	Service composition	Service contract	Effect	Service interface	Information type	Event	Policy	Thing
Process	uses usedBy represents representedBy orchestratedBy	uses usedBy represents representedBy orchestratedBy	uses usedBy represents representedBy performs orchestratedBy	uses usedBy represents representedBy orchestratedBy	uses usedBy represents representedBy orchestratedBy	uses usedBy represents representedBy orchestrates orchestratedBy	uses usedBy represents representedBy orchestrates orchestratedBy	uses usedBy represents representedBy orchestrates orchestratedBy					generates respondsTo	isSubjectTo	
Service Composition	uses usedBy represents representedBy orchestratedBy	uses usedBy represents representedBy orchestratedBy	uses usedBy represents representedBy performs orchestratedBy	uses usedBy represents representedBy orchestratedBy	uses usedBy represents representedBy orchestratedBy	uses usedBy represents representedBy orchestrates orchestratedBy	uses usedBy represents representedBy orchestrates orchestratedBy	uses usedBy represents representedBy orchestrates orchestratedBy					generates respondsTo	isSubjectTo	
Service Contract			isContractFot	involvesParty						specifies				isSubjectTo	
Effect									isSpecifiedBy					isSubjectTo	
Service Interface			isInterfaceOf									hasInput hasOutput		isSubjectTo	
Information Type											isInputAt isOutputAt			isSubjectTo	
Event	generatedBy respondedToBy	generatedBy respondedToBy	generatedBy respondedToBy	generatedBy respondedToBy	generatedBy respondedToBy	generatedBy respondedToBy	generatedBy respondedToBy	generatedBy respondedToBy						isSubjectTo	
Policy	appliesTo	appliesTo	appliesTo	isSetBy appliesTo	appliesTo	appliesTo	appliesTo	appliesTo	appliesTo	appliesTo	appliesTo	appliesTo	appliesTo	appliesTo isSubjectTo	appliesTo
Thing														isSubjectTo	

Annex E
(informative)

Terms Mapping Between the SOA RA Parts

This Annex contains a table which maps the definitions in the parts of ISO/IEC 18384. In particular, the definitions in ISO/IEC 18384-1 to the ontology terms in ISO/IEC 18384-3 to highlight where more information on each of the terms and concepts can be found as well as illustrate where the definitions are aligned or deviate. If there is a deviation between the definitions, then an analysis of that definition and the difference is in italics. The third column contains the clauses in ISO/IEC 18384-2 where the concepts are defined, discussed, or referenced. The final column indicates where terms were defined if they were defined elsewhere. If there is no concept in ISO/IEC 18384-3 or ISO/IEC 18384-4 then 'n/a' is in the column.

ISO/IEC 18384-1	ISO/IEC 18384-3	ISO/ IEC 18384-2	Other references
Clause and definition *Analysis in italics where there is a difference*	**Clause and definition/discussion** *Analysis in italics where there is a difference* n/a indicates definition not found in ISO/IEC 18384-3	**Clause numbers where concept is discussed** n/a indicates definition not found in **ISO/ IEC 18384-2**	
3.1 actor (ISO/IEC 16500-8:1999, 3.1) person or system component that interacts with the system as a whole and that provides stimulus which invokes actions	n/a	Discussed in 4.5, Clause 9, Clause 11, Clause 12	(ISO/IEC 16500-8:1999, 3.1) BPMN
3.2 architecture (ISO/IEC/IEEE 42010:2011, 3.2). fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution	n/a	Discussed in Clause 4	(ISO/IEC/ IEEE 42010:2011, 3.2).
3.3 choreography type of composition (3.5) whose elements (3.8) interact in a non-directed fashion with each autonomous part knowing and following an observable predefined pattern of behaviour for the entire (global) composition *Observable characteristic was added to Ontology definition*	11.3.3 In a choreography (a composition whose composition pattern is a choreography) the elements used by the composition interact in a non-directed fashion, yet with each autonomous member knowing and following a predefined pattern of behaviour for the entire composition.	Discussed in Clause 8	

ISO/IEC 18384-1	ISO/IEC 18384-3	ISO/ IEC 18384-2	Other references
Clause and definition *Analysis in italics where there is a difference*	**Clause and definition/discussion** *Analysis in italics where there is a difference* n/a indicates definition not found in ISO/IEC 18384-3	**Clause numbers where concept is discussed** n/a indicates definition not found in **ISO/ IEC 18384-2**	
3.4 collaboration type of composition (3.5) whose elements (3.8) interact in a non-directed fashion, each according to their own plans and purposes without a predefined pattern of behaviour	11.3.4 In a collaboration (a composition whose composition pattern is a collaboration) the elements used by the composition interact in a non-directed fashion, each according to their own plans and purposes without a predefined pattern of behaviour.	Discussed in in Clause 4, Clause 8 Collaboration services in 15 Also used in English sense	
3.5 composition result of assembling a collection of elements for a particular purpose	11.2 A composition is the result of assembling a collection of things for a particular purpose.	Discussed in in Clause 4, Clause 5, Clause 8, Clause 9, Clause 10, Clause 11, Clause 12, Clause 15 Also used in English sense	
3.6 endpoint location at which information is received to invoke and configure interaction	n/a	Discussed in Clause 4, Clause 10	
3.7 effect outcome of an interaction with a service (3.20)	10.10 Interacting with something performing a service has *effects*. These comprise the outcome of that interaction, and are how a service (through the element that performs it) delivers value to its consumers.	Discussed in Clause 4, Clause 6, Clause 11	

ISO/IEC 18384-1	ISO/IEC 18384-3	ISO/ IEC 18384-2	Other references
Clause and definition *Analysis in italics where there is a difference*	**Clause and definition/discussion** *Analysis in italics where there is a difference* n/a indicates definition not found in ISO/IEC 18384-3	**Clause numbers where concept is discussed** n/a indicates definition not found in **ISO/ IEC 18384-2**	
3.8 element unit that is indivisible at a given level of abstraction and has a clearly defined boundary	8.2 An *element* is an entity that is opaque and indivisible at a given level of abstraction. The element has a clearly defined boundary.	Discussed in Clause 4, and throughout Also used in English sense	
3.9 entity individual element (3.8) in a system with an identity which can act as a service provider (3.49) or service consumer	n/a just used in definition of element and quote from Bibliography 4	Discussed in Clause 4, Clause 15	SoaML
3.10 event something that occurs to which an element may choose to respond	13.2 An *event* is something that happens, to which an element may choose to respond. Events can be responded to by any element. Similarly, events may be generated (emitted) by any element	Discussed in Clause 4, Clause 8, Clause 10, Clause 11, Clause 12	
3.11 execution context set of technical and business elements (3.8) needed by those with needs and capabilities to permit service providers (3.49) and service consumers (3.29)	n/a	n/a	SOA RM
3.12 human actor actor (3.1) restricted to a person or an organizational entity (3.9)	9.2 Human Actor A *human actor* is a person or an organization. disjoint with the *Service* and *Task* classes	Discussed in Clause 4, Clause 9	

ISO/IEC 18384-1	ISO/IEC 18384-3	ISO/ IEC 18384-2	Other references
Clause and definition *Analysis in italics where there is a difference*	**Clause and definition/discussion** *Analysis in italics where there is a difference* n/a indicates definition not found in ISO/IEC 18384-3	**Clause numbers where concept is discussed** n/a indicates definition not found in **ISO/ IEC 18384-2**	
3.13 human task task which is done by Human Actor (3.12)	*Human task not defined, task is equivalent* 9.4 task A *task* is an atomic action which accomplishes a defined result. Tasks are done by people or organizations, specifically by instances of **HumanActor**	Discussed in Clause 8	
3.14 Interface A shared boundary between two functional units, defined by various characteristics pertaining to the functions, physical interconnections, signal exchanges, and other characteristics, as appropriate [SOURCE: ISO/ IEC 2382-1:1993, 01.01.38]	10.13 *discusses interface: (consistent)* The concept of an interface is in general well understood by practitioners, including the notion that interfaces define the parameters for information going in and out of them when invoked. What differs from domain to domain is the specific nature of how an interface is invoked and how information is passed back and forth. From a design perspective interfaces may have more granular operations or may be composed of other interfaces;	Discussed in Clause 4, Clause 7, Clause 9, Clause 14	[SOURCE: ISO/IEC 2382-1:1993, 01.01.38]

ISO/IEC 18384-1	**ISO/IEC 18384-3**	**ISO/ IEC 18384-2**	**Other references**
Clause and definition *Analysis in italics where there is a difference*	**Clause and definition/discussion** *Analysis in italics where there is a difference* n/a indicates definition not found in ISO/IEC 18384-3	**Clause numbers where concept is discussed** n/a indicates definition not found in **ISO/ IEC 18384-2**	
3.15 loose coupling principle where dependencies between services are minimized	n/a 10.13 *discusses loose coupling casually but consistently* Service interfaces are typically, but not necessarily, message-based (to support loose coupling). Furthermore, service interfaces are always defined independently from any service implementing them (to support loose coupling and service mediation).	Discussed in Clause 4, Clause 6, Clause 10,	
3.16 orchestration type of composition (3.5) where one particular element (3.8) is used by the composition to oversee and direct the other elements	11.3 In an orchestration (a composition whose composition pattern is an orchestration), there is one particular element used by the composition that oversees and directs the other elements. Note that the element that directs an orchestration by definition is different than the orchestration (**Composition** instance) itself.	Discussed in Clause 4, Clause 8, Clause 9, Clause 10	

ISO/IEC 18384-1	ISO/IEC 18384-3	ISO/ IEC 18384-2	Other references
Clause and definition *Analysis in italics where there is a difference*	**Clause and definition/discussion** *Analysis in italics where there is a difference* n/a indicates definition not found in ISO/IEC 18384-3	**Clause numbers where concept is discussed** n/a indicates definition not found in **ISO/ IEC 18384-2**	
3.17 policy statement that an entity (3.9) intends to follow or intends that another entity should follow	12 A policy is a statement of direction that a human actor may intend to follow or may intend that another human actor should follow. Policies can apply to any element in a system. *Ontology is narrower than Part1, allowing only human actors to define/ follow policy.*	Discussed in Clause 4, Clause 11, Clause 13, Clause 14	
3.18 process type of composition (3.5) whose elements (3.8) are composed into a sequence or flow of activities and interactions with the objective of carrying out certain work	11.6 A process is a composition whose elements are composed into a sequence or flow of activities and interactions with the objective of carrying out certain work.	Discussed in Clause 4, Clause 8, Clause 11, Clause 13, Clause 14 Also processing in the English sense	
3.19 real world effect change relevant to and experienced by specific stakeholders (See Reference [6])	n/a *equivalent to 'effect'* effect: Interacting with something performing a service has *effects*. These comprise the outcome of that interaction, and are how a service (through the element that performs it) delivers value to its consumers	Discussed in Clause 4, Clause 6	SOA RM

ISO/IEC 18384-1	ISO/IEC 18384-3	ISO/ IEC 18384-2	Other references
Clause and definition *Analysis in italics where there is a difference*	**Clause and definition/discussion** *Analysis in italics where there is a difference* n/a indicates definition not found in ISO/IEC 18384-3	**Clause numbers where concept is discussed** n/a indicates definition not found in **ISO/ IEC 18384-2**	
3.20 service logical representation of a set of activities that has specified outcomes, is self-contained, may be composed of other services, and is a “black box” to consumers of the service (see ISO/IEC 18384-3:—, 7.2)	10.2 A service is a logical representation of a set of activities that has specified outcomes, is self-contained, may be composed of other services, and is a “black box” to consumers of the service.	Discussed in Clause 4, rest of document	
3.21 service broker Element that enables the communication with services (3.20), either at a business level or at the implementation level, i.e with intermediaries	n/a	n/a	
3.22 service bus design and runtime pattern for enabling service (3.20) interactions, such as communication, access, consumption, transformation, intermediaries, and message routing	Discussed in Clause 7, Clause 11, Clause 13	n/a	
3.23 service candidate services (3.20) identified during the SOA lifecycle (2.1.58) that meet broad service requirements, and from which one or more are selected for further development as part of an overall SOA solution (3.56)	n/a	Discussed in Clause 8	
3.24 service registry/repository service catalogue Logical collection of service descriptions (3.31) and related artifacts that supports publication, registration, search, management and retrieval of those artifacts	n/a	Discussed in Clause 4, Clause 9, Clause 12, Clause 13	
3.25 service choreography choreography (3.3) whose elements (3.8) are services (3.20) (see ISO/IEC 18384-3:—, Clause 8)	*Clause 11 discussed — consistent* *No specific definition but extension to service composition – then it is ‘the result of assembling a collection of services’ so consistent with choreography of services*	Discussed in Clause 4, Clause 8, Clause 15 Discusses choreography of services	

ISO/IEC 18384-1	**ISO/IEC 18384-3**	**ISO/ IEC 18384-2**	**Other references**
Clause and definition *Analysis in italics where there is a difference*	**Clause and definition/discussion** *Analysis in italics where there is a difference* n/a indicates definition not found in ISO/IEC 18384-3	**Clause numbers where concept is discussed** n/a indicates definition not found in **ISO/ IEC 18384-2**	
3.26 service collaboration collaboration (3.5) whose elements (3.8) are services (3.20) (see ISO/IEC 18384-3:—, Clause 8)	*Clause 11 discussed — consistent* *No specific definition but extension to service composition — then it is 'the result of assembling a collection of services' so consistent with collaboration of services*	Discussed in Clause 4, Clause 8, Clause 15 Discusses collaboration of services Collaboration in the English sense	
3.27 service component element (3.8) that implements services (3.20)	n/a	Discussed in Clause 4, Clause 6, Clause 7, Clause 8, Clause 15	
3.28 service composition composition (3.5) that provide (in the operational sense) higher level services (3.20) that are only composed of other services	11.5 A key SOA concept is the notion of *service composition*, the result of assembling a collection of services in order to perform a new higher-level service.	Discussed in Clause 4, Clause 5, Clause 8	
3.29 service consumer entity (3.9) that uses services (3.20)	10.4 that some element uses (consumes) a service	Discussed in Clause 4, Clause 8, Clause 9, Clause 10	
3.30 service contract terms, conditions, and interaction rules that interacting service consumers (3.29) and service providers (3.49) agree to (directly or indirectly) *Part 1 restricts the participants*	10.6 A *service contract* defines the terms, conditions, and interaction rules that interacting participants agree to (directly or indirectly).	Discussed in Clause 4, Clause 11, Clause 13, Clause 14	

ISO/IEC 18384-1	**ISO/IEC 18384-3**	**ISO/ IEC 18384-2**	**Other references**
Clause and definition *Analysis in italics where there is a difference*	**Clause and definition/discussion** *Analysis in italics where there is a difference* n/a indicates definition not found in ISO/IEC 18384-3	**Clause numbers where concept is discussed** n/a indicates definition not found in **ISO/ IEC 18384-2**	
3.31 service description information needed in order to use, or consider using, a service (3.20)	n/a *used in quote from bibliography 10.1* "A mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description."	Discussed in Clause 4, Clause 6, Clause 7, Clause 14	
3.32 service deployment activities by which implementations of services (3.20) are made able to run in a specific hardware and software environment and usable by service consumers (3.29)	n/a	Discussed in Clause 4, Clause 6, Clause 14	
3.33 service development activities by which needs and constraints are identified and services are designed as part of a SOA solution (3.56) in order to address those needs within the constraints	n/a	Discussed in Clause 6, Clause 14	
3.34 service implementation activities performing technical development and the physical implementation of the service (3.20) that is part of a service lifecycle (3.40), and results in the creation of a service component (3.27)	n/a	Discussed in Clause 4, Clause 5, Clause 6, Clause 14, Clause 15	
3.35 service discovery activities by which a service consumer (3.29) can find services which meet their specific functional and/or non-functional requirements	n/a	Discussed in Clause 7, Clause 8, Clause 10, Clause 13	
3.36 service governance strategy and control mechanism that applies across the service lifecycle (3.40) and service portfolio, which includes the establishment of chains of responsibility, driving monitoring of compliance with policies by providing appropriate processes and measurements as part of SOA solution governance (3.57)	n/a	Discussed in Clause 13	

ISO/IEC 18384-1	**ISO/IEC 18384-3**	**ISO/ IEC 18384-2**	**Other references**
Clause and definition *Analysis in italics where there is a difference*	**Clause and definition/discussion** *Analysis in italics where there is a difference* n/a indicates definition not found in ISO/IEC 18384-3	**Clause numbers where concept is discussed** n/a indicates definition not found in **ISO/ IEC 18384-2**	
3.37 service interaction activity involved in making use of a capability offered, usually across an ownership boundary, in order to achieve a particular desired real-world effect (3.18) (see Reference [6])	*10.8 discusses interaction aspects — not in conflict with service interaction, but not defined* Interaction aspects Anyone wanting to use a service obeys the interaction aspects (as defined in the **interactionAspect** datatype property) of any service contract applying to that interaction. In that fashion, the interaction aspects of a service contract are context-independent; they capture the defined or intrinsic ways in which a service may be used.	Discussed in Clause 4, Clause 7, Clause 10	SOA RM
3.38 service interface interface by which other elements (3.8) can interact and exchange information with the service where the form of the request and the outcome of the request is in the service description (see ISO/IEC 18384-3:—, 7.13)	10.13 A service interface defines the way in which other elements can interact and exchange information with a service.	Discussed in Clause 4, Clause 9, Clause 10, Clause 14	
3.39 service interoperability ability of service providers (3.49) and service consumers (3.29) to communicate, invoke services (3.20) and exchange information at both the syntactic and semantic level leading to effects as defined by the service description (3.31)	n/a	Discussed in Clause 4, Clause 6, Clause 10, Clause 14	
3.40 Service Level Agreement type of service contract (3.30) that defines measureable conditions of interactions between a service provider (3.49) and a service consumer (3.29)	n/a	Discussed in Clause 4, Clause 11, Clause 13, Clause 14	

ISO/IEC 18384-1	ISO/IEC 18384-3	ISO/ IEC 18384-2	Other references
Clause and definition *Analysis in italics where there is a difference*	**Clause and definition/discussion** *Analysis in italics where there is a difference* n/a indicates definition not found in ISO/IEC 18384-3	**Clause numbers where concept is discussed** n/a indicates definition not found in **ISO/ IEC 18384-2**	
3.41 service lifecycle set of phases for realizing a service (3.20) from conception and identification to instantiation and retirement	n/a	Discussed in Clause 4, Clause 11, Clause 13, Clause 14	
3.42 service management monitoring, controlling, maintaining, optimizing, and operating services (3.20)	n/a	Discussed in Clause 4, Clause 11, Clause 13	
3.43 service modelling set of activities to develop a series of service candidates (3.23) for functions or actions on a SOA solution (3.57) using service oriented analysis processes (3.46)	n/a	Discussed in Clause 4, Clause 11, Clause 12, Clause 13	
3.44 service monitoring tracking state and operational conditions related to service (3.20) execution, performance, and real world effects (3.18)	n/a	Discussed in Clause 4, Clause 11, Clause 14	
3.45 service orchestration orchestration (3.15) where the orchestrated elements (3.8) are services (3.20)	*Clause 11 discussed — consistent* *No specific definition but extension to service composition — then it is 'the result of assembling a collection of services' so consistent with orchestration of services*	Discussed in Clause 4, Clause 8, Clause 10	
3.46 service orientation approach to designing systems in terms of services (3.20) and service-based development	n/a	Discussed in Clause 4	
3.47 service oriented analysis preparatory information gathering steps that are completed in support of a service modelling sub-process that results in the creation of a set of services (3.20)	n/a	n/a	

ISO/IEC 18384-1	**ISO/IEC 18384-3**	**ISO/ IEC 18384-2**	**Other references**
Clause and definition *Analysis in italics where there is a difference*	**Clause and definition/discussion** *Analysis in italics where there is a difference* n/a indicates definition not found in ISO/IEC 18384-3	**Clause numbers where concept is discussed** n/a indicates definition not found in **ISO/ IEC 18384-2**	
3.48 service oriented architecture architectural style that supports service orientation (3.45) and is a paradigm for building business solutions	Introduction Service oriented architecture (SOA) is an architectural style in which business and IT systems are designed in terms of services available at an interface and the outcomes of these services. A service is a logical representation of a set of activities that has specified outcomes and is self-contained, it may be composed of other services but consumers of the service need not be aware of any internal structure.	Discussed in introduction, Clause 4 and throughout	

ISO/IEC 18384-1	ISO/IEC 18384-3	ISO/ IEC 18384-2	Other references
Clause and definition *Analysis in italics where there is a difference*	**Clause and definition/discussion** *Analysis in italics where there is a difference* n/a indicates definition not found in ISO/IEC 18384-3	**Clause numbers where concept is discussed** n/a indicates definition not found in **ISO/ IEC 18384-2**	
3.49 service policy policy as applied to a service (3.20)	*Service policy is discussed but not explicitly defined – consistently used that it applies to a service and is separate from a contract* 12.2 Policy A *policy* is a statement of direction that a human actor may intend to follow or may intend that another human actor should follow. provider of a service has a policy for the service, a policy for a service is not necessarily owned by the provider. One of the advantages of separating policy from service contract is that the payment policy can be changed independently of the service contract.	Discussed in Clause 4, Clause 7, Clause 11, Clause 13	
3.50 service provider entity providing services (3.20) *Part 1 restricts provider to an entity*	10.4 some element performs (provides) a service	Discussed in Clause 4, Clause 7, Clause 10, Clause 13, Clause 14	
3.51 service publishing / service registration cataloguing of service descriptions in an accessible location, such as a service registry/ repository (3.74), where supporting activities, such as search and retrieval of descriptions, make service information visible and available to potential service consumers (3.29)	n/a	Discussed in Clause 4, Clause 6, Clause 8, Clause 14, Clause 15 Some discussion on publishing events	

ISO/IEC 18384-1	**ISO/IEC 18384-3**	**ISO/ IEC 18384-2**	**Other references**
Clause and definition *Analysis in italics where there is a difference*	**Clause and definition/discussion** *Analysis in italics where there is a difference* n/a indicates definition not found in ISO/IEC 18384-3	**Clause numbers where concept is discussed** n/a indicates definition not found in **ISO/ IEC 18384-2**	
3.52 SOA implementation methods and techniques used to develop SOA (3.47) based solutions	n/a	n/a	
3.53 SOA maturity assessment of an organization's ability to adopt SOA (3.47) and the current level of adoption	n/a	Discussed in Clause 4, Clause 13	
3.54 SOA maturity model framework stating overall objectives and a method to evaluate an organisations' SOA maturity (3.53) against these objectives	n/a	n/a	
3.55 SOA resource elements (3.8) that provide the IT resources used by services (3.20)	n/a	n/a	
3.56 SOA solution solutions, in part or as a whole, implemented by applying SOA (3.47) principles, concepts, methods, and techniques	n/a (used in introduction consistently)	Discussed in Clause 4, Clause 5, Clause 8, Clause 9, Clause 10, Clause 11, Clause 13, Clause 14, Clause 15	
3.57 SOA solution governance specialization of IT governance specifically focused on management strategies and mechanisms for the end users' specific SOA solution (3.56)	n/a	Discussed in Clause 4, Clause 13	
3.58 SOA solution lifecycle set of activities for engineering SOA solutions (3.56) , including analysis, design, implementation, deployment, test and management	n/a	Discussed in Clause 4, Clause 11	

ISO/IEC 18384-1	**ISO/IEC 18384-3**	**ISO/ IEC 18384-2**	**Other references**
Clause and definition *Analysis in italics where there is a difference*	**Clause and definition/discussion** *Analysis in italics where there is a difference* n/a indicates definition not found in ISO/IEC 18384-3	**Clause numbers where concept is discussed** n/a indicates definition not found in **ISO/ IEC 18384-2**	
3.59 SOA solution management measurement, monitoring, and configuration of the entire lifecycle of a SOA solution (3.56)	n/a	Discussed in Clause 4, Clause 11	
3.60 task atomic action which accomplishes a defined result (see 18384-3 6.4)	9.4 A task is an atomic action which accomplishes a defined result.	Discussed in Clause 4, Clause 8, Clause 15 *8.1.2: has task decomposition, which would not be consistent with atomic*	BPMN 2.0
3.61 Web Services software system designed to support interoperable machine-to-machine interaction over a network	n/a	Discussed in Clause 4	

Bibliography

[1] ISO/IEC 19505-2, *Information technology — Object Management Group Unified Modeling Language (OMG UML) — Part 2: Superstructure*

[2] ISO/IEC/TR 24800-1:2007, *Information technology — JPSearch — Part 1: System framework and components*

[3] OASIS. *Reference Model for SOA*, Version 1.0, OASIS Standard, October 2006: Available from World Wide Web: http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf

[4] OMG. *Business Process Management Notation* (BPMN), see http://www.omg.org/spec/BPMN/2.0/

[5] ISO Technical Report TR9007, *Concepts and Terminology for the Conceptual Schema and the Information Base*

[6] OASIS. *Reference Architecture for SOA Foundation,* Version 1.0, OASIS Public Review Draft 1, April 2008: see docs.oasis-open.org/soa-rm/soa-ra/v1.0/soa-ra-pr-01.pdf

[7] OMG. *Model Driven Architecture (MDA) Guide,* Version 1.0.1, Object Management Group (OMG), June 2003: see www.omg.org/docs/omg/03-06-01.pdf

[8] OMG. *Unified Modeling Language* (OMG UML), Superstructure, Version 2.2, OMG Doc. No.: formal/2009-02-02, Object Management Group (OMG), February 2009: see www.omg.org/spec/UML/2.2/Superstructure

[9] OMG. *SOA Modeling Language* (OMG SoaML) Specification for the UML Profile and Metamodel for Services (UPMS), Revised Submission, OMG Doc. No.: ad/2008-11-01, Object Management Group (OMG), November 2008: see www.omg.org/cgi-bin/doc?ad/08-11-01

[10] OWL. *Web Ontology Language,* World Wide Web Consortium (W3C), February 2004: see www.w3.org/TR/owl-ref

[11] Beyond Concepts: Ontology as Reality Representation, by Barry Smith; available from http://ontology.buffalo.edu/bfo/BeyondConcepts.pdf.

[12] Std I.E.E.E. 1471-2000: IEEE Recommended Practice for Architectural Description of Software-intensive Systems (also published as ISO/IEC 42010: 2007); available from standards.ieee.org.

[13] ISO/IEC 42010: 2007, *Systems and Software Engineering – Recommended Practice for Architectural Description of Software-intensive Systems*; available from www.iso.org.

[14] What is an Ontology? Stanford University; available from www-ksl.stanford.edu/kst/what-is-an-ontology.html.

[15] OWL 2 *Web Ontology Language (Second Edition),* World Wide Web Consortium (W3C), December 2012: see http://www.w3.org/TR/owl-overview/

[16] OWL *Web Ontology Language Reference*, W3C Recommendation, 10 February 2004, World-Wide Web Consortium; available from www.w3.org/TR/owl-ref.

ICS 35.100.05

Price based on 75 pages