

1 **Document Type:** Other Document (Defined)
2
3 **Document Title:** ISO/IEC WD 1 18384 Part 3, **Distributed Application Platforms and**
4 **Services (DAPS) – SOA Ontology**
5
6 **Source:** **SC38** WG2
7
8
9 **Document Status:** **Working Draft 1**
10
11
12
13
14 **Action ID:** **Attached to Ballot for comment on Working Draft**
15

16

17 Secretariat, ISO/IEC JTC 1/SC 38, American National Standards Institute, 25 West 43rd Street, New York, NY 10036;
18 Telephone: 1 212 642 4904; Facsimile: 1 212 840 2298; Email: mpeacock@ansi.org

19

20

21

22

23

24

25

26

27

28

29

Reference number of working document: **ISO/IEC JTC 1/SC 38 N 782**

30

Date: 2011-10-21

31

Reference number of document: **ISO/WD 18384 Part 3**

32

Committee identification: **ISO/IEC JTC 1/SC 38/WG 2**

33

Secretariat: **ANSI**

34

**Distributed Application Platforms and Services (DAPS) –Reference
Architecture for Service Oriented Architecture (SOA) Part 3 – Service-
Oriented Architecture Ontology**

35

36

37

38

Warning

39

This document is not an ISO International Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an International Standard.

40

41

Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

42

Document type: Working Draft
Document subtype: if applicable
Document stage: (20) Preparation
Document language: E

44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63

Copyright notice

This ISO document is a working draft or committee draft and is copyright-protected by ISO. While the reproduction of working drafts or committee drafts in any form for use by participants in the ISO standards development process is permitted without prior permission from ISO No comments on this Annex were requested, processed, or addressed in the development of this Annex and TR, therefore was no consensus developed on this annex., neither this document nor any extract from it may be reproduced, stored or transmitted in any form for any other purpose without prior written permission from ISO.

Requests for permission to reproduce this document for the purpose of selling it should be addressed as shown below or to ISO's member body in the country of the requester:

*[Indicate :
the full address
telephone number
fax number
telex number
and electronic mail address*

as appropriate, of the Copyright Manager of the ISO member body responsible for the secretariat of the TC or SC within the framework of which the draft has been prepared]

Reproduction for sales purposes may be subject to royalty payments or a licensing agreement.

Violators may be prosecuted.

64	Contents		Page
65	Foreword		5
66	Introduction		6
67	1 Scope		8
68	2 Normative references		8
69	3 Terms, Definitions, Notations, and Conventions		8
70	3.1 Definitions		8
71	3.2 Acronyms		9
72	3.3 Notations		9
73	3.4 Conventions		9
74	4 SOA Ontology Overview		9
75	4.1.1 Applications		10
76	4.1.2 Conformance		11
77	5 System and Element		11
78	5.1 Introduction		11
79	5.2 The Element Class		12
80	5.3 The uses and usedBy Properties		13
81	5.4 Element – Organizational Example		14
82	5.5 The System Class		14
83	5.6 System – Examples		16
84	5.6.1 Organizational Example		16
85	5.6.2 Service Composition Example		16
86	5.6.3 Car Wash Example		16
87	5.7 The represents and representedBy Properties		16
88	5.8 Examples		18
89	5.8.1 Organizational Example		18
90	5.8.2 Car Wash Example		19
91	6 HumanActor and Task		19
92	6.1 Introduction		19
93	6.2 The HumanActor Class		20
94	6.3 HumanActor – Examples		21
95	6.3.1 The uses and usedBy Properties Applied to HumanActor		21
96	6.3.2 The represents and representedBy Properties Applied to HumanActor		21
97	6.3.3 Organizational Example		21
98	6.3.4 Car Wash Example		21
99	6.4 The Task Class		22
100	6.5 The does and doneBy Properties		23
101	6.6 Task – Examples		24
102	6.6.1 The uses and usedBy Properties Applied to Task		24
103	6.6.2 The represents and representedBy Properties Applied to Task		24
104	6.6.3 Organizational Example		24
105	6.6.4 Car Wash Example		25

106	7	Service, ServiceContract, and ServiceInterface	25
107	7.1	Introduction	25
108	7.2	The Service Class	26
109	7.3	The performs and performedBy Properties	27
110	7.3.1	Service Consumers and Service Providers	28
111	7.4	Service – Examples	28
112	7.4.1	The uses and usedBy Properties Applied to Service	28
113	7.4.2	The represents and representedBy Properties Applied to Service	29
114	7.4.3	Exemplifying the Difference between Doing a Task and Performing a Service	29
115	7.4.4	Car Wash Example.....	30
116	7.5	The ServiceContract Class	30
117	7.5.1	The interactionAspect and legalAspect Datatype Properties	31
118	7.6	The hasContract and isContractFor Properties.....	32
119	7.7	The involvesParty and isPartyTo Properties	33
120	7.8	The Effect Class	34
121	7.9	The specifies and isSpecifiedBy Properties	34
122	7.10	ServiceContract – Examples.....	36
123	7.10.1	Service-Level Agreements	36
124	7.10.2	Service Sourcing.....	36
125	7.10.3	Car Wash Example.....	37
126	7.11	The ServiceInterface Class	37
127	7.11.1	The Constraints Datatype Property	38
128	7.12	The hasInterface and isInterfaceOf Properties	39
129	7.13	The InformationType Class.....	39
130	7.14	The hasInput and isInputAt Properties.....	40
131	7.15	The hasOutput and isOutputAt Properties.....	41
132	7.16	Examples	41
133	7.16.1	Interaction Sequencing	41
134	7.16.2	Car Wash Example.....	42
135	8	Composition and its Subclasses	42
136	8.1	Introduction	42
137	8.2	The Composition Class	42
138	8.2.1	The compositionPattern Datatype Property.....	44
139	8.3	The orchestrates and orchestratedBy Properties	46
140	8.4	The ServiceComposition Class	48
141	8.5	The Process Class	49
142	8.6	Service Composition and Process Examples.....	50
143	8.6.1	Simple Service Composition Example	50
144	8.6.2	Process Example	51
145	8.6.3	Process and Service Composition Example.....	51
146	8.6.4	Car Wash Example.....	51
147	9	Policy	51
148	9.1	Introduction	51
149	9.2	The Policy Class	52
150	9.2.1	The appliesTo and isSubjectTo Properties	54
151	9.3	The setsPolicy and isSetBy Properties	54
152	9.4	Examples	55
153	9.4.1	Car Wash Example.....	55
154	10	Event	55
155	10.1	Introduction	55
156	10.2	The Event Class	55

157	10.3	The generates and generatedBy Properties	56
158	10.4	The respondsTo and respondedToBy Properties	57
159	11	Complete Car Wash Example	57
160	11.1	The Organizational Aspect.....	57
161	11.2	The Washing Services.....	59
162	11.3	Interfaces to the Washing Services	60
163	11.4	The Washing Processes.....	61
164	11.5	The Washing Policies	62
165	12	Internet Purchase Example.....	63
166	Annex A	The OWL Definition of the SOA Ontology	65
167	Annex B (Informative)	Class Relationship Matrix.....	78
168	Annex C (Informative)	Issues List	81
169	Annex D (Informative)	Bibliography.....	82
170			

171 **Foreword**

172 ISO (the International Organization for Standardization) is a worldwide federation of national standards
173 bodies (ISO member bodies). The work of preparing International Standards is normally carried out through
174 ISO technical committees. Each member body interested in a subject for which a technical committee has
175 been established has the right to be represented on that committee. International organizations,
176 governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely
177 with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

178 International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

179 The main task of technical committees is to prepare International Standards. Draft International Standards
180 adopted by the technical committees are circulated to the member bodies for voting. Publication as an
181 International Standard requires approval by at least 75 % of the member bodies casting a vote.

182 **Attention is drawn to the possibility that some of the elements of this document may be the subject of patent**
183 **rights. ISO shall not be held responsible for identifying any or all such patent rights.**

184 ISO 18384-n was prepared by Technical Committee ISO/JTC 1, Subcommittee SC 38, SC DAPS Work
185 Group 2, SOA Working Group.

186 ISO 18384 consists of three parts, under the general title: Reference Architecture for Service Oriented
187 Architecture Part 1 is: SOA Terminology and Concepts, Part 2 is Reference Architecture for SOA: this
188 document is Part 3, SOA Ontology.

189 Introduction

190 The purpose of this International Standard is to contribute to developing and fostering common
191 understanding of Service-Oriented Architecture (SOA) in order to improve alignment between the business
192 and information technology communities, and facilitate SOA adoption.

193 It does this in two specific ways:

194 It defines the concepts, terminology, and semantics of SOA in both business and technical terms, in order to:

- 195 • Create a foundation for further work in domain-specific areas
- 196 • Enable communications between business and technical people
- 197 • Enhance the understanding of SOA concepts in the business and technical communities
- 198 • Provide a means to state problems and opportunities clearly and unambiguously to promote mutual
199 understanding
- 200 • It potentially contributes to model-driven SOA implementation.

201 The ontology is designed for use by:

- 202 • Business people, to give them a deeper understanding of SOA concepts and how they are used in the
203 enterprise and its environment
- 204 • Architects, as metadata for architectural artifacts
- 205 • Architecture methodologists, as a component of SOA meta-models
- 206 • System and software designers for guidance in terminology and structure
207

208 This report defines the following clauses:

209 Clause 3 – terminology – defines terms used when discussing or designing service oriented solutions. Terms
210 defined here are used in some unique fashion for SOA. It does not define terms that are used in general English
211 manner.

212 Clause 4 – Overview clause provides an introduction to the whole standard.

ISO/IEC WD 1 18384 Part 3 SOA Ontology

213 Clauses 5 through 10 provide the formal definitions (OWL and natural language) of the terms and
214 concepts included in the ontology.
215

216 Clause 4 – System and Element

217 Clause 5 – Human Actor and Task

218 Clause 6 – Service, Service Contract, and Service Interface

219 Clause 7 – Composition and its Subclasses

220 Clause 8 – Policy

221 Clause 9 – Event

222 Clause 11 contains the complete car wash example that is used as a common example throughout.

223 Clause 11 contains an additional elaborate example utilizing most of the classes in the ontology.

224 Appendix **Error! Reference source not found.** contains the formal OWL definitions of the ontology,
225 collected together.

226 Appendix **Error! Reference source not found.** describes the relation of this ontology to other work.

227 Appendix **Error! Reference source not found.** contains a relationship matrix that details the class
228 relationships implied by the OWL definitions of the ontology.
229

230

231

232 **Distributed Application Platforms and Services (DAPS)**

233

SOA Reference Architecture

234

Service Oriented Architecture Ontology

235 **1 Scope**

236 This Standard defines a formal ontology for Service-Oriented Architecture (SOA). SOA is an architectural style that
237 supports service-orientation. This is the official definition of SOA as defined by The SOA Reference Architecture Part
238 1. For full details, [see SOA Reference Architecture Part 1]

239 **2 Normative references**

240 The following referenced documents are indispensable for the application of this document. For dated references, only
241 the edition cited applies. For undated references, the latest edition of the referenced document (including any
242 amendments) applies.

243 **Editors note: Normative references need to be identified**

244 **3 Terms, Definitions, Notations, and Conventions**

245 For the purposes of this document, the following terms and definitions apply:

246 Those terms and definitions defined by SOA reference Architecture Part 1.

247

248

249 **3.1 Definitions**

250 3.1.1 Opaque

251 any possible internal structure of something is invisible to an external observer

252

253

254

255 **3.2 Acronyms**

256 BPMN – Business Process Management Notation

257 IT – Information Technology

258 EA – Enterprise Architecture

259 RA – Reference Architecture

260 SLA – Service Level Agreement

261 SOA - Service Oriented Architecture

262 **Editors note: Acronyms need to be identified and added**

263 **3.3 Notations**

264

265 **3.4 Conventions**

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

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

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

269 An unlabeled arrow in the illustrative UML diagrams means subclass.

270

271 The examples in this document are strictly informative and are for illustrative purposes.

272 **4 SOA Ontology Overview**

273 This Technical Standard defines a formal ontology for Service-Oriented Architecture (SOA). SOA is an
274 architectural style that supports service-orientation. This is the official definition of SOA as defined by
275 The SOA Reference Architecture Part 1. For full details, [see SOA Reference Architecture Part 1]

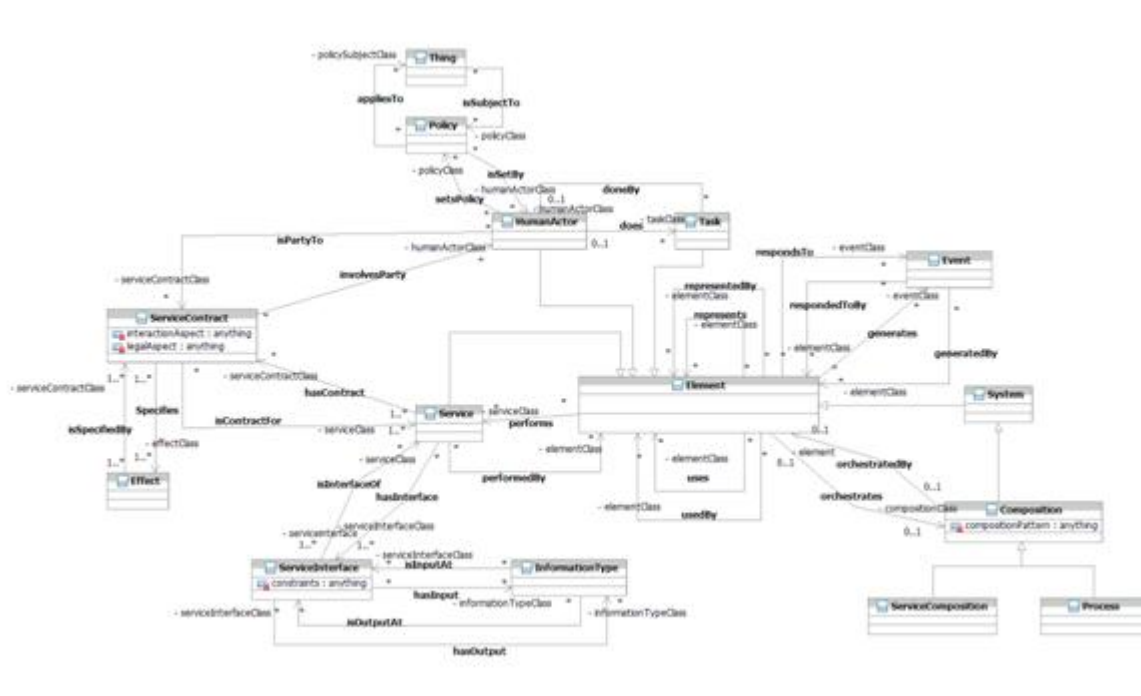
276 The ontology is represented in the Web Ontology Language (OWL) defined by the World-Wide Web
277 Consortium. OWL has three increasingly expressive sub-languages: OWL-Lite, OWL-DL, and OWL-
278 Full. (See www.w3.org/2004/OWL for a definition of these three dialects of OWL.) This ontology uses
279 OWL-DL, the sub-language that provides the greatest expressiveness possible while retaining
280 computational completeness and decidability.

281 The ontology contains classes and properties corresponding to the core concepts of SOA. The formal
282 OWL definitions are supplemented by natural language descriptions of the concepts, with graphic

283 illustrations of the relations between them, and with examples of their use. For purposes of exposition,
 284 the ontology also includes UML diagrams that graphically illustrate its classes and properties of the
 285 ontology. The natural language and OWL definitions contained in this specification constitute the
 286 authoritative definition of the ontology; the diagrams are for explanatory purposes only. Some of the
 287 natural language terms used to describe the concepts are not formally represented in the ontology; those
 288 terms are meant in their natural language sense.

289 This Technical Standard uses examples to illustrate the ontology. One of these, the car-wash example, is
 290 used consistently throughout to illustrate the main concepts. (See Clause 11 for the complete example.)
 291 Other examples are used ad hoc in individual clauses to illustrate particular points.

292 A graphically compressed visualization of the entire ontology is shown below (in Figure 1).



293

294

Figure 1: SOA Ontology – Graphical Overview

295 The concepts illustrated in this figure (Figure 1) are described in the body of this Technical
 296 Standard.

297 4.1.1 Applications

298 The SOA ontology specification was developed in order to aid understanding, and potentially be a basis
 299 for model-driven implementation.

300 To aid understanding, this specification can simply be read. To be a basis for model-driven
 301 implementation, it should be applied to particular usage domains and application to example usage
 302 domains will aid understanding.

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

307 The ontology defines the relations between terms, but does not prescribe exactly how they should be
308 applied. (Explanations of what ontologies are and why they are needed can be found in, for example,
309 Beyond Concepts: Ontology as Reality Representation and What is an Ontology?) The examples
310 provided in this Technical Standard are describing one way in which the ontology could be applied in
311 practical situations. Different applications of the ontology to the same situations would nevertheless be
312 possible. The precise instantiation of the ontology in particular practical situations is a matter for users of
313 the ontology; as long as the concepts and constraints defined by the ontology are correctly applied, the
314 instantiation is valid.

315 4.1.2 Conformance

316 There are two kinds of applications that can potentially conform to this ontology. One is other OWL-
317 based ontologies (typically extensions of the SOA ontology); the other is a non-OWL application such as
318 a meta-model or a piece of software.

319 A conforming OWL application (derived OWL-based ontology):

- 320 • Must conform to the OWL standard
- 321 • Must include (in the OWL sense) the whole of the ontology contained in Appendix AA of this
322 Technical Standard
- 323 • Can add other OWL constructs, including class and property definitions
- 324 • Can import other ontologies in addition to the SOA ontology

325 A conforming non-OWL application:

- 326 • Must include a defined and consistent transform to a non-trivial subset of the ontology contained in
327 Appendix AA of this Technical Standard
- 328 • Can add other constructs, including class and property definitions
- 329 • Can leverage other ontologies in addition to the SOA ontology

330 5 System and Element

331 5.1 Introduction

332 *System* and *element* are two of the core concepts of this ontology. Both are concepts that are often used
333 by practitioners, including the notion that systems have elements and that systems can be hierarchically

334 combined (systems of systems). What differs from domain to domain is the specific nature of systems
 335 and elements; for instance, an electrical system has very different kinds of elements than an SOA system.

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

339 This Clause describes the following classes of the ontology:

340 **ElementSystem**

341 In addition, it defines the following properties:

342 **uses** and **usedBy**

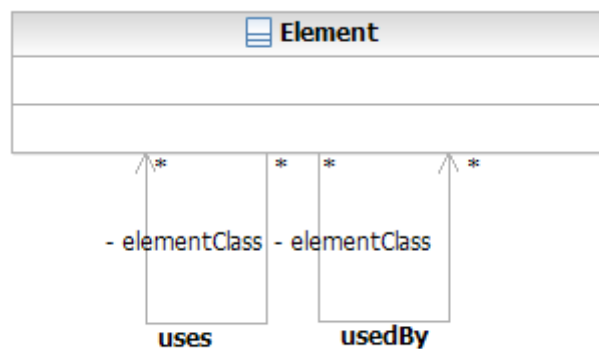
343 **represents** and **representedBy**

344 **5.2 The Element Class**

345 <owl:Class rdf:about="#Element">

346 </owl:Class>

347 An *element* is an opaque entity that is indivisible at a given level of abstraction. The element has a
 348 clearly defined boundary. The concept of element is captured by the **Element** OWL class, which is
 349 illustrated [below \(in Figure 1\)](#).



350

351 **Figure 1: The Element Class**

352 In the context of the SOA ontology we consider in detail only functional elements that belong to the
 353 SOA domain. There are other kinds of Elements than members of the four named subclasses (System,

354 HumanActor, Task, and Service) described later in this ontology. Examples of such other kinds of
355 Elements are things like software components or technology components (such as Enterprise Service
356 Bus (ESB) implementations, etc.).

357 5.3 The uses and usedBy Properties

```
358 <owl:ObjectProperty rdf:about="#uses">
```

```
359   <rdfs:domain rdf:resource="#Element"/>
```

```
360   <rdfs:range rdf:resource="#Element"/>
```

```
361 </owl:ObjectProperty>
```

```
362
```

```
363 <owl:ObjectProperty rdf:ID="usedBy">
```

```
364   <owl:inverseOf>
```

```
365     <owl:ObjectProperty rdf:ID="uses"/>
```

```
366   </owl:inverseOf>
```

```
367 </owl:ObjectProperty>
```

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

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

379 For the purposes of this ontology we have chosen not to attempt to enumerate and formally define the
380 multitude of different possible semantics of a *uses* relationship. We leave the semantic interpretations to
381 a particular sub-domain, application or even design approach.

382 5.4 Element – Organizational Example

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

386 Inside the boundary of the organizational unit we want to express the fact that an organizational unit uses
 387 the people that are members of it. Note that the same person can in fact be a member of (be used by)
 388 multiple organizational units.

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

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

395 5.5 The System Class

```

396 <owl:Class rdf:ID="System">
397   <owl:disjointWith>
398     <owl:Class rdf:ID="Task"/>
399   </owl:disjointWith>
400   <owl:disjointWith>
401     <owl:Class rdf:ID="Service"/>
402   </owl:disjointWith>
403   <rdfs:subClassOf>
404     <owl:Class rdf:about="#Element"/>
405   </rdfs:subClassOf>
406 </owl:Class>

```

407 A *system* is an organized collection of other things. Specifically things in a system collection are
 408 instances of **Element**, each such instance being used by the system. The concept of *system* is captured by
 409 the **System** OWL class, which is illustrated [below \(in Figure 2\)](#).



410

411

Figure 2: The System Class

412 This definition of System is heavily influenced by IEEE Std 1471-2000, adopted by ISO/IEC JTC1/SC7
 413 as ISO/IEC 42010:2007: Systems and Software Engineering – Recommended Practice for Architectural
 414 Description of Software-intensive Systems.

415 In the context of the SOA ontology we consider in detail only functional systems that belong to the SOA
 416 domain. Note that a fully described instance of **System** should have by its nature (as a collection) a *uses*
 417 relationship to at least one instance of **Element**.

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

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

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

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

438 5.6 System – Examples

439 5.6.1 Organizational Example

440 Continuing the organizational example from above, we can now express that an organizational unit as an
441 instance of **System** has the people in it as members (and instances of element).

442 5.6.2 Service Composition Example

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

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

449 5.6.3 Car Wash Example

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

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

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

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

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

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

458 5.7 The represents and representedBy Properties

459 <owl:ObjectProperty rdf:about="#represents">

460 <rdfs:domain rdf:resource="#Element"/>

461 <rdfs:range rdf:resource="#Element"/>

462 </owl:ObjectProperty>

463

464 <owl:ObjectProperty rdf:ID="representedBy">

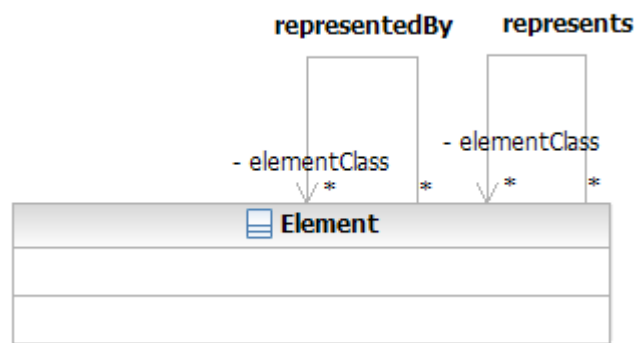
465 <owl:inverseOf>

466 <owl:ObjectProperty rdf:ID="represents"/>

467 </owl:inverseOf>

468 </owl:ObjectProperty>

469 The environment described by an SOA is intrinsically hierarchically composite (see also Clause 6.2 for a
 470 definition of the *Composition* class); in other words, the elements of SOA systems can be repeatedly
 471 composed to ever higher levels of abstraction. One aspect of this has already been addressed by the **uses**
 472 and **usedBy** properties in that we can use these to express the notion of systems of systems. This is still a
 473 very concrete relationship though, and does not express the concept of architectural abstraction. We find
 474 the need for architectural abstraction in various places such as a role representing the people playing that
 475 role, an organizational unit representing the people within it (subtly different from that same
 476 organizational unit using the people within it, as the **represents** relationship indicates the organizational
 477 unit as a substitute interaction point), an architectural building block representing an underlying
 478 construct (for instance, important to enterprise architects wanting to explicitly distinguish between
 479 constructs and building blocks), and an Enterprise Service Bus (ESB) representing the services that are
 480 accessible through it (for instance, relevant when explicitly modeling operational interaction and
 481 dependencies). The concept of such an explicitly changing view point, or level of abstraction, is captured
 482 by the *represents* and *representedBy* properties illustrated [below \(in Figure 3\)](#).



483

484 **Figure 3: The represents and representedBy Properties**

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

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

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

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

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

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

506 5.8 Examples

507 5.8.1 Organizational Example

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

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

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

520 In this way the abstraction aspect of the **represents** property induces an important difference in the
521 semantics of the collection defining the new organization. Any instantiation of the ontology can and

522 should use the **represents** and **representedBy** properties to crisply define the implied semantics and
523 lines of visibility/change.

524 **5.8.2 Car Wash Example**

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

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

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

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

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

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

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

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

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

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

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

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

540

541 **6 HumanActor and Task**

542 **6.1 Introduction**

543 People, organizations, and the things they do are important aspects of SOA systems. *HumanActor* and
544 *Task* capture this as another set of core concepts of the ontology. Both are concepts that are generic and
545 have relevance outside the domain of SOA. For the purposes of this SOA ontology we have chosen to
546 give them specific scope in that tasks are intrinsically atomic (corresponding to, for instance, the

547 Business Process Modeling Notation (BPMN) 2.0 definition of Task) and human actors are restricted to
 548 people and organizations.

549 This Clause describes the following classes of the ontology:

550 **HumanActor**

551 **Task**

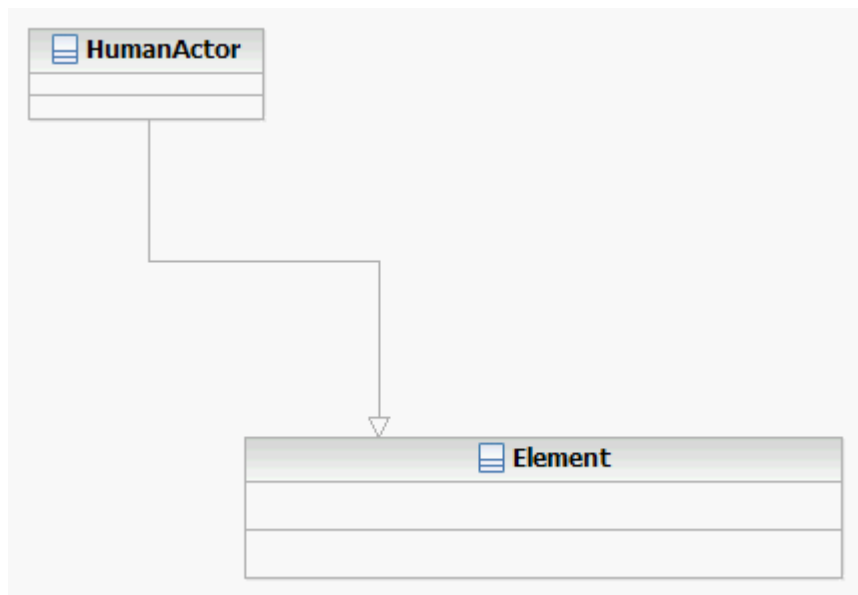
552 In addition, it defines the following properties:

553 **does** and **doneBy**

554 6.2 The HumanActor Class

```
555 <owl:Class rdf:about="#HumanActor">
556   <rdfs:subClassOf>
557     <owl:Class rdf:ID="Element"/>
558   </rdfs:subClassOf>
559   <owl:disjointWith>
560     <owl:Class rdf:ID="Task"/>
561   </owl:disjointWith>
562   <owl:disjointWith>
563     <owl:Class rdf:ID="Service"/>
564   </owl:disjointWith>
565 </owl:Class>
```

566 A *human actor* is a person or an organization. The concept of *human actor* is captured by the
 567 **HumanActor** OWL class, which is illustrated [below](#) (in Figure 4).



568

569

Figure 4: The HumanActor Class

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

574 6.3 HumanActor – Examples

575 6.3.1 The uses and usedBy Properties Applied to HumanActor

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

579 6.3.2 The represents and representedBy Properties Applied to HumanActor

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

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

588 Note that we have chosen not to define a “role class”, as we believe that using **Element** with the
589 **represents** property is a more general approach which does not limit the ability to also define role-based
590 systems. For all practical purposes there is simply a “role subclass” of **Element**, a subclass that we have
591 chosen not to define explicitly.

592 6.3.3 Organizational Example

593 Continuing the organizational example from above, we can now express that P1 (*John*), P2 (*Jack*), P3
594 (*Joe*), and P4 (*Mary*) as instances of **Element** are in fact (people) instances of **HumanActor**. We can
595 also express (if we so choose) that all of O1 (*CarWashBusiness*), O3 (*CarWash*), and O4
596 (*Administration*) are (organization) human actors from an action perspective at the same time that they
597 are systems from a collection/composition perspective.

598 6.3.4 Car Wash Example

599 See Clause 11.1 for the complete organizational aspect of the car wash example.

600 **6.4 The Task Class**

```

601 <owl:Class rdf:about="#Task">
602   <owl:disjointWith>
603     <owl:Class rdf:ID="System"/>
604   </owl:disjointWith>
605   <owl:disjointWith>
606     <owl:Class rdf:ID="HumanActor"/>
607   </owl:disjointWith>
608   <owl:disjointWith>
609     <owl:Class rdf:ID="Service"/>
610   </owl:disjointWith>
611   <rdfs:subClassOf>
612     <owl:Class rdf:ID="Element"/>
613   </rdfs:subClassOf>
614 </owl:Class>

```

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

617 The Business Process Modeling Notation (BPMN) 2.0 defines task as follows: “A Task is an atomic
618 Activity within a Process flow. A Task is used when the work in the Process cannot be broken down to a
619 finer level of detail. Generally, an end-user and/or applications are used to perform the Task when it is
620 executed.” For the purposes of the ontology we have added precision by formally separating the notion
621 of doing from the notion of performing. Tasks are (optionally) done by human actors, furthermore (as
622 instances of Element) tasks can use services that are performed by technology components (see details in
623 Clause 7.3; see also the example in Clause 12).

624 The concept of *task* is captured by the **Task** OWL class, which is illustrated [below](#) (in Figure 5).

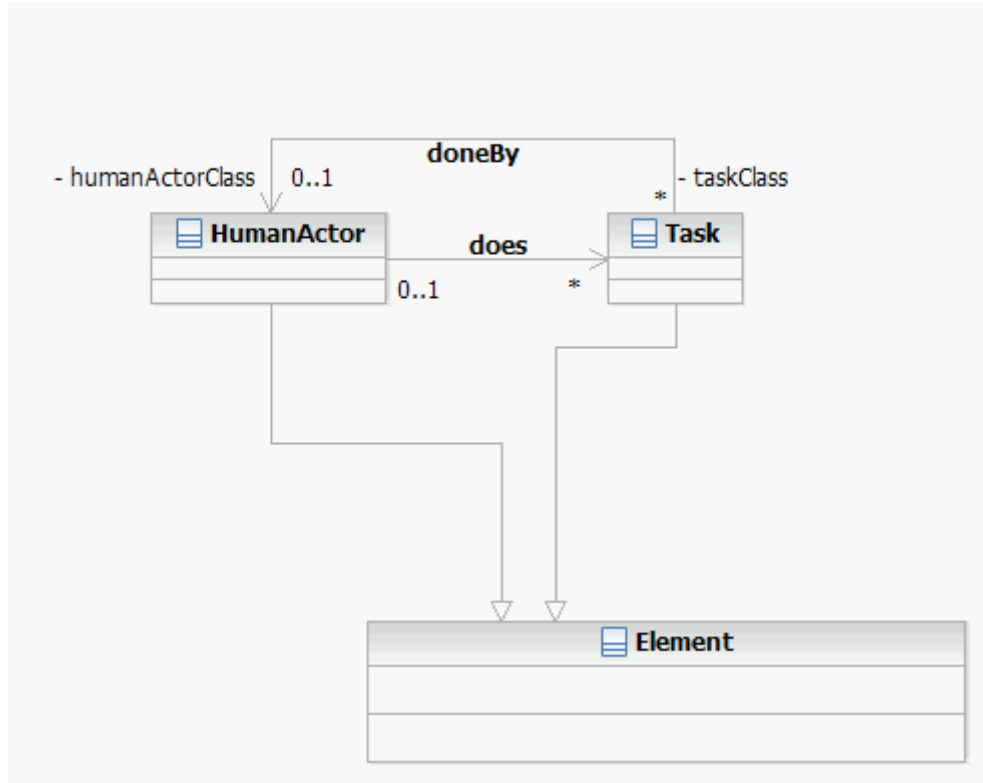


Figure 5: The Task Class

625

626

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

629 **6.5 The does and doneBy Properties**

```

630 <owl:ObjectProperty rdf:about="#doneBy">
631   <rdfs:domain rdf:resource="#Task"/>
632   <rdfs:range rdf:resource="#HumanActor"/>
633 </owl:ObjectProperty>
634
635 <owl:ObjectProperty rdf:ID="does">
636   <owl:inverseOf>
637     <owl:ObjectProperty rdf:about="#doneBy"/>
638   </owl:inverseOf>
639 </owl:ObjectProperty>
640
641 <owl:Class rdf:ID="Task">
642   <rdfs:subClassOf>
643     <owl:Restriction>
644       <owl:onProperty>
645         <owl:ObjectProperty rdf:ID="doneBy"/>
646       </owl:onProperty>
647       <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
    
```

```

648         >0</owl:minCardinality>
649     </owl:Restriction>
650 </rdfs:subClassOf>
651 <rdfs:subClassOf>
652     <owl:Restriction>
653         <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
654         >1</owl:maxCardinality>
655         <owl:onProperty>
656             <owl:ObjectProperty rdf:about="#doneBy"/>
657         </owl:onProperty>
658     </owl:Restriction>
659 </rdfs:subClassOf>
660 </owl:Class>

```

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

668 6.6 Task – Examples

669 6.6.1 The uses and usedBy Properties Applied to Task

670 In one direction, the most common case of a task using another element is where an automated task (in
671 an orchestrated process; see Clause **Error! Reference source not found.** for the definition of *process*
672 and *orchestration*) uses a service as its realization. In the other direction, a task can, for instance, be used
673 by a system (as an element within that system, such as a task in a process).

674 6.6.2 The represents and representedBy Properties Applied to Task

675 As mentioned in the introduction to this clause, tasks are intrinsically part of SOA systems. Yet in many
676 cases as an element of an SOA system we talk about not the actual thing being done, rather an abstract
677 representation of it that is used as an element in systems, processes, etc. In other words, we talk about
678 elements representing tasks.

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

682 6.6.3 Organizational Example

683 Continuing the organizational example from above, we can now express which tasks that are done by
684 human actors (people) P1, P2, P3, and P4, and how those tasks can be elements in bigger systems that

685 describe things such as organizational processes. Clause **Error! Reference source not found.** will deal
686 formally with the concept of *composition*, including properly defining the concept of a *process* as one
687 particular kind of *composition*.

688 **6.6.4 Car Wash Example**

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

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

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

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

694 **7 Service, ServiceContract, and ServiceInterface**

695 **7.1 Introduction**

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

699 “*A service is a logical representation of a repeatable activity that has a specified outcome. It is self-*
700 *contained and is a ‘black box’ to its consumers.*”

701 This corresponds to the existing official Open Group definition of the term; refer to [the Open Group](#)
702 [Definition of SOA](#).

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

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

- 711 • “*A mechanism to enable access to one or more capabilities, where the access is provided using a*
712 *prescribed interface and is exercised consistent with constraints and policies as specified by the*
713 *service description.*” (Source: OASIS SOA Reference Model)

- 714 • *“A capability offered by one entity or entities to others using well-defined ‘terms and conditions’*
715 *and interfaces.”* (Source: OMG SoaML Specification)

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

719 This clause describes the following classes of the ontology:

- 720 • **Service**
- 721 • **ServiceContract**
- 722 • **ServiceInterface**
- 723 • **InformationType**

724 In addition, it defines the following properties:

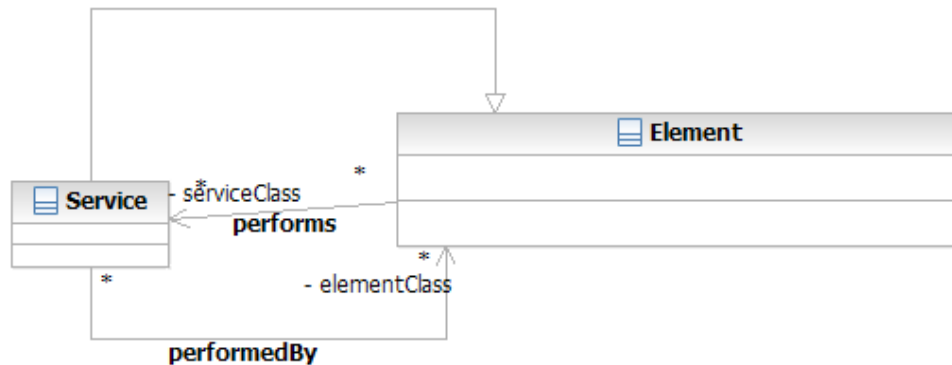
- 725 • **performs** and **performedBy**
- 726 • **hasContract** and **isContractFor**
- 727 • **involvesParty** and **isPartyTo**
- 728 • **specifies** and **isSpecifiedBy**
- 729 • **hasInterface** and **isInterfaceOf**
- 730 • **hasInput** and **isInputAt**
- 731 • **hasOutput** and **isOutputAt**

732 **7.2 The Service Class**

```
733       <owl:Class rdf:about="#Service">
734        <owl:disjointWith>
735         <owl:Class rdf:ID="System"/>
736       </owl:disjointWith>
737       <owl:disjointWith>
738         <owl:Class rdf:ID="Task"/>
739       </owl:disjointWith>
740       <owl:disjointWith>
741         <owl:Class rdf:ID="HumanActor"/>
742       </owl:disjointWith>
743       <rdfs:subClassOf>
744         <owl:Class rdf:about="#Element"/>
```

745 </rdfs:subClassOf>
 746 </owl:Class>

747 A *service* is a logical representation of a repeatable activity that has a specified outcome. It is self-
 748 contained and is a ‘black box’ to its consumers. The concept of *service* is captured by the **Service** OWL
 749 class, which is illustrated [below \(in Figure 6\)](#).



750

751

Figure 6: The Service Class

752 In the context of the SOA ontology we consider only SOA-based services. Other domains, such as
 753 Integrated Service Management, can have services that are not SOA-based hence are outside the
 754 intended scope of the SOA ontology.

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

758 7.3 The performs and performedBy Properties

```

759 <owl:ObjectProperty rdf:ID="performs">
760   <rdfs:domain rdf:resource="#Element"/>
761   <rdfs:range rdf:resource="#Service"/>
762 </owl:ObjectProperty>
763
764 <owl:ObjectProperty rdf:ID="performedBy">
765   <owl:inverseOf>
766     <owl:ObjectProperty rdf:ID="performs"/>
767   </owl:inverseOf>
768 </owl:ObjectProperty>
    
```

769 As a service itself is only a logical representation, any service is *performed* by something. The
 770 something that *performs* a service must be opaque to anyone interacting with it, an opaqueness which is
 771 the exact nature of the **Element** class. This concept is captured by the *performs* and *performedBy*

772 properties as illustrated in [The Service Class](#) (Figure 6). This also captures the fact that services can be
 773 performed by elements of other types than systems. This includes elements such as software
 774 components, human actors, and tasks.

775 Note that the same instance of **Service** can be performed by many different instances of **Element**. As
 776 long as the service performed is the same, an external observer cannot tell the difference (for contractual
 777 obligations, SLAs, etc. see the definition of the *ServiceContract* class in Clause 7.5.). Conversely, any
 778 instance of **Element** may perform more than one service or none at all.

779 While a service can be performed by other elements, the service itself (as a purely logical representation)
 780 does not perform other services. See the [Simple Service Composition Example](#) (Clause 8.6.1) for an
 781 example of how to represent service compositions formally in the ontology.

782 7.3.1 Service Consumers and Service Providers

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

- 785 • It does not distinguish between the contractual obligation aspect of consume/provide and the
 786 interaction aspect of consume/provide. A contractual obligation does not necessarily translate to an
 787 interaction dependency, if for no other reason than because the realization of the contractual
 788 obligation may have been sourced to a third party.
- 789 • Consuming or providing a service is a statement that only makes sense in context – either a
 790 contractual context or an interaction context. These terms are consequently not well suited for making
 791 statements about elements and services in isolation.

792 The above are the reasons why the ontology has chosen not to adopt consume and provide as core concepts,
 793 rather instead allows consume or provide terms used with contractual obligations and/or interaction rules
 794 described by service contracts; see the definition of the *ServiceContract* class in Clause 7.5. In its simplest
 795 form, outside the context of a formal service contract, the interaction aspect of consuming and providing
 796 services may even be expressed simply by saying that some element uses (consumes) a service or that some
 797 element performs (provides) a service; see also the examples below.

798 7.4 Service – Examples

799 7.4.1 The uses and usedBy Properties Applied to Service

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

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

- 807 • Picking the service to interact with (this statement is agnostic as to whether this is done
808 dynamically at runtime or statically at design and/or construct time)
- 809 • Picking an element that performs that service (in a typical SOA environment, this is most often
810 done “inside” an Enterprise Service Bus (ESB))
- 811 • Interacting with the chosen element (that performs the chosen) service (often also facilitated by an
812 ESB)

813 7.4.2 The represents and representedBy Properties Applied to Service

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

822 While a service being represented by something else is quite natural, it is harder to imagine what the
823 service itself might represent. To some degree we have already captured the fact that a service represents
824 any embodiment of it, only we have chosen to use the *performs* and *performedBy* properties to described
825 this rather than the generic *represents* and *representedBy* properties. As a consequence, we do not expect
826 practical applications of the ontology to have services represent anything.

827 7.4.3 Exemplifying the Difference between Doing a Task and Performing a Service

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

- 831 • *John* is an instance of **HumanActor**.
- 832 • *WashWindows* is an instance of **Task** and is done by *John*.
- 833 • *SoapWater* is an instance of **Service**.
- 834 • *WaterTap* is an instance of **Element**.
- 835 • *WaterTap* performs *SoapWater*.

836 • *John* uses *SoapWater* (to do *WashWindows*).

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

838 7.4.4 **Car Wash Example**

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

841 • *GoldWash* is an instance of **Service**.

842 • *BasicWash* is an instance of **Service**.

843 • *CarWash* performs both *BasicWash* and *GoldWash*.

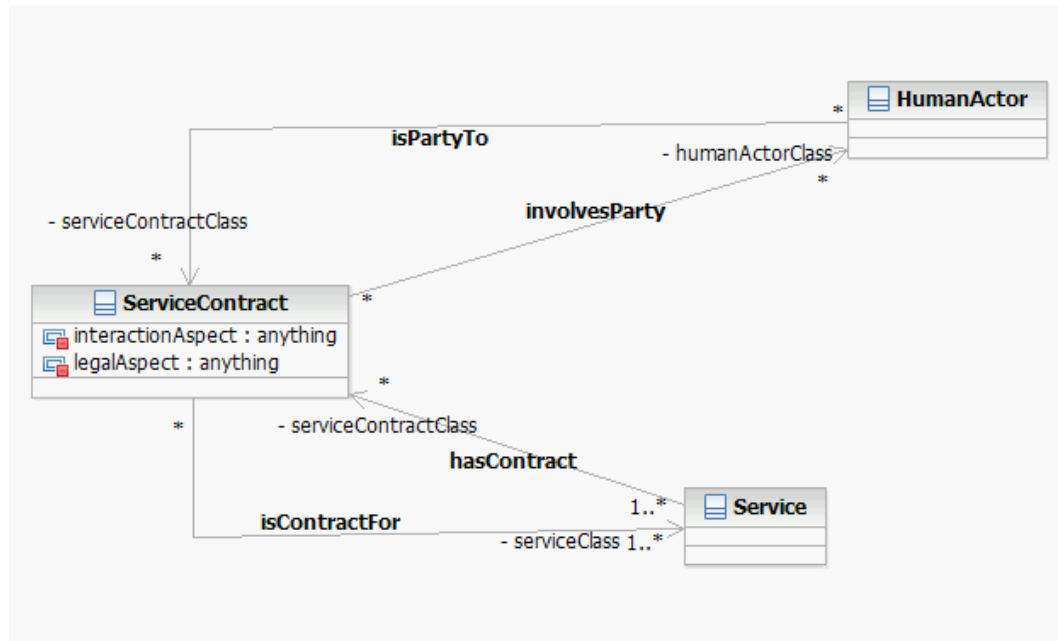
844 • *WashManager* represents both *BasicWash* and *GoldWash* (i.e., is the interaction point where
845 customers can order services as well as pay for them).

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

849 7.5 **The ServiceContract Class**

```
850       <owl:Class rdf:about="#ServiceContract">
851        <owl:disjointWith>
852         <owl:Class rdf:ID="HumanActor"/>
853       </owl:disjointWith>
854       <owl:disjointWith>
855         <owl:Class rdf:ID="Task"/>
856       </owl:disjointWith>
857       </owl:Class>
```

858 In many cases, specific agreements are needed in order to define how to use a service. This can either be
859 because of a desire to regulate such use or can simply be because the service will not function properly
860 unless interaction with it is done in a certain sequence. A *service contract* defines the terms, conditions,
861 and interaction rules that interacting participants must agree to (directly or indirectly). A *service contract*
862 is binding on all participants in the interaction, including the service itself and the element that provides
863 it for the particular interaction in question. The concept of *service contract* is captured by the
864 **ServiceContract** OWL class, which is illustrated [below \(in Figure 7\)](#).



865

866

Figure 7: The ServiceContract Class

867 **7.5.1 The interactionAspect and legalAspect Datatype Properties**

```

868 <owl:DatatypeProperty rdf:about="#interactionAspect">
869   <rdfs:domain rdf:resource="#ServiceContract"/>
870 </owl:DatatypeProperty>
871
872 <owl:DatatypeProperty rdf:about="#legalAspect">
873   <rdfs:domain rdf:resource="#ServiceContract"/>
874 </owl:DatatypeProperty>
875
876 <owl:Class rdf:about="#ServiceContract">
877   <rdfs:subClassOf>
878     <owl:Restriction>
879       <owl:onProperty>
880         <owl:DatatypeProperty rdf:ID="legalAspect"/>
881       </owl:onProperty>
882       <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
883         >1</owl:minCardinality>
884     </owl:Restriction>
885   </rdfs:subClassOf>
886   <rdfs:subClassOf>
887     <owl:Restriction>
888       <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
889         >1</owl:maxCardinality>
890     <owl:onProperty>
891       <owl:DatatypeProperty rdf:ID="legalAspect"/>
892     </owl:onProperty>
893   </owl:Restriction>

```

```

894     </rdfs:subClassOf>
895 <rdfs:subClassOf>
896   <owl:Restriction>
897     <owl:onProperty>
898       <owl:DatatypeProperty rdf:ID="interactionAspect"/>
899     </owl:onProperty>
900     <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
901       >1</owl:maxCardinality>
902   </owl:Restriction>
903 </rdfs:subClassOf>
904 <rdfs:subClassOf>
905   <owl:Restriction>
906     <owl:onProperty>
907       <owl:DatatypeProperty rdf:about="#interactionAspect"/>
908     </owl:onProperty>
909     <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
910       >1</owl:minCardinality>
911   </owl:Restriction>
912 </rdfs:subClassOf>
913 </owl:Class>

```

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

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

921 7.6 The *hasContract* and *isContractFor* Properties

```

922 <owl:ObjectProperty rdf:about="#isContractFor">
923   <rdfs:domain rdf:resource="#ServiceContract"/>
924   <rdfs:range rdf:resource="#Service"/>
925 </owl:ObjectProperty>
926
927 <owl:ObjectProperty rdf:ID="hasContract">
928   <owl:inverseOf>
929     <owl:ObjectProperty rdf:about="#isContractFor"/>
930   </owl:inverseOf>
931 </owl:ObjectProperty>
932
933 <owl:Class rdf:about="#ServiceContract">
934   <rdfs:subClassOf>
935     <owl:Restriction>
936       <owl:onProperty>
937         <owl:ObjectProperty rdf:ID="isContractFor"/>
938       </owl:onProperty>
939       <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
940         >1</owl:minCardinality>

```

```

941         </owl:Restriction>
942     </rdfs:subClassOf>
943 </owl:Class>

```

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

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

953 7.7 The **involvesParty** and **isPartyTo** Properties

```

954 <owl:ObjectProperty rdf:about="#isPartyTo">
955     <rdfs:domain rdf:resource="#HumanActor"/>
956     <rdfs:range rdf:resource="#ServiceContract"/>
957 </owl:ObjectProperty>
958
959 <owl:ObjectProperty rdf:ID="involvesParty">
960     <owl:inverseOf>
961         <owl:ObjectProperty rdf:ID="isPartyTo"/>
962     </owl:inverseOf>
963 </owl:ObjectProperty>

```

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

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

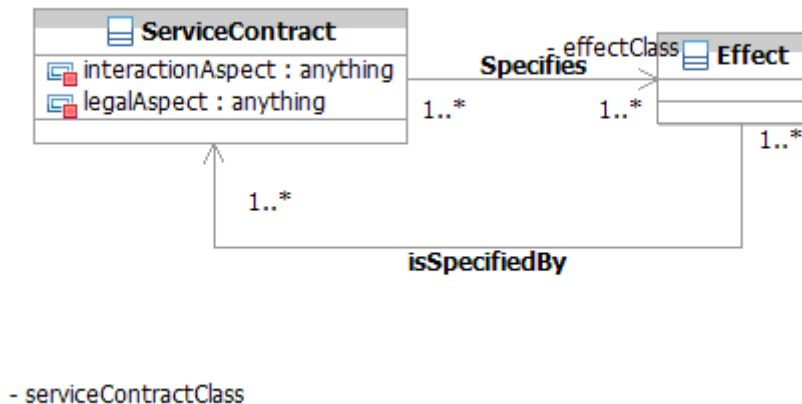
974 There is a many-to-many relationship between service contracts and human actors. A given human actor
 975 may be party to none, one, or many service contracts. Similarly, a given service contract may involve
 976 none, one, or multiple human actors (none in the case where that particular service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

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

984 7.8 The Effect Class

```
985 <owl:Class rdf:about="#Effect">
986   <owl:disjointWith>
987     <owl:Class rdf:ID="ServiceInterface"/>
988   </owl:disjointWith>
989 </owl:Class>
```

990 Interacting with something performing a service has *effects*. These comprise the outcome of that
 991 interaction, and are how a service (through the element that performs it) delivers value to its consumers.
 992 The concept of *effect* is captured by the **Effect** OWL class, which is illustrated [below \(in Figure 8\)](#).



993

994

Figure 8: The Effect Class

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

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

1000 7.9 The specifies and isSpecifiedBy Properties

```
1001 <owl:ObjectProperty rdf:about="#specifies">
1002   <rdfs:domain rdf:resource="#ServiceContract"/>
1003   <rdfs:range rdf:resource="#Effect"/>
```

34

```

1004 </owl:ObjectProperty>
1005
1006 <owl:ObjectProperty rdf:about="#isSpecifiedBy">
1007   <owl:inverseOf>
1008     <owl:ObjectProperty rdf:about="#specifies"/>
1009   </owl:inverseOf>
1010 </owl:ObjectProperty>
1011
1012 <owl:Class rdf:ID="Effect">
1013   <rdfs:subClassOf>
1014     <owl:Restriction>
1015       <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
1016         >1</owl:minCardinality>
1017       <owl:onProperty>
1018         <owl:ObjectProperty rdf:ID="isSpecifiedBy"/>
1019       </owl:onProperty>
1020     </owl:Restriction>
1021   </rdfs:subClassOf>
1022 </owl:Class>
1023
1024 <owl:Class rdf:about="#ServiceContract">
1025   <rdfs:subClassOf>
1026     <owl:Restriction>
1027       <owl:onProperty>
1028         <owl:ObjectProperty rdf:ID="specifies"/>
1029       </owl:onProperty>
1030       <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
1031         >1</owl:minCardinality>
1032     </owl:Restriction>
1033   </rdfs:subClassOf>
1034 </owl:Class>

```

1035 While a service intrinsically has an effect every time someone interacts with it, in order to trust the effect
1036 to be something in particular, the effect needs to be specified as part of a service contract. The **specifies**
1037 property, and its inverse **isSpecifiedBy**, capture the abstract notion of a service contract specifying a
1038 particular effect as part of the agreement for using a service. Note that the specified effect can apply to
1039 both the **interactionAspect** datatype property (simply specifying what will happen when interacting
1040 with the service according to the service contract) and the **legalAspect** datatype property (specifying a
1041 contractually promised effect).

1042 Anyone wanting a guaranteed effect of the interaction with a given service must ensure that the desired
1043 effect is specified in a service contract applying to that interaction. By definition, any service contract
1044 must specify at least one effect. In the other direction, an effect must be an effect of at least one service
1045 contract; this represents that fact that we have chosen only to formalize those effects that are specified by
1046 service contracts (and not all intrinsic effects of all services).

1047 **7.10 ServiceContract – Examples**1048 **7.10.1 Service-Level Agreements**

1049 A Service-Level Agreement (SLA) on a service has been agreed by organizations A and B. It is important to
 1050 realize that an SLA always has a context of the parties that have agreed to it, involving at a minimum one
 1051 legal “consumer” and one legal “provider”. This can be represented in the ontology as follows:

- 1052 • *A* and *B* are instances of **HumanActor**.
- 1053 • *Service* is an instance of **Service**.
- 1054 • *ServiceContract* is an instance of **ServiceContract**.
- 1055 • *ServiceContract* isContractFor *Service*.
- 1056 • *ServiceContract* involvesParty *A*.
- 1057 • *ServiceContract* involvesParty *B*.
- 1058 • The **legalAspect** datatype property on *ServiceContract* describes the SLA.

1059 **7.10.2 Service Sourcing**

1060 Organizations A and B have agreed on B providing certain services for A, yet B wants to source the actual
 1061 delivery of those services to third party C. This can be represented in the ontology as follows:

- 1062 • *A*, *B*, and *C* are instances of **HumanActor**.
- 1063 • *Service* is an instance of **Service**.
- 1064 • *C* provides *Service*.
- 1065 • *ServiceContract* is an instance of **ServiceContract**.
- 1066 • *ServiceContract* isContractFor *Service*.
- 1067 • *ServiceContract* involvesParty *A*.
- 1068 • *ServiceContract* involvesParty *B*.
- 1069 • The **legalAspect** datatype property on *ServiceContract* describes the legal obligation of B to provide
 1070 *Service* for *A*.

1071 **7.10.3 Car Wash Example**

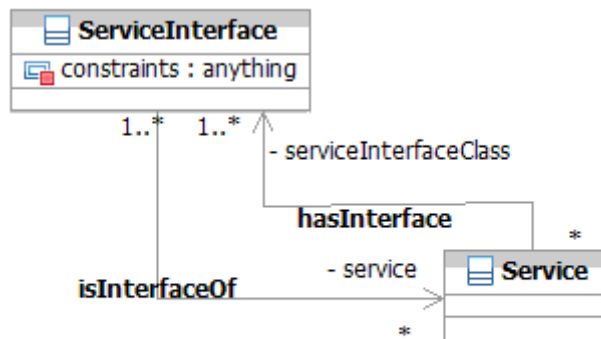
1072 See Clause 11.2 for the complete **Service** and **ServiceContract** aspects of the car wash example.

1073 **7.11 The ServiceInterface Class**

```

1074 <owl:Class rdf:about="#ServiceInterface">
1075   <owl:disjointWith>
1076     <owl:Class rdf:ID="Service"/>
1077   </owl:disjointWith>
1078   <owl:disjointWith>
1079     <owl:Class rdf:ID="ServiceContract"/>
1080   </owl:disjointWith>
1081   <owl:disjointWith>
1082     <owl:Class rdf:ID="Effect"/>
1083   </owl:disjointWith>
1084   <owl:disjointWith>
1085     <owl:Class rdf:ID="HumanActor"/>
1086   </owl:disjointWith>
1087   <owl:disjointWith>
1088     <owl:Class rdf:ID="Task"/>
1089   </owl:disjointWith>
1090 </owl:Class>
  
```

1091 An important characteristic of services is that they have simple, well-defined interfaces. This makes it
 1092 easy to interact with them, and enables other elements to use them in a structured manner. A *service*
 1093 *interface* defines the way in which other elements can interact and exchange information with a service.
 1094 This concept is captured by the **ServiceInterface** class which is illustrated [below \(in Figure 9\)](#).



1095

1096 **Figure 9: The ServiceInterface Class**

1097 The concept of an interface is in general well understood by practitioners, including the notion that
 1098 interfaces define the parameters for information going in and out of them when invoked. What differs

1099 from domain to domain is the specific nature of how an interface is invoked and how information is
 1100 passed back and forth. Service interfaces are typically, but not necessarily, message-based (to support
 1101 loose coupling). Furthermore, service interfaces are always defined independently from any service
 1102 implementing them (to support loose coupling and service mediation).

1103 From a design perspective interfaces may have more granular operations or may be composed of other
 1104 interfaces. We have chosen to stay at the concept level and not include such design aspects in the
 1105 ontology.

1106 **ServiceInterface** is defined as disjoint with the **Service**, **ServiceContract**, and **Effect** classes. Instances
 1107 of these classes are considered not to define (by themselves) the way in which other elements can
 1108 interact and exchange information with a service. Note that that there is a natural synergy between
 1109 **ServiceInterface** and the **interactionAspect** datatype property on **ServiceContract**, as the latter defines
 1110 any multi-interaction and/or sequencing constraints on how to use a service through interaction with its
 1111 service interfaces.

1112 7.11.1 The Constraints Datatype Property

```

1113 <owl:DatatypeProperty rdf:about="#constraints">
1114   <rdfs:domain rdf:resource="#ServiceInterface"/>
1115 </owl:DatatypeProperty>
1116
1117 <owl:Class rdf:about="#ServiceInterface">
1118   <rdfs:subClassOf>
1119     <owl:Restriction>
1120       <owl:onProperty>
1121         <owl:DatatypeProperty rdf:ID="constraints"/>
1122       </owl:onProperty>
1123       <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
1124         >1</owl:minCardinality>
1125     </owl:Restriction>
1126   </rdfs:subClassOf>
1127   <rdfs:subClassOf>
1128     <owl:Restriction>
1129       <owl:onProperty>
1130         <owl:DatatypeProperty rdf:about="#constraints"/>
1131       </owl:onProperty>
1132       <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
1133         >1</owl:maxCardinality>
1134     </owl:Restriction>
1135   </rdfs:subClassOf>
1136 </owl:Class>

```

1137 The **Constraints** datatype property on **ServiceInterface** captures the notion that there can be constraints
 1138 on the allowed interaction such as only certain value ranges allowed on given parameters. Depending on
 1139 the nature of the service and the service interface in question these constraints may be defined either
 1140 formally or informally (the informal case being relevant at a minimum for certain types of real-world
 1141 services).

1142 **7.12 The hasInterface and isInterfaceOf Properties**

```

1143 <owl:ObjectProperty rdf:about="#hasInterface">
1144   <rdfs:domain rdf:resource="#Service"/>
1145   <rdfs:range rdf:resource="#ServiceInterface"/>
1146 </owl:ObjectProperty>
1147
1148 <owl:ObjectProperty rdf:ID="isInterfaceOf">
1149   <owl:inverseOf>
1150     <owl:ObjectProperty rdf:about="#hasInterface"/>
1151   </owl:inverseOf>
1152 </owl:ObjectProperty>
1153
1154 <owl:Class rdf:about="#Service">
1155   <rdfs:subClassOf>
1156     <owl:Restriction>
1157       <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
1158         >1</owl:minCardinality>
1159       <owl:onProperty>
1160         <owl:ObjectProperty rdf:ID="hasInterface"/>
1161       </owl:onProperty>
1162     </owl:Restriction>
1163   </rdfs:subClassOf>
1164 </owl:Class>

```

1165 The **hasInterface** property, and its inverse **isInterfaceOf**, capture the abstract notion of a service having
 1166 a particular service interface.

1167 In one direction, any service must have at least one service interface; anything else would be contrary to
 1168 the definition of a service as a representation of a repeatable activity that has a specified outcome and is
 1169 a ‘black box’ to its consumers. In the other direction, there can be service interfaces that are not yet
 1170 interfaces of any defined services. Also, the same service interface can be an interface of multiple
 1171 services. The latter does not mean that these services are the same, nor even that they have the same
 1172 effect; it only means that it is possible to interact with all these services in the manner defined by the
 1173 service interface in question.

1174 **7.13 The InformationType Class**

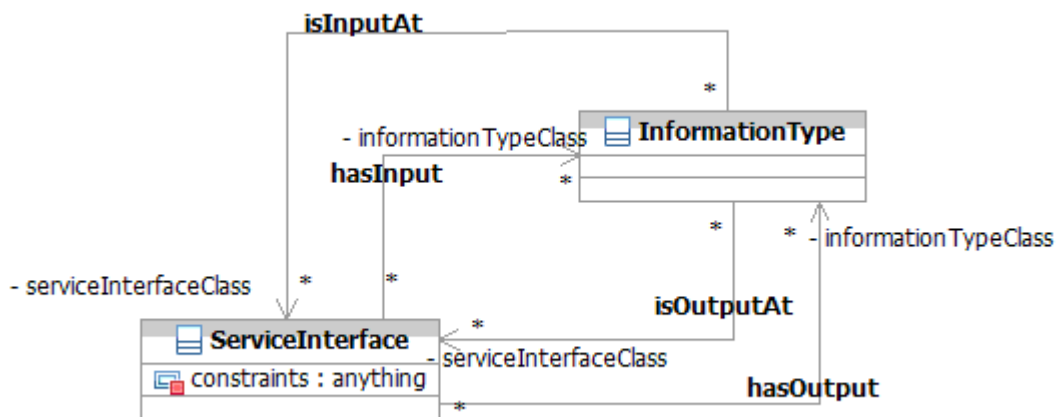
```

1175 <owl:Class rdf:ID="InformationType">
1176   <owl:disjointWith>
1177     <owl:Class rdf:ID="Effect"/>
1178   </owl:disjointWith>
1179   <owl:disjointWith>
1180     <owl:Class rdf:ID="ServiceContract"/>
1181   </owl:disjointWith>
1182 </owl:Class>

```

1183 A service interface can enable another element to give information to or receive information from a
 1184 service (when it uses that service); specifically the types of information given or received. The concept

1185 of *information type* is captured by the **InformationType** OWL class, which is illustrated [below \(in](#)
 1186 [Figure 10](#)).



1187

1188

Figure 10: The InformationType Class

1189 In any concrete interaction through a service interface the information types on that interface are
 1190 instantiated by information items, yet for the service interface itself it is the types that are important.
 1191 Note that the **constraints** datatype property on **ServiceInterface**, if necessary, can be used to express
 1192 constraints on allowed values for certain information types.

1193 7.14 The hasInput and isInputAt Properties

```

1194 <owl:ObjectProperty rdf:ID="hasInput">
1195   <rdfs:domain rdf:resource="#ServiceInterface"/>
1196   <rdfs:range rdf:resource="#InformationType"/>
1197 </owl:ObjectProperty>
1198
1199 <owl:ObjectProperty rdf:ID="isInputAt">
1200   <owl:inverseOf>
1201     <owl:ObjectProperty rdf:ID="hasInput"/>
1202   </owl:inverseOf>
1203 </owl:ObjectProperty>
  
```

1204 The **hasInput** property, and its inverse **isInputAt**, capture the abstract notion of a particular type of
 1205 information being given when interacting with a service through a service interface.

1206 Note that there is a many-to-many relationship between service interfaces and input information types. A
 1207 given information type may be input at many service interfaces or none at all. Similarly, a given service
 1208 interface may have many information types as input or none at all. It is important to realize that some
 1209 services may have only inputs (triggering an asynchronous action without a defined response) and other

1210 services may have only outputs (elements performing these services execute independently yet may
1211 provide output that is used by other elements).

1212 7.15 The **hasOutput** and **isOutputAt** Properties

```
1213 <owl:ObjectProperty rdf:ID="hasOutput">  
1214   <rdfs:domain rdf:resource="#ServiceInterface"/>  
1215   <rdfs:range rdf:resource="#InformationType"/>  
1216 </owl:ObjectProperty>  
1217  
1218 <owl:ObjectProperty rdf:ID="isOutputAt">  
1219   <owl:inverseOf>  
1220     <owl:ObjectProperty rdf:ID="hasOutput"/>  
1221   </owl:inverseOf>  
1222 </owl:ObjectProperty>
```

1223 The **hasOutput** property, and its inverse **isOutputAt**, capture the abstract notion of a particular type of
1224 information being received when interacting with a service through a service interface.

1225 Note that there is a many-to-many relationship between service interfaces and output information types.
1226 A given information type may be output at many service interfaces or none at all. Similarly, a given
1227 service interface may have many information types as output or none at all. It is important to realize that
1228 some services may have only inputs (triggering an asynchronous action without a defined response) and
1229 other services may have only outputs (elements performing these services execute independently yet
1230 may provide output that is used by other elements).

1231 7.16 Examples

1232 7.16.1 Interaction Sequencing

1233 A service contract on a service expresses that the services interfaces on that services must be used in a
1234 certain order:

- 1235 • *Service* is an instance of **Service**.
- 1236 • *ServiceContract* is an instance of **ServiceContract**.
- 1237 • *ServiceContract* isContractFor *Service*.
- 1238 • *X* is an instance of *ServiceInterface*.
- 1239 • *X* isInterfaceOf *Service*.
- 1240 • *Y* is an instance of *ServiceInterface*.

1241 • *Y* isInterfaceOf *Service*.

1242 • The **interactionAspect** datatype property on *ServiceContract* describes that *X* must be used before
1243 *Y* may be used.

1244 7.16.2 Car Wash Example

1245 See Clause 11.2 for the complete **ServiceInterface** aspect of the car wash example.

1246

1247 8 Composition and its Subclasses

1248 8.1 Introduction

1249 The notion of Composition is a core concept of SOA. Services can be composed of other services.
1250 Processes are composed of human actors, tasks, and possibly services. Experienced SOA practitioners
1251 intuitively apply composition as an integral part of architecting, designing, and realizing SOA systems;
1252 in fact, any well structured SOA environment is intrinsically composite in the way services and
1253 processes support business capabilities. What differs from practitioner to practitioner is the exact nature
1254 of the composition – the composition pattern being applied.

1255 This clause describes the following classes of the ontology:

1256 **Composition** (as a subclass of **System**)

1257 **ServiceComposition** (as a subclass of **Composition**)

1258 **Process** (as a subclass of **Composition**)

1259 In addition, it defines the following datatype property:

1260 **compositionPattern**

1261 8.2 The Composition Class

```
1262 <owl:Class rdf:about="#Composition">
```

```
1263   <rdfs:subClassOf>
```

```
1264     <owl:Class rdf:ID="System"/>
```

```
1265   </rdfs:subClassOf>
```

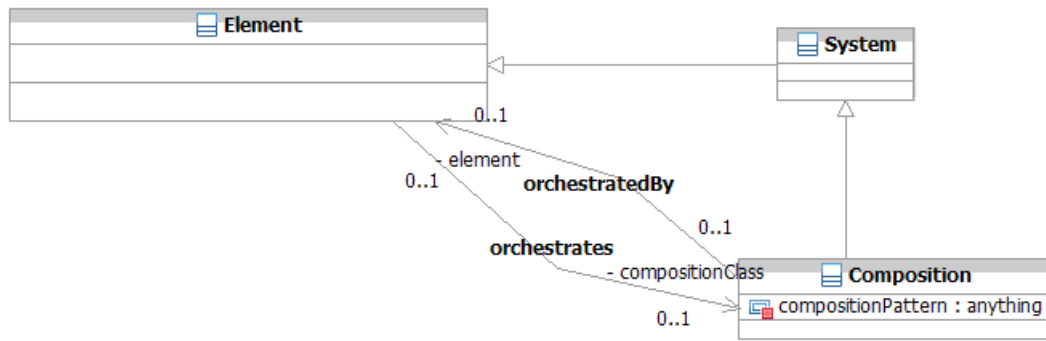
```
1266   <owl:disjointWith>
```

```

1267     <owl:Class rdf:ID="Task"/>
1268
1269 </owl:disjointWith>
1270
1271 </owl:Class>

```

1270 A *composition* is the result of assembling a collection of things for a particular purpose. Note in
1271 particular that we have purposefully distinguished between the act of composing and the resulting
1272 composition as a thing, and that it is in the latter sense we are using the concept of *composition* here. The
1273 concept of *composition* is captured by the **Composition** OWL class, which is illustrated [below \(in Figure](#)
1274 [11\)](#).



1275

1276

Figure 11: The Composition Class

1277 Being intrinsically (also) an organized collection of other, simpler things, the **Composition** class is a
1278 subclass of the **System** class. While a composition is always also a system, a system is not necessarily a
1279 composition in that it is not necessarily a result of anything – note here the difference between a system
1280 producing a result and the system itself being a result. A perhaps more tangible difference between a
1281 system and a composition is that the latter must have associated with it a specific composition pattern
1282 that renders the composition (as a whole) as the result when that composition pattern is applied to the
1283 elements used in the composition. One implication of this is that there is not a single member of a
1284 composition that represents (as an element) that composition as a whole; in other words, the composition
1285 itself is not one of the things being assembled. On the other hand, *composition* is in fact a recursive
1286 concept (as are all subclasses of **System**) – being a system, a composition is also an element which
1287 means that it can be used by a higher-level composition.

1288 In the context of the SOA ontology we consider in detail only functional compositions that belong to the
1289 SOA domain. Note that a fully described instance of **Composition** must have by its nature a *uses*
1290 relationship to at least one instance of **Element**. (It need not necessarily have more than one as the
1291 composition pattern applied may be, for instance, simply a transformation.) Again (as for **System**) it is
1292 important to realize that a composition can use elements outside its own boundary.

1293 Since **Composition** is a subclass of **Element**, all compositions have a boundary and are opaque to an
1294 external observer (black box view). The composition pattern in turn is the internal view point (white box
1295 view) of a composition. As an example, for the notion of a service composition this would correspond to

1296 the difference between seeing the service composition as an element providing a (higher-level) service or
 1297 seeing the service composition as a composite structure of (lower-level) services.

1298 **8.2.1 The compositionPattern Datatype Property**

```

1299 <owl:DatatypeProperty rdf:about="#compositionPattern">
1300     <rdfs:domain rdf:resource="#Composition"/>
1301 </owl:DatatypeProperty>
1302
1303 <owl:Class rdf:about="#Composition">
1304     <rdfs:subClassOf>
1305         <owl:Restriction>
1306             <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
1307                 >1</owl:maxCardinality>
1308             <owl:onProperty>
1309                 <owl:DatatypeProperty rdf:ID="compositionPattern"/>
1310             </owl:onProperty>
1311         </owl:Restriction>
1312     </rdfs:subClassOf>
1313     <rdfs:subClassOf>
1314         <owl:Restriction>
1315             <owl:onProperty>
1316                 <owl:DatatypeProperty rdf:ID="compositionPattern"/>
1317             </owl:onProperty>
1318             <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
1319                 >1</owl:minCardinality>
1320         </owl:Restriction>
1321     </rdfs:subClassOf>

```


1322 </owl:Class>

1323 As discussed, above any composition must have associated with it a specific composition pattern, that
 1324 pattern describing the way in which a collection of elements is assembled to a result. The concept of a
 1325 composition pattern is captured by the **compositionPattern** datatype property. Note that even though
 1326 certain kinds of composition patterns are of special interest within SOA (see below), the
 1327 **compositionPattern** data type property may take any value as long as that value describes how to
 1328 assemble the elements used by the composition with which it is associated.

1329 **• The Orchestration Composition Pattern**

1330 One kind of composition pattern that has special interest within SOA is an *Orchestration*. In an
 1331 orchestration (a composition whose composition pattern is an orchestration), there is one particular
 1332 element used by the composition that oversees and directs the other elements. Note that the element that
 1333 directs an orchestration by definition is different than the orchestration (**Composition** instance) itself.

1334 Think of an orchestrated executable workflow as an example of an orchestration. The workflow
 1335 construct itself is one of the elements being used in the composition, yet it is different from the
 1336 composition itself – the composition itself is the result of applying (executing) the workflow on the
 1337 processes, human actors, services, etc. that are orchestrated by the workflow construct.

1338 A non-IT example is the foreman of a road repair crew. If the foreman chooses to exert direct control
 1339 over the tasks done by his crew, than the resulting composition becomes an orchestration (with the
 1340 foreman as the director and provider of the composition pattern). Note that under other circumstances,
 1341 with a different team composition model, a road repair crew can also act as a collaboration or a
 1342 choreography. (See below for definitions of collaboration and choreography.)

1343 As the last example clearly shows, using an orchestration composition pattern is not a guarantee that
 1344 “nothing can go wrong”. That would, in fact, depend on the orchestration director’s ability to handle
 1345 exceptions.

1346 **• The Choreography Composition Pattern**

1347 Another kind of composition pattern that has special interest within SOA is a *Choreography*. In a
 1348 choreography (a composition whose composition pattern is a choreography) the elements used by the
 1349 composition interact in a non-directed fashion, yet with each autonomous member knowing and
 1350 following a predefined pattern of behavior for the entire composition.

1351 Think of a process model as an example of a choreography. The process model does not direct the
 1352 elements within it, yet does provide a predefined pattern of behavior that each such element is expected
 1353 to conform to when “executing”.

1354 **• The Collaboration Composition Pattern**

1355 A third kind of composition pattern that has special interest within SOA is a *Collaboration*. In a
 1356 collaboration (a composition whose composition pattern is a collaboration) the elements used by the
 1357 composition interact in a non-directed fashion, each according to their own plans and purposes without a
 1358 predefined pattern of behavior. Each element simply knows what it has to do and does it independently,

1359 initiating interaction with the other members of the composition as applicable on its own initiative. This
 1360 means that there is no overall predefined “flow” of the collaboration, though there may be a run-time
 1361 “observed flow of interactions”.

1362 A good example of a collaboration is a work meeting. There is no script for how the meeting will unfold
 1363 and only after the meeting has concluded can we describe the sequence of interactions that actually
 1364 occurred.

1365 **8.3 The orchestrates and orchestratedBy Properties**

1366 <owl:ObjectProperty rdf:about="#orchestratedBy">

1367 <rdfs:domain rdf:resource="#Composition"/>

1368 <rdfs:range rdf:resource="#Element"/>

1369 </owl:ObjectProperty>

1370

1371 <owl:ObjectProperty rdf:about="#orchestrates">

1372 <owl:inverseOf>

1373 <owl:ObjectProperty rdf:ID="orchestratedBy"/>

1374 </owl:inverseOf>

1375 </owl:ObjectProperty>

1376

1377 <owl:Class rdf:about="#Composition">

1378 <rdfs:subClassOf>

1379 <owl:Restriction>

1380 <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"

1381 >1</owl:maxCardinality>

1382 <owl:onProperty>

1383 <owl:ObjectProperty rdf:ID="orchestratedBy"/>

1384 </owl:onProperty>

1385 </owl:Restriction>

1386 </rdfs:subClassOf>

```
1387     <rdfs:subClassOf>
1388         <owl:Restriction>
1389             <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
1390                 >0</owl:minCardinality>
1391             <owl:onProperty>
1392                 <owl:ObjectProperty rdf:ID="orchestratedBy"/>
1393             </owl:onProperty>
1394         </owl:Restriction>
1395     </rdfs:subClassOf>
1396 </owl:Class>
1397
1398 <owl:Class rdf:about="#Element">
1399     <rdfs:subClassOf>
1400         <owl:Restriction>
1401             <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
1402                 >0</owl:minCardinality>
1403             <owl:onProperty>
1404                 <owl:ObjectProperty rdf:ID="orchestrates"/>
1405             </owl:onProperty>
1406         </owl:Restriction>
1407     </rdfs:subClassOf>
1408     <rdfs:subClassOf>
1409         <owl:Restriction>
1410             <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
1411                 >1</owl:maxCardinality>
1412             <owl:onProperty>
```

```

1413         <owl:ObjectProperty rdf:about="#orchestrates"/>
1414     </owl:onProperty>
1415 </owl:Restriction>
1416 </rdfs:subClassOf>
1417 </owl:Class>

```

1418 As defined above, an orchestration has one particular element that oversees and directs the other
 1419 elements used by the composition. This type of relationship is important enough that we have chosen to
 1420 capture the abstract notion in the **orchestrates** property and its inverse **orchestratedBy**.

1421 In one direction, a composition has at most one element that orchestrates it, and the cardinality can only
 1422 be 1 if in fact the composition pattern of that composition is an orchestration. In the other direction, an
 1423 element can orchestrate at most one composition which then must have an orchestration as its
 1424 composition pattern.

1425 Note that in practical applications of the ontology, even though **Service** is a subclass of **Element**, a
 1426 service (as a purely logical representation) is not expected to orchestrate a composition.

1427 **8.4 The ServiceComposition Class**

```

1428 <owl:Class rdf:ID="ServiceComposition">
1429     <rdfs:subClassOf>
1430         <owl:Class rdf:ID="Composition"/>
1431     </rdfs:subClassOf>
1432     <owl:disjointWith>
1433         <owl:Class rdf:ID="ServiceContract"/>
1434     </owl:disjointWith>
1435     <owl:disjointWith>
1436         <owl:Class rdf:ID="ServiceInterface"/>
1437     </owl:disjointWith>
1438 </owl:Class>

```

1439 A key SOA concept is the notion of *service composition*, the result of assembling a collection of services
 1440 in order to perform a new higher-level service. The concept of *service composition* is captured by the
 1441 **ServiceComposition** OWL class, which is illustrated [below \(in Figure 12\)](#).

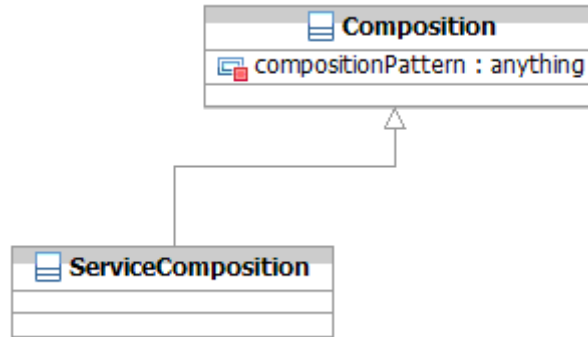


Figure 12: The ServiceComposition Class

1442

1443

1444 As a *service composition* is the result of assembling a collection of services, **ServiceComposition** is
 1445 naturally a subclass of **Composition**.

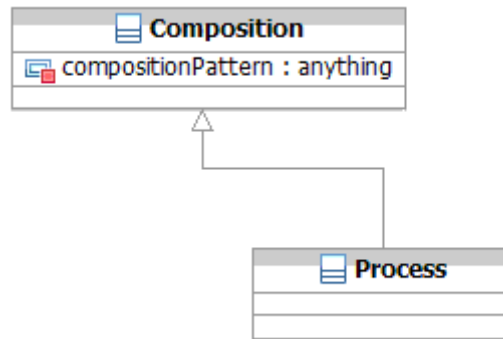
1446 A service composition may, and typically will, add logic (or even “code”) via the composition pattern.
 1447 Note that a service composition is *not* the new higher-level service itself (due to the **System** and **Service**
 1448 classes being disjoint); rather it performs (as an element) that higher-level service.

1449 **8.5 The Process Class**

```

1450 <owl:Class rdf:ID="Process">
1451   <rdfs:subClassOf>
1452     <owl:Class rdf:ID="Composition"/>
1453   </rdfs:subClassOf>
1454   <owl:disjointWith>
1455     <owl:Class rdf:ID="ServiceContract"/>
1456   </owl:disjointWith>
1457   <owl:disjointWith>
1458     <owl:Class rdf:ID="ServiceInterface"/>
1459   </owl:disjointWith>
1460 </owl:Class>
  
```

1461 Another key SOA concept is the notion of *process*. A *process* is a composition whose elements are
 1462 composed into a sequence or flow of activities and interactions with the objective of carrying out certain
 1463 work. This definition is consistent with, for instance, the Business Process Modeling Notation (BPMN)
 1464 2.0 definition of a process. The concept of *process* is captured by the **Process** OWL class, which is
 1465 illustrated [below](#) (in Figure 13).



1466

1467

Figure 13: The Process Class

1468 Elements in process compositions can be things like human actors, tasks, services, other processes, etc.
 1469 A process always adds logic via the composition pattern; the result is more than the parts. According to
 1470 their collaboration pattern, processes can be:

1471 **Orchestrated:** When a process is orchestrated in a Business Process Management System, then the
 1472 resulting IT artifact is in fact an orchestration; i.e., it has an orchestration collaboration pattern.
 1473 This type of process is often called a “Process Orchestration”.

1474 **Choreographed:** For example, a process model representing a defined pattern of behavior. This type of
 1475 process is often called a “Process Choreography”.

1476 **Collaborative:** No (pre)defined pattern of behavior (model); the process represents observed
 1477 (executed) behavior.

1478 8.6 Service Composition and Process Examples

1479 8.6.1 Simple Service Composition Example

1480 Using a service composition example, services A and B are instances of **Service** and the composition of A
 1481 and B is an instance of **ServiceComposition** (that uses A and B):

1482 A and B are instances of **Service**.

1483 X is an instance of **ServiceComposition**.

1484 X uses both A and B (composes them according to its service composition pattern).

1485 Note that there are various ways in which the service composition pattern can compose A and B, all of
 1486 which are relevant in one situation or another. For example, interfaces of X may or may not include some
 1487 subset of the interfaces of A and B. Furthermore, the interfaces of A and B may or may not also be
 1488 (directly) invocable without going through X – that is, a matter of the service contracts and/or access

1489 policies apply to the A and B. Finally, X may also use other elements that are not services at all (examples
1490 are composition code, adaptors, etc.).

1491 **8.6.2 Process Example**

1492 Using a process example, tasks T1 and T2 are instances of **Task**, roles R1 and R2 are instances of **Element**,
1493 and the composition of T1, T2, R1, and R2 is an instance of **Process** (that uses T1, T2, R1, and R2):

1494 *T1* and *T2* are instances of **Task**.

1495 *R1* and *R2* are instances of **Element**.

1496 *Y* is an instance of **Process**.

1497 *Y* uses all of *T1*, *T2*, *R1*, and *R2* (composes them according to its process composition pattern).

1498

1499 **8.6.3 Process and Service Composition Example**

1500 Elaborating on the process example above, if T1 is done using service S then:

1501 *S* is an instance of **Service**.

1502 *T1* uses *S*.

1503 Note that depending on the particular design approach chosen (and the resulting composition pattern), *Y*
1504 may or may not use *S* directly. This depends on whether *Y* carries the binding between T1 and S or whether
1505 that binding is encapsulated in T1.

1506 **8.6.4 Car Wash Example**

1507 See Clause 11.4 for the **Process** aspect of the car wash example.

1508 **9 Policy**

1509 **9.1 Introduction**

1510 Policies, the human actors defining them, and the things that they apply to are important aspects of any
1511 system, certainly also SOA systems with their many different interacting elements. Policies can apply to
1512 any element in a system. The concept of *Policy* is captured by the **Policy** class and its relationships to the
1513 **HumanActor** and **Thing** classes.

1514 This clause describes the following classes of the ontology:

1515 **Policy**

1516 In addition, it defines the following properties:

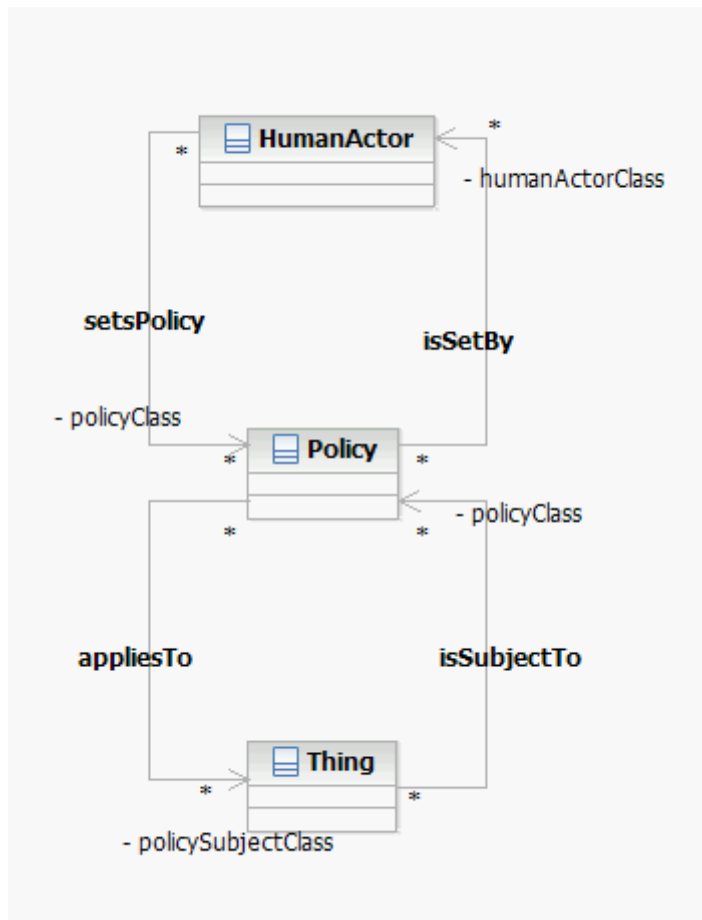
1517 **appliesTo** and **isSubjectTo**1518 **setsPolicy** and **isSetBy**1519 **9.2 The Policy Class**

```

1520 <owl:Class rdf:about="#Policy">
1521   <owl:disjointWith>
1522     <owl:Class rdf:ID="InformationType"/>
1523   </owl:disjointWith>
1524   <owl:disjointWith>
1525     <owl:Class rdf:ID="ServiceInterface"/>
1526   </owl:disjointWith>
1527   <owl:disjointWith>
1528     <owl:Class rdf:ID="Element"/>
1529   </owl:disjointWith>
1530   <owl:disjointWith>
1531     <owl:Class rdf:ID="Effect"/>
1532   </owl:disjointWith>
1533   <owl:disjointWith>
1534     <owl:Class rdf:ID="Event"/>
1535   </owl:disjointWith>
1536   <owl:disjointWith>
1537     <owl:Class rdf:ID="ServiceContract"/>
1538   </owl:disjointWith>
1539 </owl:Class>

```

1540 A *policy* is a statement of direction that a human actor may intend to follow or may intend that another
 1541 human actor should follow. Knowing the policies that apply to something makes it easier and more
 1542 transparent to interact with that something. The concept of *policy* is captured by the **Policy** OWL class,
 1543 which is illustrated [below](#) (in Figure 14).



1544

1545

Figure 14: The Policy Class

1546 *Policy* as a concept is generic and has relevance outside the domain of SOA. For the purposes of this
 1547 SOA ontology it has not been necessary or relevant to restrict the generic nature of the **Policy** class itself.
 1548 The relationships between **Policy** and **HumanActor** are of course bound by the SOA-specific
 1549 restrictions that have been applied on the definition of **HumanActor**.

1550 From a design perspective policies may have more granular parts or may be expressed and made
 1551 operational through specific rules. We have chosen to stay at the concept level and not include such
 1552 design aspects in the ontology.

1553 *Policy* is distinct from all other concepts in this ontology, hence the **Policy** class is defined as disjoint
 1554 with all other defined classes. In particular, **Policy** is disjoint with **ServiceContract**. While policies may
 1555 apply to service contracts – such as security policies on who may change a given service contract – or
 1556 conversely be referred to by service contracts as part of the terms, conditions, and interaction rules that
 1557 interacting participants must agree to, service contracts are themselves not policies as they do not
 1558 describe an intended course of action.

1559 **9.2.1 The appliesTo and isSubjectTo Properties**

```

1560 <owl:ObjectProperty rdf:ID="appliesTo">
1561   <rdfs:domain rdf:resource="#Policy"/>
1562 </owl:ObjectProperty>
1563
1564 <owl:ObjectProperty rdf:ID="isSubjectTo">
1565   <owl:inverseOf>
1566     <owl:ObjectProperty rdf:ID="appliesTo"/>
1567   </owl:inverseOf>
1568 </owl:ObjectProperty>

```

1569 Policies can apply to things other than elements; in fact, policies can apply to anything at all, including
 1570 other policies. For instance, a security policy might specify which actors have the authority to change
 1571 some other policy. The **appliesTo** property, and its inverse **isSubjectTo**, capture the abstract notion that
 1572 a policy can apply to any instance of **Thing**. Note specifically that **Element** is a subclass of **Thing**,
 1573 hence policies by inference can apply to any instance of **Element**.

1574 In one direction, a policy can apply to zero (in the case where a policy has been formulated but not yet
 1575 explicitly applied to anything), one, or more instances of **Thing**. Note that having a policy apply to
 1576 multiple things does not mean that these things are the same, only that they are (partly) regulated by the
 1577 same intent. In the other direction, an instance of **Thing** may be subject to zero, one, or more policies.
 1578 Note that where multiple policies apply to the same instance of **Thing** this is often because the multiple
 1579 policies are from multiple different policy domains (such as security and governance).

1580 The SOA ontology does not attempt to enumerate different policy domains; such policy-focused details
 1581 are deemed more appropriate for a policy ontology. It is worth pointing out that a particular policy
 1582 ontology may also restrict (if desired) the kinds of things that policies can apply to.

1583 **9.3 The setsPolicy and isSetBy Properties**

```

1584 <owl:ObjectProperty rdf:about="#setsPolicy">
1585   <rdfs:domain rdf:resource="#HumanActor"/>
1586   <rdfs:range rdf:resource="#Policy"/>
1587 </owl:ObjectProperty>
1588
1589 <owl:ObjectProperty rdf:ID="isSetBy">
1590   <owl:inverseOf>
1591     <owl:ObjectProperty rdf:ID="setsPolicy"/>
1592   </owl:inverseOf>
1593 </owl:ObjectProperty>

```

1594 The **setsPolicy** property, and its inverse **isSetBy**, capture the abstract notion that a policy can be set by
 1595 one or more human actors.

1596 In one direction, a policy can be set by zero (in the case where actors setting the policy by choice are not
 1597 defined or captured), one, or more human actors. Note specifically that some policies are set by multiple
 1598 human actors in conjunction, meaning that all these human actors need to discuss and agree on the policy
 1599 before it can take effect. A real-world example would be two parents in conjunction setting policies for

1600 acceptable child behavior. In the other direction, a human actor may potentially set (or be part of setting)
1601 multiple policies.

1602 The SOA ontology purposefully separates the setting of the policy itself and the application of the policy
1603 to one or more instances of **Thing**. In some cases these two acts may be inseparably bound together, yet
1604 in other cases they are definitely not. One such example is an overall compliance policy that is
1605 formulated at the corporate level yet applied by the compliance officer in each line of business.

1606 Also, while a particular case of interest for this ontology is that where the provider of a service has a
1607 policy for the service, a policy for a service is not necessarily owned by the provider. For example,
1608 government food and hygiene regulations (a policy that is law) cover restaurant services independently
1609 of anything desired or defined by the restaurant owner.

1610 **9.4 Examples**

1611 **9.4.1 Car Wash Example**

1612 See [The Washing Policies](#) (Clause 11.5) for the **Policy** aspect of the car wash example.

1613 **10 Event**

1614 **10.1 Introduction**

1615 Events and the elements that generate or respond to them are important aspects of any event emitting
1616 system. SOA systems are in fact often event emitting, hence *event* is defined as a concept in the SOA
1617 ontology.

1618 This clause describes the following classes of the ontology:

1619 **Event**

1620 In addition, it defines the following properties:

1621 **generates** and **generatedBy**

1622 **respondsTo** and **respondedToBy**

1623 **10.2 The Event Class**

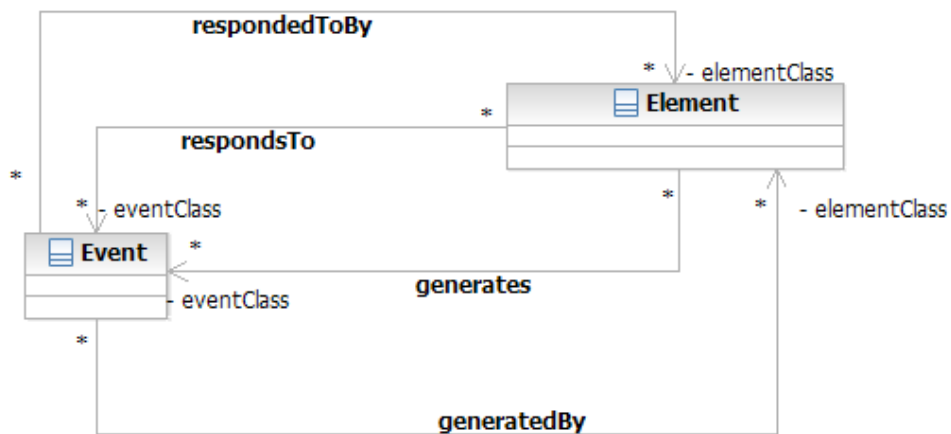
```
1624 <owl:Class rdf:about="#Event">  
1625   <owl:disjointWith>  
1626     <owl:Class rdf:ID="Policy"/>  
1627   </owl:disjointWith>  
1628   <owl:disjointWith>  
1629     <owl:Class rdf:ID="ServiceContract"/>  
1630   </owl:disjointWith>
```

```

1631     <owl:disjointWith>
1632         <owl:Class rdf:ID="ServiceInterface"/>
1633     </owl:disjointWith>
1634 </owl:Class>

```

1635 An *event* is something that happens, to which an element may choose to respond. Events can be
 1636 responded to by any element. Similarly, events may be generated (emitted) by any element. Knowing the
 1637 events generated or responded to by an element makes it easier and more transparent to interact with that
 1638 element. Note that some events may occur whether generated or responded to by an element or not. The
 1639 concept of *event* captured by the **Event** OWL class, which is illustrated below (in Figure 15).



1640

1641

Figure 15: The Event Class

1642 *Event* as a concept is generic and has relevance to the domain of SOA as well as many other domains.
 1643 For the purposes of this ontology, *Event* is used in its generic sense.

1644 From a design perspective events may have more granular parts or may be expressed and made
 1645 operational through specific syntax or semantics. We have chosen to stay at the concept level and not
 1646 include such design aspects in the ontology.

1647 10.3 The generates and generatedBy Properties

```

1648 <owl:ObjectProperty rdf:ID="generates">
1649     <rdfs:domain rdf:resource="#Element"/>
1650     <rdfs:range rdf:resource="#Event"/>
1651 </owl:ObjectProperty>
1652
1653 <owl:ObjectProperty rdf:ID="generatedBy">
1654     <owl:inverseOf>
1655         <owl:ObjectProperty rdf:ID="generates"/>
1656     </owl:inverseOf>
1657 </owl:ObjectProperty>

```

1658 Events can, but need not necessarily, be generated by elements. The **generates** property, and its inverse
1659 **generatedBy**, capture the abstract notion that an element generates an event.

1660 Note that the same event may be generated by many different elements. Similarly, the same element may
1661 generate many different events.

1662 **10.4 The respondsTo and respondedToBy Properties**

```
1663 <owl:ObjectProperty rdf:ID="respondsTo">  
1664   <rdfs:domain rdf:resource="#Element"/>  
1665   <rdfs:range rdf:resource="#Event"/>  
1666 </owl:ObjectProperty>  
1667  
1668 <owl:ObjectProperty rdf:ID="respondedToBy">  
1669   <owl:inverseOf>  
1670     <owl:ObjectProperty rdf:ID="respondsTo"/>  
1671   </owl:inverseOf>  
1672 </owl:ObjectProperty>
```

1673 Events can, but need not necessarily, be responded to by elements. The **respondsTo** property, and its
1674 inverse **respondedToBy**, capture the abstract notion that an element responds to an event.

1675 Note that the same event may be responded to by many different elements. Similarly, the same element
1676 may respond to many different events.

1677 **11 Complete Car Wash Example**

1678 This clause contains the complete car wash example that has been used in parts throughout the
1679 definitional clauses of the ontology.

1680 **11.1 The Organizational Aspect**

1681 Joe the owner chooses to organize his business into two organizational units: *Administration* and *CarWash*:

1682 *CarWashBusiness* is an instance of both **HumanActor** and **System**.

1683 *Administration* is an instance of **HumanActor** (organizational unit).

1684 *CarWash* is an instance of **HumanActor** (organizational unit).

1685 *CarWashBusiness* uses (has organizational units) *Administration* and *CarWash*.

1686 *AdministrativeSystem* is an instance of **System**.

1687 *Administration* represents *AdministrativeSystem*.

1688 *CarWashSystem* is an instance of **System**.

1689 *CarWash* represents *CarWashSystem*.

1690 And using well-defined roles within each organization:

1691 *Owner* (role) is an instance of **Element** and is used by *AdministrativeSystem*.

1692 *Joe* is an instance of **HumanActor** and is represented by (has role) *Owner*.

1693 *Secretary* (role) is an instance of **Element** and is used by *AdministrativeSystem*.

1694 *Mary* is an instance of **HumanActor** and is represented by (has role) *Secretary*.

1695 *PreWashGuy* (role) is an instance of **Element** and is used by *CarWashSystem*.

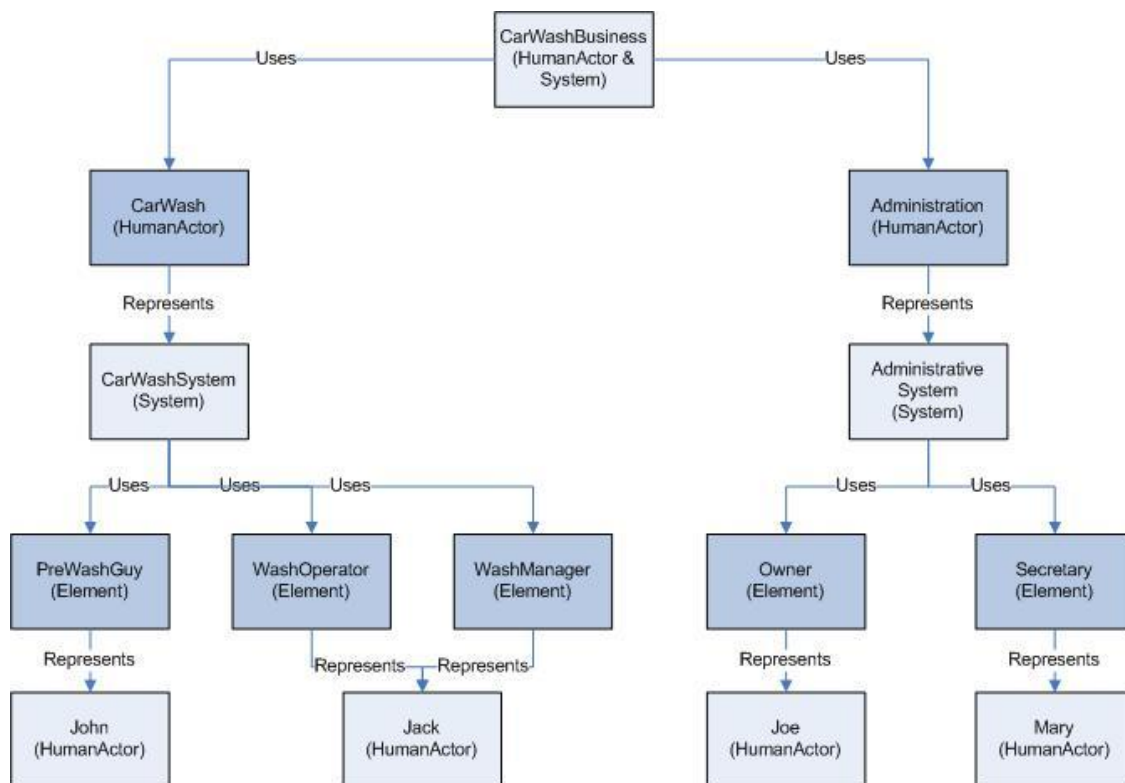
1696 *John* is an instance of **HumanActor** and is represented by (has role) *PreWashGuy*.

1697 *WashManager* (role) is an instance of **Element** and is used by *CarWashSystem*.

1698 *WashOperator* (role) is an instance of **Element** and is used by *CarWashSystem*.

1699 *Jack* is an instance of **HumanActor** and is represented by (has roles) both *WashManager* and

1700 *WashOperator*.



1701

1702

Figure 16: Car Wash Example – The Organizational Aspect

1703 **11.2 The Washing Services**

1704 Joe offers two different services to his customers: a basic wash and a gold wash:

1705 *GoldWash* is an instance of **Service**.

1706 *BasicWash* is an instance of **Service**.

1707 *CarWash* performs both *BasicWash* and *GoldWash*.

1708 *WashManager* represents both *BasicWash* and *GoldWash* (i.e., it is the interaction point where customers
1709 can order services as well as pay for them).

1710 In return for payment, Joe's *BasicWash* service cleans the car of customer Judy:

1711 *Judy* is an instance of **HumanActor** (the customer).

1712 *BasicWashContract* is an instance of **ServiceContract**.

1713 *BasicWash* has contract *BasicWashContract*.

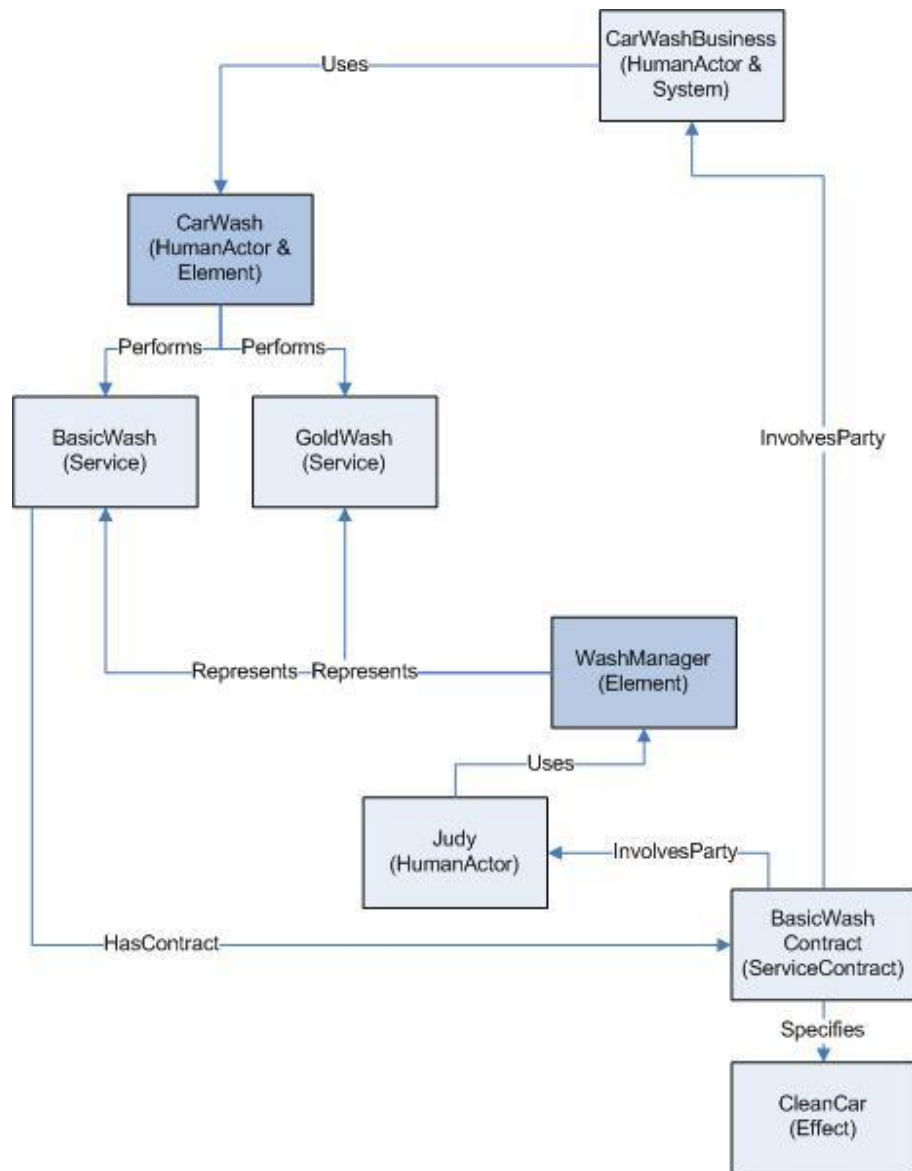
1714 *CleanCar* is an instance of **Effect**.

1715 *BasicWashContract* specifies *CleanCar* as its effect.

1716 *BasicWashContract* involves parties *CarWashBusiness* and *Judy* and specifies that *Judy* (as the legal
1717 consumer) pays *CarWashBusiness* (as the legal provider) \$10 for the one consumption of *BasicWash*
1718 with the effect of (one) *CleanCar*. Note that *BasicWash* is actually performed by *CarWash* and not by
1719 the legal provider *CarWashBusiness* – in this particular example *CarWash* happens to be a member
1720 of *CarWashBusiness* but such need not always be the case, *CarWash* could have been some third
1721 party provider.

1722 *Judy* uses *WashManager* (in order to invoke the *BasicWash* service).

1723 Note that in this example Judy does not interact with the (abstract) *BasicWash* service directly, rather she
1724 interacts with the *WashManager* that represents the service. This is due to Joe deciding that in his car wash
1725 customers are not to interact with the washing machinery directly.



1726

1727

Figure 17: Car Wash Example – The Washing Services

1728 11.3 Interfaces to the Washing Services

1729 The way to interact with the car wash services is simple for the customer; he or she simply gives money
 1730 to the wash manager and asks to have to the car washed using one of the two available wash services.
 1731 Due to the fact that Joe has decided to interpose the wash manager between the customer and the
 1732 washing machine, the customer actually never interacts with the wash services themselves. We could
 1733 have chosen to formally define a proxy service provided by the wash manager but have omitted that
 1734 level of formality in this real-world example.

1735 The wash manager in turn does interact with the wash services through their interfaces defined as
1736 follows:

1737 *WashingMachineInterface* is an instance of **ServiceInterface**.

1738 *TypeOfWash* is an instance of **InformationType**.

1739 *WashingMachineInterface* has input *TypeOfWash*.

1740 *BasicWash* has interface *WashingMachineInterface*.

1741 *GoldWash* has interface *WashingMachineInterface*.

1742 Note how both washing services in fact have the same service interface. Even though Joe has chosen to
1743 offer basic wash and gold wash as two different services, both are in effect done by the same washing
1744 machine (one simply has to choose the type of wash when initializing the washing machine).

1745 11.4 The Washing Processes

1746 An important part of the car wash system is the car washing process itself:

1747 *AutomatedCarWashProcess* is an instance of both **Process** and **Orchestration**.

1748 *Wash* is an instance of **Task** and is used by *AutomatedCarWashProcess*.

1749 *Dry* is an instance of **Task** and is used by *AutomatedCarWashProcess*.

1750 *AutomatedCarWash* is an instance of **Element** (the automated washing machine) and represents
1751 *AutomatedCarWashProcess* (encapsulates the process) as well as directs *AutomatedCarWashProcess*.

1752 *CarWashProcess* is an instance of **Process** and is used by (part of) *CarWashSystem* (no need to create an
1753 explicit opaque building block).

1754 *AutomatedCarWash* is used by *CarWashProcess* (automated activity in the process).

1755 *WashWindows* is an instance of **Task** and is done by *John*.

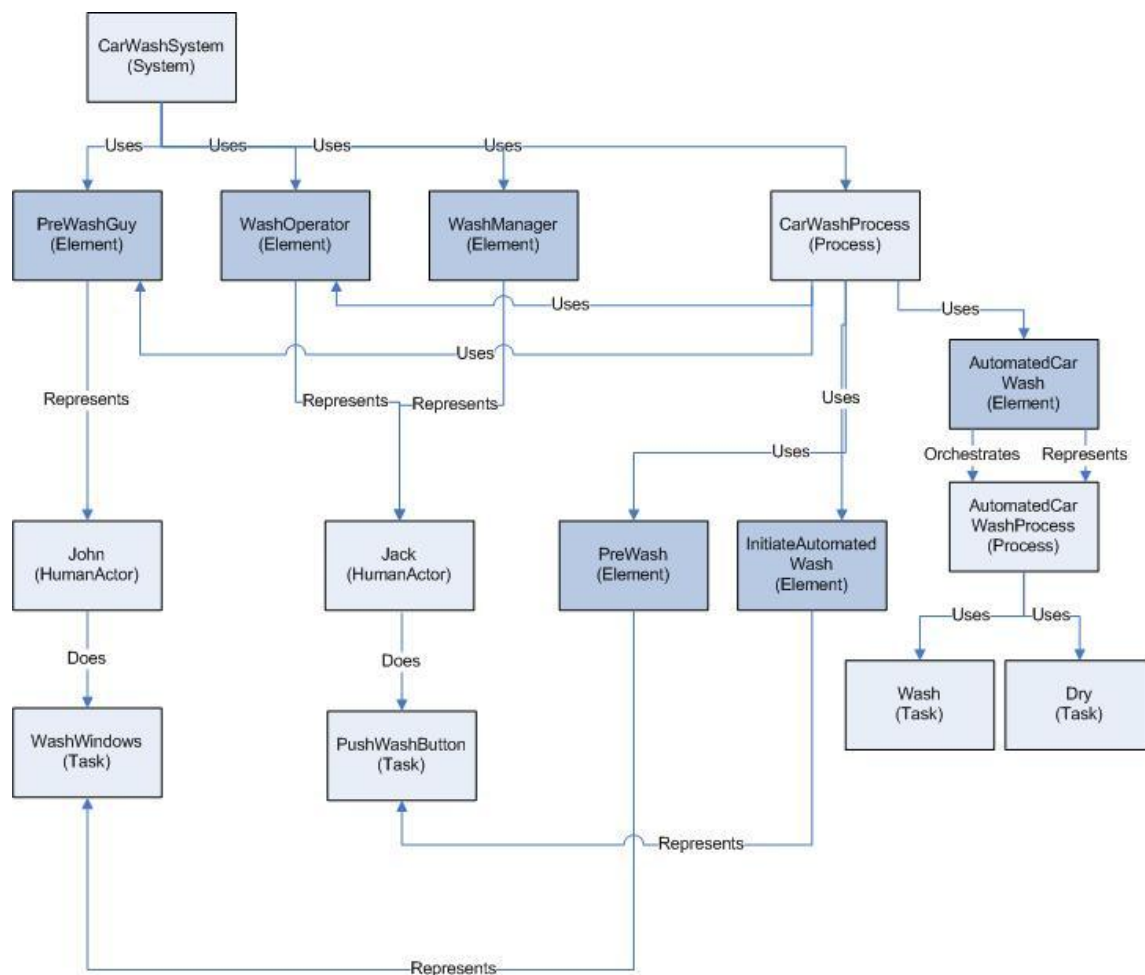
1756 *PreWash* is an instance of **Element**, represents *WashWindows*, and is used by *CarWashProcess* (logical
1757 activity in the process).

1758 *PrewashGuy* is a member of *CarWashProcess* (role in the process).

1759 *PushWashButton* is an instance of **Task** and is done by *Jack*.

1760 *InitiateAutomatedWash* is an instance of **Element**, represents *PushWashButton*, and is used by
1761 *CarWashProcess* (logical activity in the process).

1762 *WashOperator* is a member of *CarWashProcess* (role in the process).



1763
1764
1765 **Figure 18: Car Wash Example – The Washing Processes**

11.5 The Washing Policies

1766 Joe sets a payment up-front policy for the washing services:

1767 *PaymentUpFront* is an instance of both **Policy**.

1768 *PaymentUpFront* is set by *Joe*.

1769 *PaymentUpFront* applies to both *GoldWash* and *BasicWash*.

1770 Note how the *PaymentUpFront* policy enhances the service contract *BasicWashContract*. While
1771 *BasicWashContract* only specifies that *Judy* has to pay \$10 for one consumption of the *BasicWash* service,
1772 the *PaymentUpFront* policy makes it specific that payment has to happen up-front. One of the advantages
1773 of separating policy from service contract is that the payment policy can be changed independently of the
1774 service contract. For instance, at some later point in time Joe may decide that recurring customers need not

1775 pay up-front, and can institute this change in policy without changing anything else related to
1776 *CarWashBusiness*.

1777

1778 12 Internet Purchase Example

1779 Jill is purchasing a new TV on the Internet through an online sales site:

1780 *Jill* is an instance of **Actor** (person).

1781 *PurchaseTV* is an instance of **Task**.

1782 *Jill* does *PurchaseTV*.

1783 *BuyTVOnline* is an instance of **Service**.

1784 *PurchaseTV* uses *BuyTVOnline*.

1785 *OnlineTVSales* is the company that is selling TVs:

1786 *OnlineTVSales* is an instance of **Actor** (organization).

1787 *BuyTVOnlineContract* is an instance of **ServiceContract** (and describes how to interact with
1788 *BuyTVOnline* as well as the legal contract between TV buyer and *OnlineTVSales*).

1789 *BuyTVOnline* has contract *BuyTVOnlineContract*.

1790 *OnlineTVSales* is party to *BuyTVOnlineContract*.

1791 *Jill* is party to *BuyTVOnlineContract*.

1792 The online site is implemented using web site software:

1793 *OnlineSalesComponent* is an instance of **Element**.

1794 *OnlineSalesComponent* performs *OnlineTVSales*.

1795 *SelectWhatToBuyComponent* is an instance of **Element**.

1796 *SelectWhatToBuyService* is an instance of **Service**.

1797 *SelectWhatToBuyComponent* performs *SelectWhatToBuyService*.

1798 *PayComponent* is an instance of **Element**.

- 1799 *PayService* is an instance of **Service**.
- 1800 *PayComponent* performs *PayService*.
- 1801 *OnlineSalesComponent* is also an instance of **ServiceComposition**.
- 1802 *OnlineSalesComponent* uses *SelectWhatToBuyService* and *PayService*.
- 1803 To complete the purchase transaction, Jill needs to pay for the purchase and then the TV will be delivered:
- 1804 *PayForTV* is an instance of **Task**.
- 1805 *Jill* does *PayForTV*.
- 1806 *PayForTV* uses *BuyTVOnline*.
- 1807 *DeliverTV* is an instance of **Task**.
- 1808 *OnlineTVSales* does *DeliverTV*.
- 1809 *OnlineTVSalesProcess* is an instance of **Process**.
- 1810 *OnlineTVSalesProcess* uses *Jill*, *OnlineTVSales*, *PurchaseTV*, *PayForTV*, and *DeliverTV*.
- 1811
- 1812
- 1813

1814 **Annex A The OWL Definition of the SOA Ontology**

1815 The OWL ontology is available online at:

1816 **13 Editors note: need to find out from JTC1 how / where to post the Ontology RDF file**

1817 14 The Ontology is reproduced below.

```

1818 15 <?xml version="1.0"?>
1819 16
1820 17 <rdf:RDF
1821 18     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
1822 19     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
1823 20     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
1824 21     xmlns:owl="http://www.w3.org/2002/07/owl#"
1825 22     xmlns="http://www.semanticweb.org/ontologies/2010/01/core-soa.owl#"
1826 23     xml:base="http://www.semanticweb.org/ontologies/2010/01/core-soa.owl"
1827 24 >
1828 25
1829 26 <!-- ontology -->
1830 27
1831 28 <owl:Ontology rdf:about=""/>
1832 29
1833 30 <!-- classes -->
1834 31
1835 32 <owl:Class rdf:ID="Event">
1836 33     <owl:disjointWith>
1837 34         <owl:Class rdf:ID="Policy"/>
1838 35     </owl:disjointWith>
1839 36     <owl:disjointWith>
1840 37         <owl:Class rdf:ID="ServiceContract"/>
1841 38     </owl:disjointWith>
1842 39     <owl:disjointWith>
1843 40         <owl:Class rdf:ID="ServiceInterface"/>
1844 41     </owl:disjointWith>
1845 42 </owl:Class>
1846 43
1847 44 <owl:Class rdf:ID="InformationType">
1848 45     <owl:disjointWith>
1849 46         <owl:Class rdf:about="#Policy"/>
1850 47     </owl:disjointWith>
1851 48     <owl:disjointWith>
1852 49         <owl:Class rdf:ID="ServiceContract"/>
1853 50     </owl:disjointWith>
1854 51     <owl:disjointWith>
1855 52         <owl:Class rdf:ID="Effect"/>
1856 53     </owl:disjointWith>
1857 54 </owl:Class>
1858 55

```

```

1859 56 <owl:Class rdf:ID="ServiceComposition">
1860 57   <rdfs:subClassOf>
1861 58     <owl:Class rdf:ID="Composition"/>
1862 59   </rdfs:subClassOf>
1863 60   <owl:disjointWith>
1864 61     <owl:Class rdf:ID="ServiceContract"/>
1865 62   </owl:disjointWith>
1866 63   <owl:disjointWith>
1867 64     <owl:Class rdf:ID="ServiceInterface"/>
1868 65   </owl:disjointWith>
1869 66 </owl:Class>
1870 67
1871 68 <owl:Class rdf:ID="Effect">
1872 69   <owl:disjointWith>
1873 70     <owl:Class rdf:about="#Policy"/>
1874 71   </owl:disjointWith>
1875 72   <owl:disjointWith>
1876 73     <owl:Class rdf:ID="ServiceInterface"/>
1877 74   </owl:disjointWith>
1878 75   <owl:disjointWith>
1879 76     <owl:Class rdf:ID="InformationType"/>
1880 77   </owl:disjointWith>
1881 78   <rdfs:subClassOf>
1882 79     <owl:Restriction>
1883 80       <owl:minCardinality
1884 rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
1885 81         >1</owl:minCardinality>
1886 82       <owl:onProperty>
1887 83         <owl:ObjectProperty rdf:ID="isSpecifiedBy"/>
1888 84       </owl:onProperty>
1889 85     </owl:Restriction>
1890 86   </rdfs:subClassOf>
1891 87 </owl:Class>
1892 88
1893 89 <owl:Class rdf:about="#Task">
1894 90   <owl:disjointWith>
1895 91     <owl:Class rdf:ID="Policy"/>
1896 92   </owl:disjointWith>
1897 93   <owl:disjointWith>
1898 94     <owl:Class rdf:ID="System"/>
1899 95   </owl:disjointWith>
1900 96   <owl:disjointWith>
1901 97     <owl:Class rdf:ID="HumanActor"/>
1902 98   </owl:disjointWith>
1903 99   <owl:disjointWith>
1904 100     <owl:Class rdf:ID="Service"/>
1905 101   </owl:disjointWith>
1906 102   <owl:disjointWith>
1907 103     <owl:Class rdf:ID="ServiceContract"/>
1908 104   </owl:disjointWith>
1909 105   <owl:disjointWith>
1910 106     <owl:Class rdf:ID="ServiceInterface"/>
1911 107   </owl:disjointWith>

```

ISO/IEC WD 1 18384 Part 3 SOA Ontology

```
1912 108 <owl:disjointWith>
1913 109 <owl:Class rdf:ID="Composition"/>
1914 110 </owl:disjointWith>
1915 111 <rdfs:subClassOf>
1916 112 <owl:Class rdf:ID="Element"/>
1917 113 </rdfs:subClassOf>
1918 114 <rdfs:subClassOf>
1919 115 <owl:Restriction>
1920 116 <owl:onProperty>
1921 117 <owl:ObjectProperty rdf:ID="doneBy"/>
1922 118 </owl:onProperty>
1923 119 <owl:minCardinality
1924 rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
1925 120 >0</owl:minCardinality>
1926 121 </owl:Restriction>
1927 122 </rdfs:subClassOf>
1928 123 <rdfs:subClassOf>
1929 124 <owl:Restriction>
1930 125 <owl:maxCardinality
1931 rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
1932 126 >1</owl:maxCardinality>
1933 127 <owl:onProperty>
1934 128 <owl:ObjectProperty rdf:about="#doneBy"/>
1935 129 </owl:onProperty>
1936 130 </owl:Restriction>
1937 131 </rdfs:subClassOf>
1938 132 </owl:Class>
1939 133
1940 134 <owl:Class rdf:about="#System">
1941 135 <owl:disjointWith>
1942 136 <owl:Class rdf:ID="Task"/>
1943 137 </owl:disjointWith>
1944 138 <owl:disjointWith>
1945 139 <owl:Class rdf:ID="Service"/>
1946 140 </owl:disjointWith>
1947 141 <rdfs:subClassOf>
1948 142 <owl:Class rdf:about="#Element"/>
1949 143 </rdfs:subClassOf>
1950 144 </owl:Class>
1951 145
1952 146 <owl:Class rdf:about="#Service">
1953 147 <owl:disjointWith>
1954 148 <owl:Class rdf:ID="System"/>
1955 149 </owl:disjointWith>
1956 150 <owl:disjointWith>
1957 151 <owl:Class rdf:ID="Task"/>
1958 152 </owl:disjointWith>
1959 153 <owl:disjointWith>
1960 154 <owl:Class rdf:ID="HumanActor"/>
1961 155 </owl:disjointWith>
1962 156 <owl:disjointWith>
1963 157 <owl:Class rdf:ID="ServiceInterface"/>
1964 158 </owl:disjointWith>
```

```

1965 159 <rdfs:subClassOf>
1966 160 <owl:Class rdf:about="#Element"/>
1967 161 </rdfs:subClassOf>
1968 162 <rdfs:subClassOf>
1969 163 <owl:Restriction>
1970 164 <owl:minCardinality
1971 rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
1972 165 >1</owl:minCardinality>
1973 166 <owl:onProperty>
1974 167 <owl:ObjectProperty rdf:ID="hasInterface"/>
1975 168 </owl:onProperty>
1976 169 </owl:Restriction>
1977 170 </rdfs:subClassOf>
1978 171 </owl:Class>
1979 172
1980 173 <owl:Class rdf:about="#Policy">
1981 174 <owl:disjointWith>
1982 175 <owl:Class rdf:ID="InformationType"/>
1983 176 </owl:disjointWith>
1984 177 <owl:disjointWith>
1985 178 <owl:Class rdf:ID="ServiceInterface"/>
1986 179 </owl:disjointWith>
1987 180 <owl:disjointWith>
1988 181 <owl:Class rdf:ID="Element"/>
1989 182 </owl:disjointWith>
1990 183 <owl:disjointWith>
1991 184 <owl:Class rdf:ID="Effect"/>
1992 185 </owl:disjointWith>
1993 186 <owl:disjointWith>
1994 187 <owl:Class rdf:ID="Event"/>
1995 188 </owl:disjointWith>
1996 189 <owl:disjointWith>
1997 190 <owl:Class rdf:ID="ServiceContract"/>
1998 191 </owl:disjointWith>
1999 192 </owl:Class>
2000 193
2001 194 <owl:Class rdf:about="#HumanActor">
2002 195 <rdfs:subClassOf>
2003 196 <owl:Class rdf:ID="Element"/>
2004 197 </rdfs:subClassOf>
2005 198 <owl:disjointWith>
2006 199 <owl:Class rdf:ID="Task"/>
2007 200 </owl:disjointWith>
2008 201 <owl:disjointWith>
2009 202 <owl:Class rdf:ID="Service"/>
2010 203 </owl:disjointWith>
2011 204 <owl:disjointWith>
2012 205 <owl:Class rdf:ID="ServiceContract"/>
2013 206 </owl:disjointWith>
2014 207 <owl:disjointWith>
2015 208 <owl:Class rdf:ID="ServiceInterface"/>
2016 209 </owl:disjointWith>
2017 210 </owl:Class>

```



```

2018 211
2019 212 <owl:Class rdf:about="#Composition">
2020 213   <owl:disjointWith>
2021 214     <owl:Class rdf:ID="Task"/>
2022 215   </owl:disjointWith>
2023 216   <rdfs:subClassOf>
2024 217     <owl:Class rdf:ID="System"/>
2025 218   </rdfs:subClassOf>
2026 219   <rdfs:subClassOf>
2027 220     <owl:Restriction>
2028 221       <owl:maxCardinality
2029 222         rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
2030 223         >1</owl:maxCardinality>
2031 224       <owl:onProperty>
2032 225         <owl:DatatypeProperty rdf:ID="compositionPattern"/>
2033 226       </owl:onProperty>
2034 227     </owl:Restriction>
2035 228   </rdfs:subClassOf>
2036 229   <rdfs:subClassOf>
2037 230     <owl:Restriction>
2038 231       <owl:onProperty>
2039 232         <owl:DatatypeProperty rdf:ID="compositionPattern"/>
2040 233       </owl:onProperty>
2041 234       <owl:minCardinality
2042 235         rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
2043 236         >1</owl:minCardinality>
2044 237     </owl:Restriction>
2045 238   </rdfs:subClassOf>
2046 239   <rdfs:subClassOf>
2047 240     <owl:Restriction>
2048 241       <owl:maxCardinality
2049 242         rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
2050 243         >1</owl:maxCardinality>
2051 244       <owl:onProperty>
2052 245         <owl:ObjectProperty rdf:ID="orchestratedBy"/>
2053 246       </owl:onProperty>
2054 247     </owl:Restriction>
2055 248   </rdfs:subClassOf>
2056 249   <rdfs:subClassOf>
2057 250     <owl:Restriction>
2058 251       <owl:minCardinality
2059 252         rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
2060 253         >0</owl:minCardinality>
2061 254       <owl:onProperty>
2062 255         <owl:ObjectProperty rdf:ID="orchestratedBy"/>
2063 256       </owl:onProperty>
2064 257     </owl:Restriction>
2065 258   </rdfs:subClassOf>
2066 259 </owl:Class>
2067
2068 257 <owl:Class rdf:about="#ServiceInterface">
2069 258   <owl:disjointWith>
2070 259     <owl:Class rdf:ID="Service"/>

```

```

2071 260 </owl:disjointWith>
2072 261 <owl:disjointWith>
2073 262 <owl:Class rdf:ID="ServiceContract"/>
2074 263 </owl:disjointWith>
2075 264 <owl:disjointWith>
2076 265 <owl:Class rdf:ID="Effect"/>
2077 266 </owl:disjointWith>
2078 267 <owl:disjointWith>
2079 268 <owl:Class rdf:ID="Policy"/>
2080 269 </owl:disjointWith>
2081 270 <owl:disjointWith>
2082 271 <owl:Class rdf:ID="HumanActor"/>
2083 272 </owl:disjointWith>
2084 273 <owl:disjointWith>
2085 274 <owl:Class rdf:ID="Task"/>
2086 275 </owl:disjointWith>
2087 276 <owl:disjointWith>
2088 277 <owl:Class rdf:ID="ServiceComposition"/>
2089 278 </owl:disjointWith>
2090 279 <owl:disjointWith>
2091 280 <owl:Class rdf:ID="Process"/>
2092 281 </owl:disjointWith>
2093 282 <owl:disjointWith>
2094 283 <owl:Class rdf:ID="Event"/>
2095 284 </owl:disjointWith>
2096 285 <rdfs:subClassOf>
2097 286 <owl:Restriction>
2098 287 <owl:onProperty>
2099 288 <owl:DatatypeProperty rdf:ID="constraints"/>
2100 289 </owl:onProperty>
2101 290 <owl:maxCardinality
2102 rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
2103 291 >1</owl:maxCardinality>
2104 292 </owl:Restriction>
2105 293 </rdfs:subClassOf>
2106 294 <rdfs:subClassOf>
2107 295 <owl:Restriction>
2108 296 <owl:minCardinality
2109 rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
2110 297 >1</owl:minCardinality>
2111 298 <owl:onProperty>
2112 299 <owl:DatatypeProperty rdf:about="#constraints"/>
2113 300 </owl:onProperty>
2114 301 </owl:Restriction>
2115 302 </rdfs:subClassOf>
2116 303 </owl:Class>
2117 304
2118 305 <owl:Class rdf:about="#Element">
2119 306 <owl:disjointWith>
2120 307 <owl:Class rdf:ID="Policy"/>
2121 308 </owl:disjointWith>
2122 309 <rdfs:subClassOf>
2123 310 <owl:Restriction>

```

```

2124      311      <owl:minCardinality
2125 rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
2126      312      >0</owl:minCardinality>
2127      313      <owl:onProperty>
2128      314      <owl:ObjectProperty rdf:ID="orchestrates"/>
2129      315      </owl:onProperty>
2130      316      </owl:Restriction>
2131      317      </rdfs:subClassOf>
2132      318      <rdfs:subClassOf>
2133      319      <owl:Restriction>
2134      320      <owl:maxCardinality
2135 rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
2136      321      >1</owl:maxCardinality>
2137      322      <owl:onProperty>
2138      323      <owl:ObjectProperty rdf:about="#orchestrates"/>
2139      324      </owl:onProperty>
2140      325      </owl:Restriction>
2141      326      </rdfs:subClassOf>
2142      327      </owl:Class>
2143      328
2144      329      <owl:Class rdf:about="#ServiceContract">
2145      330      <owl:disjointWith>
2146      331      <owl:Class rdf:ID="ServiceInterface"/>
2147      332      </owl:disjointWith>
2148      333      <owl:disjointWith>
2149      334      <owl:Class rdf:ID="Policy"/>
2150      335      </owl:disjointWith>
2151      336      <owl:disjointWith>
2152      337      <owl:Class rdf:ID="HumanActor"/>
2153      338      </owl:disjointWith>
2154      339      <owl:disjointWith>
2155      340      <owl:Class rdf:ID="Task"/>
2156      341      </owl:disjointWith>
2157      342      <owl:disjointWith>
2158      343      <owl:Class rdf:ID="ServiceComposition"/>
2159      344      </owl:disjointWith>
2160      345      <owl:disjointWith>
2161      346      <owl:Class rdf:ID="Process"/>
2162      347      </owl:disjointWith>
2163      348      <owl:disjointWith>
2164      349      <owl:Class rdf:ID="Event"/>
2165      350      </owl:disjointWith>
2166      351      <owl:disjointWith>
2167      352      <owl:Class rdf:ID="InformationType"/>
2168      353      </owl:disjointWith>
2169      354      <rdfs:subClassOf>
2170      355      <owl:Restriction>
2171      356      <owl:onProperty>
2172      357      <owl:DatatypeProperty rdf:ID="legalAspect"/>
2173      358      </owl:onProperty>
2174      359      <owl:minCardinality
2175 rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
2176      360      >1</owl:minCardinality>

```

```

2177 361     </owl:Restriction>
2178 362 </rdfs:subClassOf>
2179 363 <rdfs:subClassOf>
2180 364     <owl:Restriction>
2181 365         <owl:maxCardinality
2182 rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
2183 366         >1</owl:maxCardinality>
2184 367         <owl:onProperty>
2185 368             <owl:DatatypeProperty rdf:ID="legalAspect"/>
2186 369         </owl:onProperty>
2187 370     </owl:Restriction>
2188 371 </rdfs:subClassOf>
2189 372 <rdfs:subClassOf>
2190 373     <owl:Restriction>
2191 374         <owl:onProperty>
2192 375             <owl:DatatypeProperty rdf:ID="interactionAspect"/>
2193 376         </owl:onProperty>
2194 377         <owl:maxCardinality
2195 rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
2196 378         >1</owl:maxCardinality>
2197 379     </owl:Restriction>
2198 380 </rdfs:subClassOf>
2199 381 <rdfs:subClassOf>
2200 382     <owl:Restriction>
2201 383         <owl:onProperty>
2202 384             <owl:DatatypeProperty rdf:about="#interactionAspect"/>
2203 385         </owl:onProperty>
2204 386         <owl:minCardinality
2205 rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
2206 387         >1</owl:minCardinality>
2207 388     </owl:Restriction>
2208 389 </rdfs:subClassOf>
2209 390 <rdfs:subClassOf>
2210 391     <owl:Restriction>
2211 392         <owl:onProperty>
2212 393             <owl:ObjectProperty rdf:ID="isContractFor"/>
2213 394         </owl:onProperty>
2214 395         <owl:minCardinality
2215 rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
2216 396         >1</owl:minCardinality>
2217 397     </owl:Restriction>
2218 398 </rdfs:subClassOf>
2219 399 <rdfs:subClassOf>
2220 400     <owl:Restriction>
2221 401         <owl:onProperty>
2222 402             <owl:ObjectProperty rdf:ID="specifies"/>
2223 403         </owl:onProperty>
2224 404         <owl:minCardinality
2225 rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
2226 405         >1</owl:minCardinality>
2227 406     </owl:Restriction>
2228 407 </rdfs:subClassOf>
2229 408 </owl:Class>

```

```

2230      409
2231      410 <owl:Class rdf:about="#Process">
2232      411   <owl:disjointWith>
2233      412     <owl:Class rdf:ID="ServiceContract"/>
2234      413   </owl:disjointWith>
2235      414   <owl:disjointWith>
2236      415     <owl:Class rdf:ID="ServiceInterface"/>
2237      416   </owl:disjointWith>
2238      417   <rdfs:subClassOf>
2239      418     <owl:Class rdf:ID="Composition"/>
2240      419   </rdfs:subClassOf>
2241      420 </owl:Class>
2242      421
2243      422 <!-- object properties -->
2244      423
2245      424 <owl:ObjectProperty rdf:about="#isPartyTo">
2246      425   <rdfs:domain rdf:resource="#HumanActor"/>
2247      426   <rdfs:range rdf:resource="#ServiceContract"/>
2248      427 </owl:ObjectProperty>
2249      428
2250      429 <owl:ObjectProperty rdf:ID="involvesParty">
2251      430   <owl:inverseOf>
2252      431     <owl:ObjectProperty rdf:ID="isPartyTo"/>
2253      432   </owl:inverseOf>
2254      433 </owl:ObjectProperty>
2255      434
2256      435 <owl:ObjectProperty rdf:about="#orchestratedBy">
2257      436   <rdfs:domain rdf:resource="#Composition"/>
2258      437   <rdfs:range rdf:resource="#Element"/>
2259      438 </owl:ObjectProperty>
2260      439
2261      440 <owl:ObjectProperty rdf:about="#orchestrates">
2262      441   <owl:inverseOf>
2263      442     <owl:ObjectProperty rdf:ID="orchestratedBy"/>
2264      443   </owl:inverseOf>
2265      444 </owl:ObjectProperty>
2266      445
2267      446 <owl:ObjectProperty rdf:about="#isContractFor">
2268      447   <rdfs:domain rdf:resource="#ServiceContract"/>
2269      448   <rdfs:range rdf:resource="#Service"/>
2270      449 </owl:ObjectProperty>
2271      450
2272      451 <owl:ObjectProperty rdf:ID="hasContract">
2273      452   <owl:inverseOf>
2274      453     <owl:ObjectProperty rdf:about="#isContractFor"/>
2275      454   </owl:inverseOf>
2276      455 </owl:ObjectProperty>
2277      456
2278      457 <owl:ObjectProperty rdf:about="#setsPolicy">
2279      458   <rdfs:domain rdf:resource="#HumanActor"/>
2280      459   <rdfs:range rdf:resource="#Policy"/>
2281      460 </owl:ObjectProperty>
2282      461

```

```

2283 462 <owl:ObjectProperty rdf:ID="isSetBy">
2284 463   <owl:inverseOf>
2285 464     <owl:ObjectProperty rdf:ID="setsPolicy"/>
2286 465   </owl:inverseOf>
2287 466 </owl:ObjectProperty>
2288 467
2289 468 <owl:ObjectProperty rdf:ID="generates">
2290 469   <rdfs:domain rdf:resource="#Element"/>
2291 470   <rdfs:range rdf:resource="#Event"/>
2292 471 </owl:ObjectProperty>
2293 472
2294 473 <owl:ObjectProperty rdf:ID="generatedBy">
2295 474   <owl:inverseOf>
2296 475     <owl:ObjectProperty rdf:ID="generates"/>
2297 476   </owl:inverseOf>
2298 477 </owl:ObjectProperty>
2299 478
2300 479 <owl:ObjectProperty rdf:about="#represents">
2301 480   <rdfs:domain rdf:resource="#Element"/>
2302 481   <rdfs:range rdf:resource="#Element"/>
2303 482 </owl:ObjectProperty>
2304 483
2305 484 <owl:ObjectProperty rdf:ID="representedBy">
2306 485   <owl:inverseOf>
2307 486     <owl:ObjectProperty rdf:ID="represents"/>
2308 487   </owl:inverseOf>
2309 488 </owl:ObjectProperty>
2310 489
2311 490 <owl:ObjectProperty rdf:ID="hasInput">
2312 491   <rdfs:domain rdf:resource="#ServiceInterface"/>
2313 492   <rdfs:range rdf:resource="#InformationType"/>
2314 493 </owl:ObjectProperty>
2315 494
2316 495 <owl:ObjectProperty rdf:ID="isInputAt">
2317 496   <owl:inverseOf>
2318 497     <owl:ObjectProperty rdf:ID="hasInput"/>
2319 498   </owl:inverseOf>
2320 499 </owl:ObjectProperty>
2321 500
2322 501 <owl:ObjectProperty rdf:about="#doneBy">
2323 502   <rdfs:domain rdf:resource="#Task"/>
2324 503   <rdfs:range rdf:resource="#HumanActor"/>
2325 504 </owl:ObjectProperty>
2326 505
2327 506 <owl:ObjectProperty rdf:ID="does">
2328 507   <owl:inverseOf>
2329 508     <owl:ObjectProperty rdf:about="#doneBy"/>
2330 509   </owl:inverseOf>
2331 510 </owl:ObjectProperty>
2332 511
2333 512 <owl:ObjectProperty rdf:about="#specifies">
2334 513   <rdfs:domain rdf:resource="#ServiceContract"/>
2335 514   <rdfs:range rdf:resource="#Effect"/>

```

```

2336     515 </owl:ObjectProperty>
2337     516
2338     517 <owl:ObjectProperty rdf:about="#isSpecifiedBy">
2339     518     <owl:inverseOf>
2340     519         <owl:ObjectProperty rdf:about="#specifies"/>
2341     520     </owl:inverseOf>
2342     521 </owl:ObjectProperty>
2343     522
2344     523 <owl:ObjectProperty rdf:ID="appliesTo">
2345     524     <rdfs:domain rdf:resource="#Policy"/>
2346     525 </owl:ObjectProperty>
2347     526
2348     527 <owl:ObjectProperty rdf:ID="isSubjectTo">
2349     528     <owl:inverseOf>
2350     529         <owl:ObjectProperty rdf:ID="appliesTo"/>
2351     530     </owl:inverseOf>
2352     531 </owl:ObjectProperty>
2353     532
2354     533 <owl:ObjectProperty rdf:about="#hasInterface">
2355     534     <rdfs:domain rdf:resource="#Service"/>
2356     535     <rdfs:range rdf:resource="#ServiceInterface"/>
2357     536 </owl:ObjectProperty>
2358     537
2359     538 <owl:ObjectProperty rdf:ID="isInterfaceOf">
2360     539     <owl:inverseOf>
2361     540         <owl:ObjectProperty rdf:about="#hasInterface"/>
2362     541     </owl:inverseOf>
2363     542 </owl:ObjectProperty>
2364     543
2365     544 <owl:ObjectProperty rdf:ID="respondsTo">
2366     545     <rdfs:domain rdf:resource="#Element"/>
2367     546     <rdfs:range rdf:resource="#Event"/>
2368     547 </owl:ObjectProperty>
2369     548
2370     549 <owl:ObjectProperty rdf:ID="respondedBy">
2371     550     <owl:inverseOf>
2372     551         <owl:ObjectProperty rdf:ID="respondsTo"/>
2373     552     </owl:inverseOf>
2374     553 </owl:ObjectProperty>
2375     554
2376     555 <owl:ObjectProperty rdf:ID="performs">
2377     556     <rdfs:domain rdf:resource="#Element"/>
2378     557     <rdfs:range rdf:resource="#Service"/>
2379     558 </owl:ObjectProperty>
2380     559
2381     560 <owl:ObjectProperty rdf:ID="performedBy">
2382     561     <owl:inverseOf>
2383     562         <owl:ObjectProperty rdf:ID="performs"/>
2384     563     </owl:inverseOf>
2385     564 </owl:ObjectProperty>
2386     565
2387     566 <owl:ObjectProperty rdf:about="#uses">
2388     567     <rdfs:domain rdf:resource="#Element"/>

```

```

2389     568     <rdfs:range rdf:resource="#Element"/>
2390     569 </owl:ObjectProperty>
2391     570
2392     571 <owl:ObjectProperty rdf:ID="usedBy">
2393     572     <owl:inverseOf>
2394     573         <owl:ObjectProperty rdf:ID="uses"/>
2395     574     </owl:inverseOf>
2396     575 </owl:ObjectProperty>
2397     576
2398     577 <owl:ObjectProperty rdf:ID="hasOutput">
2399     578     <rdfs:domain rdf:resource="#ServiceInterface"/>
2400     579     <rdfs:range rdf:resource="#InformationType"/>
2401     580 </owl:ObjectProperty>
2402     581
2403     582 <owl:ObjectProperty rdf:ID="isOutputAt">
2404     583     <owl:inverseOf>
2405     584         <owl:ObjectProperty rdf:ID="hasOutput"/>
2406     585     </owl:inverseOf>
2407     586 </owl:ObjectProperty>
2408     587
2409     588 <!-- datatype properties -->
2410     589
2411     590 <owl:DatatypeProperty rdf:about="#legalAspect">
2412     591     <rdfs:domain rdf:resource="#ServiceContract"/>
2413     592 </owl:DatatypeProperty>
2414     593
2415     594 <owl:DatatypeProperty rdf:about="#constraints">
2416     595     <rdfs:domain rdf:resource="#ServiceInterface"/>
2417     596 </owl:DatatypeProperty>
2418     597
2419     598 <owl:DatatypeProperty rdf:about="#compositionPattern">
2420     599     <rdfs:domain rdf:resource="#Composition"/>
2421     600 </owl:DatatypeProperty>
2422     601
2423     602 <owl:DatatypeProperty rdf:about="#interactionAspect">
2424     603     <rdfs:domain rdf:resource="#ServiceContract"/>
2425     604 </owl:DatatypeProperty>
2426     605
2427     606 </rdf:RDF>
2428
2429

```


2430

2431 Annex B (Informative) Class Relationship Matrix

2432 This appendix contains a class relationship matrix that illustrates the class-to-class relationships intrinsic in
2433 the OWL definitions of the SOA ontology. The matrix is deterministically derived from the ontology OWL
2434 definitions. Each row X and each column Y corresponds to an OWL class. A relation appears in cell (X,Y) if
2435 and only if class X is part of the domain and class Y is part of the range of the corresponding OWL property.
2436 Note that this means that datatype properties (which do not have a range) are not included in the class
2437 relationship matrix.

2438 As outlined in the body of the document there are four relationships in the table (plus their inverses and sub-
2439 classed derivatives) that are technically allowed according to the OWL definitions, but would not be expected
2440 to occur in a practical application of the ontology. Specifically, services are not expected to perform services,
2441 services are not expected to use elements (directly), services are not expected to represent elements, and
2442 services are not expected to orchestrate compositions – all due to the **Service** class being defined as a logical
2443 representation of a repeatable activity; see [The performs and performedBy Properties \(Clause 7.3\)](#), [The uses
2444 and usedBy Properties Applied to Service \(Clause 7.4.1\)](#), [The represents and representedBy Properties
2445 Applied to Service \(Clause 7.4.2\)](#) and [The orchestrates and orchestratedBy Properties \(Clause 8.3\)](#) for details.

ISO/IEC WD 1 18384 Part 3 SOA Ontology

	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 performedBy	Uses usedBy represents representedBy performedBy	uses usedBy represents representedBy performedBy	uses usedBy represents representedBy orchestrates	uses usedBy represents representedBy orchestrates	uses usedBy represents representedBy 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	
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	appliesTo
Thing														isSubjectTo	

Annex C (Informative) Issues List

The following issues remain to be addressed:

Comment Ref	Comment summary	Action/Disposition

:

Annex D (Informative) Bibliography

Editors Note: The bibliography needs reducing, ensure all references are actually used

1. ISO/IEC Directives, Part 2, *Rules for the structure and drafting of International Standards*, 2001
2. ISO/IEC TR 10000-1, *Information technology — Framework and taxonomy of International Standardized Profiles — Part 1: General principles and documentation framework*
3. ISO 10241, *International terminology standards — Preparation and layout*
4. ISO 128-30, *Technical drawings — General principles of presentation — Part 30: Basic conventions for views*
 1. ISO 690, *Documentation — Bibliographic references — Content, form and structure*
 2. ISO 690-2, *Information and documentation — Bibliographic references — Part 2: Electronic documents or parts thereof*
3. ISO/IEC JTC 1/SC 38 N0043, *Research Report on China's SOA Standards System*
4. ISO/IEC JTC 1/SC 38 N0022, *Chinese National Body Contribution on Proposed NP for General Technical Requirement of Service Oriented Architecture*
5. 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>>
6. The Open Group, *Open Group Standard SOA Reference Architecture Technical Standard*, Available from World Wide Web: < http://www.opengroup.org/soa/source-book/soa_refarch/index.htm>, pdf: <https://www2.opengroup.org/ogsys/jsp/publications/PublicationDetails.jsp?publicationid=12490>
7. The Open Group, *Technical Standard Service-Oriented Architecture Ontology* Available from World Wide Web: <http://www.opengroup.org/soa/source-book/ontology/index.htm>, pdf format available: <https://www2.opengroup.org/ogsys/jsp/publications/PublicationDetails.jsp?publicationid=12245>
8. Open Group Technical Standard, *SOA Governance Framework*, Available from World Wide Web: <http://www.opengroup.org/soa/source-book/gov/intro.htm>, pdf format available: <https://www2.opengroup.org/ogsys/jsp/publications/PublicationDetails.jsp?publicationid=12205>
9. OMG *Business Process Management Notation (BPMN)*, see <http://www.omg.org/spec/BPMN/2.0/>
10. ISO Technical Report TR9007, *Concepts and Terminology for the Conceptual Schema and the Information Base*
11. *The Open Group Architecture Framework (TOGAF)*, section 8.1.1 Version 9 Enterprise Edition, February 2009; see www.opengroup.org/togaf
12. 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
13. W3C *Web Services Description Language (WSDL) 1.1*, W3C Note 15 March 2001, see <http://www.w3.org/TR/wSDL>
14. OASIS *Web Services for Remote Portlets Specification v2.0* OASIS Standard, 1 April 2008 (WSRP), see <http://docs.oasis-open.org/wsrp/v2/wsrp-2.0-spec.html>

ISO/IEC WD 1 18384 Part 3 SOA Ontology

15. 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
16. 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
17. 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
18. W3C *Web Ontology Language (OWL)*, World Wide Web Consortium (W3C), April 2009: see www.w3.org/2007/OWL/wiki/OWL_Working_Group
19. Garrett, Jesse James, *A New Approach to Web Applications*, Feb 18, 2005 see <http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications>
20. OASIS Web Services Coordination (WS-Coordination) Version 1.2, OASIS Standard, Feb 2, 2009, <http://docs.oasis-open.org/ws-tx/wstx-wscoor-1.2-spec-os.pdf>
21. Web Services Atomic Transaction (WS-Atomic Transaction) Versions 1.2 OASIS Standard, Feb 2, 2009, <http://docs.oasis-open.org/ws-tx/wstx-wsat-1.2-spec-os.pdf>
22. Web Services Business Activity (WS-Business Activity) Version 1.2 OASIS Standard, Feb 2, 2009, <http://docs.oasis-open.org/ws-tx/wstx-wsba-1.2-spec-os.pdf>
23. Business Process Modeling Notation (BPMN), Version 1.1, Object Management Group; available from www.omg.org.
24. Beyond Concepts: Ontology as Reality Representation, by Barry Smith; available from <http://ontology.buffalo.edu/bfo/BeyondConcepts.pdf>.
25. Definition of SOA: The Open Group; available from www.opengroup.org/soa/soa/def.htm#_Definition_of_SOA.
26. IEEE Std 1471-2000: IEEE Recommended Practice for Architectural Description of Software-intensive Systems (adopted by ISO/IEC JTC1/SC7 as ISO/IEC 42010:2007); available from standards.ieee.org.
27. IETF RFC 2119: Key Words for use in RFCs to Indicate Requirement Levels, March 1997; refer to www.ietf.org.
28. ISO/IEC 42010:2007: Systems and Software Engineering – Recommended Practice for Architectural Description of Software-intensive Systems; available from www.iso.org.
29. Navigating the SOA Open Standards Landscape Around Architecture (W096), White Paper published by The Open Group, November 2009.
30. OASIS Reference Model for Service-Oriented Architecture, Version 1.0, Organization for the Advancement of Structured Information Standards (OASIS); available from www.oasis-open.org.
31. OWL Web Ontology Language Reference, W3C Recommendation, 10 February 2004, World-Wide Web Consortium; available from www.w3.org/TR/owl-ref.

32. Service-Oriented Architecture Modeling Language (SoaML), Object Management Group; available from www.omg.org.
33. The Open Group Architecture Framework (TOGAF), The Open Group; available from www.opengroup.org.
34. What is an Ontology? Stanford University; available from www-ksl.stanford.edu/kst/what-is-an-ontology.html.