Second version of the ASSERT
Ontology

Deliverable D.3.4

Rev. 1.0
**Project Title**
ASSERT4SoA – Advanced Security Service cERTificate for SOA

**Grant Number**
257351

**Project Type**
CP - STREP

<table>
<thead>
<tr>
<th><strong>Title of Deliverable</strong></th>
<th>Second version of the ASSERT Ontology</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Subtitle of Deliverable</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Deliverable Number</strong></td>
<td>3.4</td>
</tr>
<tr>
<td><strong>Dissemination Level</strong></td>
<td>PU</td>
</tr>
<tr>
<td><strong>Internal Rev. No.</strong></td>
<td>1.0</td>
</tr>
<tr>
<td><strong>Contractual Delivery Date</strong></td>
<td>31 May 2012</td>
</tr>
<tr>
<td><strong>Actual Delivery Date</strong></td>
<td>31 May 2013</td>
</tr>
<tr>
<td><strong>Contributing WPs</strong></td>
<td>WP3</td>
</tr>
<tr>
<td><strong>Editor(s)</strong></td>
<td>Valentina Di Giacomo</td>
</tr>
<tr>
<td><strong>Author(s)</strong></td>
<td>Valentina Di Giacomo, Claudia Pandolfo, Domenico Presenza, Stefania D'Agostini</td>
</tr>
<tr>
<td><strong>Reviewer(s)</strong></td>
<td>Henk Birkholz, Andreas Fuchs, Sigi Gürgens (SIT), Hristo Koshutanski (UMA)</td>
</tr>
</tbody>
</table>
Executive Summary

The aim of this document is to provide a comprehensive description of the second version of the ASSERT4SOA Ontology, aimed at supporting the description of certificates and security requirements and enabling the interoperability and comparison of heterogeneous certificates in the ASSERT4SOA Framework. The ASSERT4SOA Ontology models concepts related to security, such as security requirements and verification techniques, and is built by extending existing ontologies. The identified top concepts are used to model certification specific concepts, including the model of the three types of ASSERTs defined in this project (ASSERT-M, ASSERT-E and ASSERT-O).

To achieve this objective the present deliverable contains some material already presented in the deliverable D3.2 – “First Version of the ASSERT Ontology v.1.0”. More specifically:

- Chapters 1 – “Introduction”, 2 – “The ASSERT4SOA Top Ontology” and 3 – “ASSERT-O Models” remained unchanged;
- Chapters 4 – “ASSERT-E” and 5 – “ASSERT-M” has been largely rewritten to provide a more precise description of how the ASSERT4SOA ontology can be used to model services, security properties and certificates of the ASSERT-M and ASSERT-E defined in this project;
- Chapter 6 – “Semantic Similarity” contains completely new material presenting the semantic similarity function proposed to express the similarity of two ASSERT-O;
- The description of the modules of the ontology has been moved into the Appendix A.
Contents

1. Introduction ............................................................................................................. 11
   1.1 Source distribution and requirements............................................................... 12
   1.2 Overview ........................................................................................................... 12
   1.3 Development Methodology .............................................................................. 13
      1.3.1 Integrating concepts from different domains.......................................... 14
      1.3.2 Identifying concepts ................................................................................. 14
   1.4 Modelling Services and Security Properties ................................................... 15
2. The ASSERT4SOA Top Ontology ............................................................................ 17
   2.1 Ontology Design Patterns ................................................................................ 17
      2.1.1 Ordering .................................................................................................. 17
      2.1.2 Set ........................................................................................................... 19
      2.1.3 Statement ............................................................................................... 19
      2.1.4 Role ......................................................................................................... 19
      2.1.5 Dependency ............................................................................................. 20
   2.2 Modelling Examples .......................................................................................... 21
      2.2.1 Mathematical Structures ........................................................................... 21
      2.2.2 Agents ..................................................................................................... 21
      2.2.3 Mental States ........................................................................................... 22
      2.2.4 Activities ................................................................................................. 23
      2.2.5 Organisational Structures ........................................................................ 27
      2.2.6 Requirements ........................................................................................... 28
3. ASSERT-O Models .................................................................................................. 33
   3.1 Ontology-based properties ................................................................................ 33
   3.2 Ontology-based certificates (ASSERT-O) ......................................................... 36
   3.3 ASSERT-O service model .................................................................................. 41
4. ASSERT-E .................................................................................................................. 45
   4.1 Representing ASSERT-E Service Models in OWL .......................................... 45
      4.1.1 Symbolic Transition System ..................................................................... 45
      4.1.2 Service Model .......................................................................................... 47
   4.2 Representing Container-level Security Properties ....................................... 48
      4.2.1 Confidentiality ......................................................................................... 48
ASSERT4SoA

4.2.2 Integrity .......................................................... 49

4.3 Certificates Model .................................................. 50

5. ASSERT-M ............................................................... 55

5.1 Representing SeMF in OWL ........................................ 55

5.1.1 Agents ......................................................... 55

5.1.2 Actions .......................................................... 56

5.1.3 Traces ............................................................ 56

5.1.4 Alphabetic language homomorphism ......................... 58

5.1.5 Agents' Knowledge ....................................... 60

5.1.6 Agents' Local View ...................................... 61

5.1.7 System ...................................................... 62

5.2 Representing Basic Properties Definitions ..................... 65

5.2.1 Authenticity ............................................... 66

5.2.2 Authenticity with Respect to a Phase ....................... 67

5.2.3 Confidentiality ............................................ 70

5.2.4 Enforcing system behavior .................................. 72

6. Semantic Similarity .................................................. 75

6.1 Background ............................................................ 75

6.2 Adopted approach ............................................... 77

6.3 Comparing ASSERT-Os ........................................... 78

6.4 Considerations on the chosen approach ......................... 82

7. Conclusions .......................................................... 83

7.1 Conclusions .......................................................... 83

8. Appendix A. Modules ................................................ 85

8.1 Modules and dependencies ..................................... 85

A.1.1. Statement .................................................. 86

A.1.2. Mereology ............................................... 87

A.1.3. Dependency ............................................ 88

A.1.4. Time ....................................................... 90

A.1.5. Primitive-types ....................................... 97

A.1.6. Roles ...................................................... 98

A.1.7. Math ......................................................... 99

A.1.8. Math-calculus ......................................... 106

A.1.9. Math-algebra ........................................ 107

A.1.10. Math-discrete ....................................... 109

A.1.11. Agents .................................................. 112

A.1.12. WSDL .................................................. 117

A.1.13. Measurements .......................................... 120

A.1.14. Mental-states ....................................... 122

A.1.15. Communication ...................................... 126

A.1.16. Organisations ....................................... 130

A.1.17. Activity ................................................. 132
### ASSERT4SoA

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.1.18. Requirements</td>
<td>134</td>
</tr>
<tr>
<td>A.1.19. A4S Language</td>
<td>136</td>
</tr>
<tr>
<td>A.1.20. USDL-SEC</td>
<td>138</td>
</tr>
<tr>
<td>A.1.21. ASSERT-E</td>
<td>143</td>
</tr>
<tr>
<td>A.1.22. ASSERT-O</td>
<td>151</td>
</tr>
<tr>
<td>A.1.23. SeMF</td>
<td>154</td>
</tr>
<tr>
<td>A.1.24. ASSERT-M</td>
<td>155</td>
</tr>
<tr>
<td>A.1.25. Topology</td>
<td>159</td>
</tr>
<tr>
<td>A.1.26. WSA</td>
<td>160</td>
</tr>
<tr>
<td>A.1.27. WSS</td>
<td>162</td>
</tr>
<tr>
<td>9. Bibliography</td>
<td>164</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

This document introduces the ASSERT4SOA Ontology, aimed at supporting the description of certificates and security requirements and enabling the interoperability and comparison of heterogeneous certificates in the ASSERT4SOA Framework [1].

In this document, the term ontology is intended as “a formal explicit specification of a shared conceptualization of a domain” [2].

The ASSERT4SOA Ontology models concepts related to security, such as security requirements and verification techniques, and is built by extending existing ontologies. The ASSERT4SOA Ontology can be exploited by the message router of an Enterprise Service Bus [3], so that it is capable to bind the service endpoint of requests from a Service Consumer, with running services satisfying the desired security requirements. The types of requirements that are addressed in this work are security-related non-functional requirements, such as authenticity, confidentiality, or robustness.

The rest of this deliverable is organized as follows. Chapter 1 gives an overview of the approach followed and the structure of the ASSERT4SOA Ontology, and provides an overview of the design principles, content, objectives and methodology of the ASSERT4SOA Ontology. Chapter 2 describes the characteristics of the main concepts of the ASSERT4SOA Top Ontology. Chapter 3, Chapter 4 and Chapter 5 show how the semantic security certificates (respectively ASSERT-O, ASSERT-E and ASSERT-M) defined by the project are modelled using the ASSERT4SOA Top Ontology. Chapter 6 investigates the Semantic Similarity between OWL concepts and how this similarity can be used to compare ASSERTs. Chapter 7 summarizes the work done. Finally, the Appendix describes in details the modules of the ontology, reporting the classes and object properties for each of them.
1.1 Source distribution and requirements

The ontology described in this document has been developed using the following software:

- Protégé ontology editor (version 4.1.0 build 2.3.9);  
- Pellet OWL 2 reasoner, Pellet reasoner plug-in for Protégé v.2.1.2;  
- Java 1.6 runtime environment.

A binary distribution of the ontology described in this document can be obtained by sending a request to the main authors of this deliverable.

1.2 Overview

In the domain of SOA Services security certification, there are multiple certification schemes (e.g. Common Criteria, CCT Mark and others), thus the interactions between the main actors (Certification Authorities, Service Providers, Evaluation Bodies and Service Consumers) are complicated by the use of a heterogeneous vocabulary to define requirements and to model services. The comparison of certificates is one of the open problems addressed by the ASSERT4SOA Ontology.

Therefore, the objective of the ASSERT4SOA Ontology is twofold [4]:

- To provide general concepts/terms useful for the definition of security requirements (e.g. Confidentiality, Integrity, Authenticity, etc.);
- To support interoperability and comparison of different kinds of certificates;

The ASSERT4SOA Ontology aims at describing the domain of SOA services security certification [5] to support the interaction between Certification Authorities, Service Providers, Evaluation Bodies and Service Consumers. This ontology is formalized using the OWL 2 DL Web Ontology Language [6] [7] which is an extension of the description logic SROIQ [7]. OWL 2 DL has been chosen because a set of inference problems is decidable in it. This set includes Ontology Consistency and Instance Checking (i.e. checking whether an individual is an instance of a class). This work proposes the use of OWL 2 DL ontologies and reasoners (e.g. Pellet) to back on decidable inference of this type of formalism, some of the decision problems arising in the context of run-time selection and binding of certified SOA services.

In the proposed approach each Security Certificate, Security Requirement and Service Model corresponds to an OWL document deployed in the Ontology Manager of the ASSERT4SOA run-time environment (see [1]). This set of documents altogether forms the ASSERT4SOA Ontology.

The ASSERT4SOA Ontology has three layers (Figure 1):

- **Top Ontology Layer**: containing a set of ontologies defining general Terms/Concepts (e.g. Sequence, Function, Activity, Agent)
- **Certificates Models Layer**: including OWL documents defining concepts specific to a certification scheme (e.g. the concept of Test Unit used to describe test-

---

1 http://protege.stanford.edu/  
2 http://clarkparsia.com/pellet/
based certificates). In Figure 1, the ASSERT-O, ASSERT-M and ASSERT-E boxes represent ontologies modelling the three types of certificates defined by the ASSERT4SOA project. However, it is possible to have additional ontologies for other certification schemes (e.g. Common Criteria) in this layer.

- **Certificates Instances Layer**: including OWL documents describing single certificates of specific services (e.g. the CC certificate of a market investment service).

With respect to the first version of the ASSERT Ontology, in this second version:

- ASSERT-E and ASSERT-M modules have been revised to provide a more precise representation of services, security properties and certificates according to the ASSERT-E and ASSERT-M models
- New modules have been developed to consider security standards (USDL-SEC\(^3\), WS-Architecture [8] and WS-Security [9]), Topological Spaces (Topology) and ASSERT4SOA related aspects (A4S Language [10] and SeMF [11]).

All ontology modules are described in details in Appendix A.

### 1.3 Development Methodology

Our ontology has been designed in accordance with the following objectives and design principles [12]:

- to provide a vocabulary to describe SOA services as OWL individuals and security requirements by means of OWL classes. This choice allows to reformulate the problem of verification that a SOA service model fulfils a security requirement as a Description Logic (DL) instance checking inference. The entailment between security requirements can be decided using the DL class expression subsumption inference.

\(^3\) [http://www.linked-usdl.org/usdl-sec](http://www.linked-usdl.org/usdl-sec)
Assert4Soa

- to support the description of a SOA service as a multi-agent system [13] [14]. Agents are entities capable to sense and act within their environment, including (passive) objects and other agents in the environment as well.
- to provide a vocabulary aimed at representing the content of a security certificate document rather than its structure in terms of sections, subsections, paragraphs and the like. In this way, these documents can be regarded as "semantic security certificates".
- to minimise the encoding bias (i.e. reduce the representation choices made purely for the convenience of notation or implementation).
- to provide a consistent set of definitions;
- to minimise the need of revision by supporting the definition of new terms starting from terms already present in the ontology.
- to support dynamic selection and integration of services in SOA environments by minimising the duration of DL inferences;

The goals, scope and main use cases of the ASSERT4SOA Ontology are described in details in deliverable D3.1 [12].

1.3.1 Integrating concepts from different domains

In building the top layer of the ASSERT4SOA ontology, different existing ontologies and modelling approaches were examined (see deliverable D3.1 [12]). A selection of these ontologies and proposals from literature were used as a base to build part of the top ontology. Table 1 lists which ontologies, standards and models were used, and where they are referenced in the ASSERT4SOA Top Ontology. In Appendix A each concept derived from an existing work in literature is marked with the similar to tag, as well as the reference to the corresponding concept.

<table>
<thead>
<tr>
<th>Reference Work</th>
<th>Modules Addressed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Legal Knowledge Interchange Format - LKIF [15]</td>
<td>mereology, time, roles, mental states</td>
</tr>
<tr>
<td>MathWorld - Wolfram [17]</td>
<td>math (all)</td>
</tr>
<tr>
<td>Ontology Design Patterns [18]</td>
<td>math (all)</td>
</tr>
<tr>
<td>Resource Description Framework - RDF [19] (W3C Recommendation)</td>
<td>statement</td>
</tr>
</tbody>
</table>

Table 1  Reference work

1.3.2 Identifying concepts

The identification of the concepts to include in the ASSERT4SOA Ontology had the goal to list two main groups of concepts:
- a set of concepts specific to each ASSERT type, to be included in the ASSERT4SOA Ontology;
• a comprehensive vocabulary, expressive enough to define the concepts specific to certification processes in general, and to define the concepts belonging to the first group (the ASSERT4SOA Top Ontology).

The identification of the concepts belonging to the first group followed a bottom-up approach. The existing work for each type of ASSERT was analyzed independently, and modelled the most relevant concepts in UML. For concepts regarding ASSERT-E and ASSERT-M, the reference work used are deliverables D4.1 [21] and D5.1 [11], respectively.

In parallel, state of the art top ontologies and literature for common fields such as mathematics, time modelling approaches and the like were examined to extract basic concepts and properties. Comparing the general concepts to the UML models, some gaps emerged and were filled. This constituted a first version of the ASSERT4SOA Top Ontology.

Then the models of ASSERTs were translated in OWL, trying to define each concept in terms of the Top Ontology. In parallel, the top ontology was refined and integrated with missing terms.

Finally, the Top Ontology was optimized removing concepts that emerged as not necessary.

1.4 Modelling Services and Security Properties

One of the objective of the ASSERT4SOA Ontology is to support interoperability and comparison of different kinds of certificates. This objective entails the capability to compare the system models carried by the different kind of certificates. To achieve this objective the following chapters present a mapping between each ASSERT specific service model and the ASSERT4SOA Top Ontology. The key benefit of the chosen encodings is that they allows to compare ASSERT-E system models (assert-e:ServiceModel) and ASSERT-M service models (semf:System) since both (Figure 2):

- extend the wsdl:Service and the organisation:OrganisationalStructure concept;
- describe system's behaviour as sequences of activity:Activity hence for both service models it is possible to represent:
  - the interface provided by the service via the wsdl:interface object property inherited by the wsdl:Service class;
  - the roles and their relationships via the mereology:hasConstituency object property inherited by the organisation:OrganisationalStructure;
  - the sequence of activities describing the behaviour of the service via the activity:performs property.
Having a service model (Figure 3) generalising both semf:System (i.e. ASSERT-M service models) and assert-e:ServiceModel (i.e. ASSERT-E service models) allows to compare ASSERT-E security properties (i.e. sets of ASSERT-E service models) and ASSERT-M security properties (i.e. sets of ASSERT-M service models).

**Figure 2** Mapping between ASSERT-E, ASSERT-M and Top Ontology concepts.

**Figure 3** The ASSERT4SOA common service model class
Chapter 2

The ASSERT4SOA Top Ontology

The following chapter describes the main concepts of the ASSERT4SOA Top Ontology, and puts a link with the sources used to derive the concepts that will be used to model ASSERTS.

2.1 Ontology Design Patterns

This section describes how high level concepts are modelled in classes and object properties. These concepts are the basis to describe concepts in the certificates specific layers.

2.1.1 Ordering

To enable the possibility to model ordered concepts, the Top Ontology instances the Sequence ontology design pattern [18], using four object properties \(\text{precedes}, \text{follows}, \text{directlyPrecedes}, \text{directlyFollows}\), defined in the math module (see appendix 0). Asserting inverses, transitivity and subclasses between these properties, enables to model all types of generic ordering (see Figure 4). These high level properties are the basis to model a variety of concepts, from lists (data structure) to spatial and temporal relations.

![Ordering properties](image.png)

**Figure 4** Ordering properties
In particular, to model the concept of Sequence, the Top Ontology follows the approach presented in [23]. This approach allows to represent regular expressions as OWL DL classes and to use standard DL reasoning for matching or even inferring hierarchies between regular expressions. The sequences are modelled using specializations of the `directlyPrecedes` and `precedes` properties, namely `hasNext` and `isFollowedBy`. The content of the Sequence is asserted using the `hasContents` property. An example of use of these properties is given in Figure 5.

![Figure 5](image)

**Figure 5** Sequences in the ASSERT4SOA Ontology

With these concepts it is possible to define a variety of patterns of sequences, exploiting the relationships between the ordering properties above.

```
Class: Seq_ABC
  EquivalentTo:
    math:Sequence
    and math:hasContents some A
    and hasNext some (Sequence and
    hasContents some B and
    hasNext some (Sequence and
    hasContents some C and
    hasNext some EmptyList ))
```

**Figure 6** definition of Seq_ABC (Figure 5) in OWL

Taking as a basis the OWL definition in Figure 6 (expressed in Manchester Syntax [24]), it is easily possible to express variations of that type of sequence. For example, replacing the first occurrence of `hasNext` (a non transitive, functional object property) with `isFollowedBy` (super property of `hasNext`, transitive), it is possible to express patterns of sequences of the type (A, ..., B, C).

Using this approach, it is possible to exploit the reasoner capabilities, to perform pattern matching on sequences. In fact, for example, instances of the type Seq_ABC in Figure 5, are also classified as subclasses of sequences of the type (A, ..., B, C) or (A, B, ...).
2.1.2 Set

To model sets, the approach followed is a simplification of the content pattern Set in [18]. With respect to the original pattern, we decided not to model the Collection concept, but Set was directly defined as anything having members, declared using the hasMember property in the mereology module. By default, OWL treats equal elements as one, so it was not necessary to specify that sets contain only one occurrence of each element. The use of the Set class in the ontology was introduced even if, at a first glance, OWL classes can be seen as set definitions themselves. The need to model an ad hoc class, derives from the necessity to express properties over sets, and to be able to assign individuals of the same class to different sets, without having to overload the ontology with unnecessary classes.

2.1.3 Statement

This pattern allows to describe object properties as OWL entities. The approach is borrowed from the RDF built-in vocabulary intended for describing RDF statements [19]. A use case for this pattern within the ASSERT4SOA Ontology is the modelling of qualificatory expressions (see section 2.2.3 Mental States). Figure 7 depicts the key elements of the Statement ontology pattern and its mapping on the QualificatoryExpression class.

![Figure 7 Elements of the Statement pattern](image)

2.1.4 Role

This pattern has been proposed in [25] to deal with situation where there is the need to represent property both as an OWL object property and in a reified form by means of an OWL class. A use case for this pattern within the ASSERT4SOA Ontology is the modelling of organisational relationships between roles (see section 0 Organisational Structure). Figure 8 depicts the key elements of the Role ontology pattern and its mapping on the Trust class.
2.1.5 Dependency

This pattern allows to describe situations where some entity \((\text{dependee})\) depends on some else \((\text{dependee})\) for something \((\text{dependum})\). This pattern is informed by the concept of dependency proposed in [26].

A use case for this pattern within the ASSERT4SOA Ontology is the modelling of the different types of organisational relationships (see section see section Organisational Structure).

Figure 9 depicts the key elements of the Dependency ontology pattern and its mapping on the Trust class.
2.2 Modelling Examples

2.2.1 Mathematical Structures

The ASSERT4SOA Top Ontology comprises a vocabulary to describe mathematical concepts. Since security requirements are often concerned with some sort of ordering (e.g. to prescribe the order of messages exchanges in a communication protocol), the description of ordered list is mandatory for an ontology suitable to describe these concepts.

The Top Ontology represents ordered lists following the approach presented in [23]. This approach allows to represent regular expressions [26] as OWL DL classes and to use standard DL reasoning for matching or even deciding subsumption between them. The Sequence class (defined within the math namespace of our Top Ontology) can be used to represent the ordered list of formal parameters of a login operation provided by some SOA authentication service.

The concept of mathematical function is subsumed in most of the proposal for semantic web services (e.g. atomic processes in OWL-S [27]). The ASSERT4SOA Top Ontology defines the class Function to describe mathematical functions in terms of their domain, codomain and association rule. Function class can also be used to define the term RetrieveUserCredentials as a mathematical function having as domain the type UserIdentifier and as codomain the type UserCredential.

2.2.2 Agents

The ASSERT4SOA Top Ontology is designed to support the description of a SOA service as a multi-agent system [13] [14]. Figure 10 depicts the agent model defined by the ASSERT4SOA Top Ontology. An Agent is an entity coupled with an Environment. An Environment is constituted by Signals. An Agent is coupled to its Environment through its Sensors and Actuators. A Sensor allows an Agent to perceive its Environment by associating Signals to Stimuli. An Actuator allows an Agent to act on its Environment by producing Signals. The Signals produced by an Agent depend on its State. The State of an Agent contains both Signals and Stimuli. The State of an Agent can be the result of a Perception, an Action, or a Decision. Perception and Decision change a state by adding or removing Stimuli. An Action can change a state by adding or removing Signals.
Table 2 shows the mappings between the WSDL schema elements and concepts within the ASSERT4SOA Top Ontology:

<table>
<thead>
<tr>
<th>WSDL Element</th>
<th>ASSERT4SOA Top Ontology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service</td>
<td>agents:Agent</td>
</tr>
<tr>
<td>Interface</td>
<td>agents:Actuator, agents:Sensor</td>
</tr>
<tr>
<td>InterfaceOperation</td>
<td>agents:Perception, agents:Action</td>
</tr>
<tr>
<td>InterfaceMessageReference</td>
<td>agents:Signal</td>
</tr>
<tr>
<td>InterfaceFaultReference</td>
<td>agents:Signal</td>
</tr>
</tbody>
</table>

Table 2  Mapping between WSDL elements and ASSERT4SOA Ontology

2.2.3 Mental States

To achieve the objective of representing the content of security certificates, the ASSERT4SOA Top Ontology borrows the symbolico-cognitivist approach based on propositional attitudes [28] and speech acts adopted by the Legal Knowledge Integration Framework (LKIF) [15]. Propositional attitudes represent mental postures (e.g. belief, desire, intention) that agents may have towards a proposition. Propositions tell something about a subject (e.g. a SOA service). Expressions are propositions borne by some medium (e.g. a security certificate document). Types of expressions include qualifications (e.g. “Service S is an integrity preserving system”), entitlements (e.g. “Service S has the right to access file X”), evaluations (e.g. “Service S is better than service T”).

We extended the LKIF framework by introducing additional propositional attitudes and expressions to represent requests (e.g. expressions like “Agent X requires that service S …”) and executive expressions (e.g. “operation O is applied to parameter P”). Figure 11 shows some of the definitions we introduced to extend the LKIF set of propositional attitudes and expressions.
The *Request* class is defined as anything referring to some *Expression* via the *request* property. Instances of this class represent those natural language expressions having the form “*X requires that Y*”. Any instance of the *Request* class is a propositional attitude since in LKIF the class *Communicated_attitude* is a subclass of *Propositional_attitude*.

We also introduced additional object properties to describe expressions as defined classes, while LKIF models them only as primitive classes. This choice allows to better define compliance expressions which are of primary importance to describe the content of security certificates.

A *Qualificatory_Expression* can be used to attribute some quality to some entity. We use this type of expressions as a basis to represent natural language expressions having the form “*X is an Y*” or “*X is qualified as Y*”. The *ConformanceExpression* in Figure 11 is an example of that.

Finally, our ontology introduces terms denoting the three major types of inference [29]: deduction, induction, and abduction. The availability of these terms allows to classify the different certificates also according to the different types of evaluation process used to produce the certificate. For example, evaluation processes based on formal methods use deductive reasoning while test based certification relies on inductive reasoning. Section 3.2 describes an example of application of use of the *Deduction* class.

### 2.2.4 Activities

The interactions between an agent and its environment are described by means of the concepts of activity as fostered by Activity Theory [30]. According to Activity Theory an agent (subject) performs an action on an object using tools and producing an outcome that is dependent on the object itself.

---

Class: Request
EquivalentTo:
  request some communication:Expression
SubClassOf:
  communication:Communicated_Attitude

Class: Executive_Expression
EquivalentTo:
  (communication:hasSubject some wsdl:InterfaceOperation)
  and (statement:hasObject some owl:Thing)
SubClassOf:
  communication:Expression

Class: Qualificatory_Expression
EquivalentTo:
  (communication:hasQualification some owl:Thing)
  and (communication:hasSubject some owl:Thing)

Class: ConformanceExpression
EquivalentTo:
  (conformsTo some SecurityPropertyReference)
  and (communication:hasSubject some wsdl:Endpoint)

Figure 11 New expressions and sample proposition from the ASSERT4SOA Ontology
The vocabulary presented in Figure 12 to describe Activities, allows to specify the six key aspects of an activity:

- the agent (Subject) involved in the interaction is the agent holding the Intention referenced via the hasGoal object property;
- the goal that the agent intend to achieve by means of the interaction via the hasGoal object property;
- the entities (Object) being transformed is the domain of the function referenced via the performs object property;
- the transformation (Operation) being performed in the interaction via the performs object property;
- the result (Outcome) produced by the activity is the codomain of the function referenced via the performs object property;
- the object (Tool) mediating the interaction between the agent and the object of transformation via the useTools object property.

Activities and Sequences are the basis for the definition of the concept of Behaviour, which is defined as a sequence of activities observed by some agent. A CommunicationActivity is an Activity representing a one-way communication between two types of agents. The type of the senders is the same as the type of the subject of the CommunicationActivity. The domain of the CommunicationActivity function, of type Executive_Expression, specifies the type of message sent to the receivers. The codomain of the function represents the type of expected behaviour of the receivers upon receiving a message. The type of the receivers is the same as the type of the subjects of the activities in the Behaviour.

Table 3 shows how elements of UML sequence diagrams map on the vocabulary shown in Figure 12. This mapping allows to describe a pattern of sequence diagram that can be used like the description of PSL occurrence tree [31]. The description of the full mapping between UML sequence diagrams and ASSERT4SOA class definitions is out of the scope of this document.
An Example

In this section we show how the sequence diagram of the Authentication Enforcer security pattern [32] (shown in Figure 13) can be encoded using concepts from the ASSERT4SOA Top Ontology (Figure 14).

![Authentication Enforcer Sequence Diagram](image)

**Figure 13** Authentication Enforcer Sequence Diagram

An *AuthenticatingSystem* is anything that performs some *AuthenticationBehaviour*. An *AuthenticationBehaviour* is a Behaviour, since all its elements have content of type *Activity* (both *doCreateRequestContext* and *doInvokeAuthenticate* are subclasses of *Activity*). Indeed, *AuthenticationBehaviour* is a pattern of activities since *doInvokeAuthenticate* is referenced via the *isFollowedBy* object property, which is defined in our ontology as in [23] (i.e. it plays the same role as the ‘*’ wildcard in regular expressions).
Any instance of activity starting with a `doCreateRequestContext`, followed by some `doInvokeAuthenticate`, is classified as an `AuthenticationBehaviour` by a DL Reasoner.

Class: AuthenticatingSystem
EquivalentTo:
- organisation:OrganisationalStructure
  and activity:performs some AuthenticationBehaviour

Class: AuthenticationBehaviour
EquivalentTo:
- math:Sequence
  and (math:hasContents only doCreateRequestContext)
  and (math:isFollowedBy some
    (math:Sequence
      and (math:hasContents only doInvokeAuthenticate)))

Class: doCreateRequestContext
EquivalentTo:
- activity:Activity
  and (activity:hasSubject some Client)
  and (activity:performs only CreateRequestContext)

Class: doInvokeAuthenticate
EquivalentTo:
- activity:CommunicationActivity
  and (activity:hasSubject some Client)
  and (activity:performs some Authenticate)

Class: Authenticate
EquivalentTo:
- math:Function
  and (math:hasDomain some AuthenticationRequest)
  and (math:hasCodomain some AuthenticationEnforcerBehaviour)

Class AuthenticationEnforcerBehaviour
EquivalentTo:
- math:Sequence
  and (math:hasContents only doRetrieveUserCredentials)
  and (math:hasNext some
    (math:Sequence
      and (math:hasContents only doCreateSubject)
      and (math:hasNext some doReturnSubject)))

Class doReturnSubject
EquivalentTo:
- CommunicationActivity
  and (activity:hasSubject some AuthenticationEnforcer)
  and (activity:performs some ReturnSubject)

Class ReturnSubject
EquivalentTo:
- math:Function
  and (math:hasDomain some AuthenticationReply)
  and (math:hasCodomain some activity:Activity)

Figure 14 Authentication Enforcer Security Pattern
The `doCreateRequestContext` is an activity that executes a `CreateRequestContext` function, returning a `RequestContext`, and performed by the Client agent.

On the other hand, the `doInvokeAuthenticate` is a communication activity and, as any communication activity, it maps `Executive_Expressions` on `Behaviour` (see section 2.2.4). In this case it represents the authentication requests submitted by the client agents and producing the consequent activation of the `AuthenticationEnforcer`.

The behaviour of the `AuthenticationEnforcer` agents is defined by the class `AuthenticationEnforcerBehaviour`. The message returned by the Authentication Enforcer to the Client is described by the `doReturnSubject` communication activity. Note that since the codomain of the `ReturnSubject` function has type `Activity`, the `AuthenticationBehaviour` class is able to match any OWL individual representing the behaviour of any specific system performing some activity upon the reception of a `Subject` by the `Authentication Enforcer`.

### 2.2.5 Organisational Structures

Typical security requirements (e.g. Separation of Duties (SoD) [33]) are concerned with static (i.e. time independent) rather than dynamic aspects of a system. Security models like RBAC [34] rely on the concept of role to define static aspects of access control policies. The ASSERT4SOA Top Ontology defines the concept of role to support the description of this type of security requirements.

The possibility to describe dependencies between roles has been recognized as a key concept in requirements engineering [35] and in the modelling of multi-agents systems [13]. Dependencies between roles are also at the basis of proposals integrating Security Engineering with Requirement Engineering for the verification of security and trust requirements [27].

The ASSERT4SOA Top Ontology allows to denote different types of dependencies (e.g. task delegation, provisioning of resources) between roles. The `Trust` type is useful when describing security certificate models. A trust dependency denotes that an agent playing a role (depennder) depends on another agent playing a second role (dependee) for a belief (i.e. the first agent believes a proposition because asserted by an agent he trusts). Section 3.2 shows a use case for the `Trust` dependency type.

Role and role dependencies contribute altogether to define the organisational structure of an organisation. Organizational structures are a conceptual tool to characterise how tasks, data and resources are distributed to agents participating in an organisation [13] [14].

The `OrganisationalRelationship` class fulfils requirement `MO4.FUN` (representation of roles and organisational relationships) since:

- the `hasDepender` object property allows to retrieve the depender role;
- the `hasDependee` object property allows to retrieve the dependee role;
- the type of subclass of the organisationalRelationship class represent the type of dependency existing between two roles;
• the hasDependum object property allows to retrieve the dependum creating a dependency between a depender and a dependee role;

![Figure 15 Vocabulary for Organisational Structures and Roles](image)

### 2.2.6 Requirements

The ASSERT4SOA Top Ontology uses an approach based on denotational semantic [36] to specify the meaning of functions. Figure 16 shows the vocabulary for expressing both functional and non-functional requirements.
**Figure 16 Vocabulary for requirements**

*Requirements* are a type of *Expressions* (see section 2.2.1). A *FunctionalRequirement* is a type of *Requirement* specifying some pre-condition (*hasPrecondition*), some post-condition (*hasPostcondition*), some invariant (*hasInvariant*). The *isInterpretationFor* object property allows to specify the term being defined by *FunctionalRequirement*.

The *Map* abstract data type [37] is often encountered in the definition of software features related to Security.

The Authentication Enforcer’s cache and the User Store found in the Authentication Enforcer security pattern [32] can be modelled after the *Map* abstract data type. Figure 17 shows how this can be achieved by means of the ASSERT4SOA vocabulary for requirement.
and (math:hasAssociationRule some putExpression)

Class: K
SubClassOf:
   math:ConstantSymbol

Class: V
SubClassOf:
   math:ConstantSymbol

Class getExpression
EquivalentTo:
   math:Expression
   and (math:hasNext some getParams)
   and (communication:hasInterpretation some successfulGet)
   and (communication:hasInterpretation some failingGet))

Class getParams
EquivalentTo:
   math:Seq_Terms
   and (math:hasContents some K)

Class putExpression
EquivalentTo:
   math:Expression
   and (math:hasNext some putParams)
   and (communication:hasInterpretation some successfulPut)

Class putParams
EquivalentTo:
   math:Seq_Terms
   and (math:hasContents some K)
   and (math:hasNext some
      (math:Seq_Terms
         and (math:hasContents some V)))

Class successfulGet
communication:FunctionalRequirement
and (communication:hasInput some K)
and (communication:hasOutput some V)
and (communication:hasState some
   (math:Set
      and (mereology:hasMember some
         (math:Pair
            and (math:first some K)
            and (math:second some V)))))

Class failingGet
EquivalentTo:
   communication:FunctionalRequirement
   and (communication:hasInput some K)
   and (communication:hasOutput some NULL)
   and (communication:hasState some
      (math:Set
         and (not (mereology:hasMember some
            (math:Pair and (math:first some K)))))

Class successfulPut
EquivalentTo:
   communication:FunctionalRequirement
   and (communication:hasInput some putParams)
   and (communication:hasNextState some
      (math:Set

and (mereology:hasMember some
    (math:Pair
        and (math:first some K)
        and (math:second some V)))))

**Figure 17** Encoding of Map abstract data type.
Chapter 3

ASSERT-O Models

The role of Ontology-based certificates (ASSERT-O) is twofold:

- to enable the automatic verification of the compliance between a service model and the security property claimed in a security certificate. To achieve this objective ASSERT-O makes use of the OWL-Description Logic (DL). Since ASSERT-O makes use of a Description Logic formalism and of the (decidable) inferences available in it, the ASSERT-O certificates can be thought as a sort of model based certificates.
- to ease the import of pre-existing non-ASSERT certificates (e.g. Common Criteria certificates) into the ASSERT4SOA Framework by defining a general model for security certificate and certification process.

The following sections describe the ASSERT-O models for security properties, certificates, and service models. It is also explained how the ASSERT-O ontology addresses the requirements identified in [12].

3.1 Ontology-based properties

This section describes the use of the ASSERT4SOA Top Ontology for the description of security properties. Figure 18 contains the listing showing the vocabulary for the definition of Ontology-based properties in Manchester OWL Syntax [24].

A SecurityProperty is a function mapping service models to boolean values. The association rule of a SecurityProperty function is defined by means of security properties signatures. A SecurityPropertySignature is a first order logic term having the form \( f(C_1,...,C_n) \) where \( f \) is a symbol for function and \( C_i \) are symbols for constants. The interpretation of a SecurityPropertySignature is given by the means of ServiceModelRequirement which denote the class of FunctionalRequirement defined for an input a sequence of constants and a service model state.
The ASSERT-O model fulfils requirement **PO1.FUN** (representation of ontology-based security properties) since:

- the `math:hasContents` object property of the `SecurityPropertySignature` allows to retrieve the name of the property (note that a `SecurityPropertySignature` is a `math:Expression` and, hence, it has a `math:hasContents` object property);
- the `math:hasNext` object property of the `SecurityPropertySignature` allows to retrieve the parameters of a security property;
- the `communication:hasInterpretation` object property of the `SecurityPropertySignature` allows to retrieve the definition of the security property.

Figure 19 shows a sample security property definition given using the vocabulary in Figure 18. The Authentication of User security property is defined as a function having `ServiceModel` as domain, `Boolean` as codomain and the expression `Authentication(User)` as association rule. The evaluation of the `Authentication(User)` is `True` when it is evaluated in an `AuthenticatingSystem`.  

---

**Figure 18** Vocabulary for the definition of ontology-based properties
state once the the User parameter of the security property is mapped on the Client role of the AuthenticationEnforcer security pattern.

Class UserAuthenticationProperty
EquivalentTo:
  math:Function
  and (math:hasDomain some assert-o_ws:ServiceModel)
  and (math:hasCodomain some primitive-types:Boolean)
  and (math:hasAssociationRule some UserAuthenticationPropertyExpression)

Class UserAuthenticationPropertyExpression
EquivalentTo:
  math:Expression
  and (math:hasNext some UserAuthenticationPropertyParams)
  and (communication:hasInterpretation some UserAuthenticationPropertyRequirement)

Class UserAuthenticationPropertyParams
EquivalentTo:
  math:Seq_Terms
  and (math:hasContents only Authentication)
  and (math:hasNext some
    (math:Sequence
      and (math:hasContents only User)))

Class: Authentication
SubClassOf:
  math:ConstantSymbol

Class: User
EquivalentTo:
  authenticationEnforcer:Client
SubClassOf:
  math:ConstantSymbol

Class authenticationEnforcer:AuthenticationEnforcer
SubClassOf:
  assert-o_ws:ServiceModel

Class AuthenticatingSystem
EquivalentTo:
  MatchingModel
  and authenticationEnforcer:AuthenticationEnforcer

Class UserAuthenticationPropertyRequirement
EquivalentTo:
  communication:FunctionalRequirement
  and (communication:hasInput some UserAuthenticationPropertyParams)
  and (communication:hasOutput only primitive-types:True)
  and (communication:hasState some AuthenticatingSystem)

Figure 19 Definition of the Authentication(User) security property
3.2 Ontology-based certificates (ASSERT-O)

This section describes the use of the ASSERT4SOA Top Ontology for the description of both the ASSERT-O Certification Process and the ASSERT-O Security Certificates.

Certification Process

This section shows how the ASSERT4SOA Top Ontology can be used to model the certification process assumed for the production and issue of ASSERT-O.

![ASSERT-O Issuing Sequence Diagram](image)

The issuing of an ASSERT-O proceeds across the following steps (Figure 20):

1. The Service Developer creates an OWL model of the service to be certified using the vocabulary provided by the ASSERT4SOA Top Ontology;
2. The Service Developer retrieves from an Ontology Manager (see [1]) the definition of the security property to be certified for the service;
3. The Service Developer creates an OWL model of the Security Controls implemented by the service. The Security Controls are a mapping between the service model created in step 1 and the security property definition retrieved in step 2.
4. The Service Developer sends to a Certification Authority a certification request containing the service model (from step 1), the security property (from step 2) and the security controls (from step 3);
5. The Certification Authority sends an evaluation request to the Evaluation Body. The evaluation request contains the same information contained in the certification request (service model, security property definition and security controls models);
6. The Evaluation Body checks the consistency between the model of the service, the security property definition and the model of the security controls by means of an OWL reasoner;
7. The Evaluation Body replies to the evaluation request by the Certification Authority using the results from step 6;
8. In case of positive result of the evaluation the Certification Authority creates an ASSERT-O for the service being certified;
9. The Certification Authority transmits to the Service Designer the ASSERT-O (from step 8) as a reply to the certification request of the Service Designer (step 1);
10. The Service Designer stores into the Ontology Manager the ASSERT-O received from the Certification Authority.

Figure 21 shows the encoding of the certification process by means of the ASSERT4SOA Top Ontology.

| Class: ASSERT-O_CertificationProcess |
| EquivalentTo: |
| ServiceModel |
| and activity:performs some CertificationRequestSubmission |

| Class: CertificationRequestSubmission |
| EquivalentTo: |
| math:Sequence |
| and (math:hasContents only doCreateWebServiceModel) |
| and (math:isFollowedBy some math:Sequence |
| and (math:hasContents only doRetrieveSecurityPropertyDefinition) |
| and (math:isFollowedBy some math:Sequence |
| and (math:hasContents only doCreateSecurityControl) |
| and (math:isFollowedBy some math:Sequence |
| and (math:hasContents only doSubmitCertificationRequest)) |

| Class doSubmitCertificationRequest |
| EquivalentTo: |
| CommunicationActivity |
| and (activity:hasSubject some ServiceDesigner) |
| and (activity:performs some SubmitCertificate) |

| Class SubmitCertificate |
| EquivalentTo: |
| math:Function |
| and (math:hasDomain some CertificationRequest) |
| and (math:hasCodomain some EvaluationRequestSubmission) |

| Class EvaluationRequestSubmission |
| EquivalentTo: |
| math:Sequence |
| and (math:hasContents only doSubmitEvaluationRequest) |

| Class doSubmitEvaluationRequest |
| EquivalentTo: |
| activity:CommunicationActivity |
| and (activity:hasSubject some CertificationAuthority) |
| and (activity:performs some Evaluate) |

| Class Evaluate |
| EquivalentTo: |
| math:Sequence |
| and (math:hasContents only doCheckSecurityControls) |
| and (math:isFollowedBy some
The process described above does not consider the case when a pre-existing non-ASSERT certificate is available. In this case the process remains mostly unchanged and the pre-existing certificate is presented as input to the certification process. Next release of this document will present a more extended version of the process.

Security Certificate
The section shows how to model a semantic security certificate (ASSERT-O). Figure 22 shows the first version of the ASSERT-O certificate model being defined by the ASSERT4SOA Project. As explained in the previous sections this model belongs to the Certificates Models Layer of the ASSERT4SOA Ontology.
A certificate of ASSERT-O type is defined as anything being the carrier of some conformance expressions (ConformanceExpression) which are asserted by a legal representative of a Certification Authority (CALegalRepresentativeAssertion). ASSERT-O are a type of Document.
The model describes how assertions by the legal representatives of the Certification Authority depend on the trust relation they have on the Authorised Lead Appraiser of the Evaluation Body. These trust relationships are modelled using the concept of Trust. This concept is related to the definition of \texttt{CALegalRepresentativeAssertion} via the \texttt{isDependumOf} object property. The trustee of the relation is the CA Legal Representative and the trustee is the EB Authorised Lead Appraiser.

Any conformance expression appearing in an ASSERT-O Certificate is believed to be true by an agent member of an evaluation body (\texttt{EBAgent}) and playing the role of authorized lead appraiser (\texttt{AuthorisedLeadAppraiser}). The appraiser is a role part of the organizational structure of some Evaluation Body (EB).

The definition of ASSERT-O also states that the beliefs of the authorized lead appraiser are the result of a deduction occurring in the context (\texttt{activity:isPerformedIn}) of an ontology-based evaluation and using as input (\texttt{math:hasDomain}) the description of security controls described externally to the certificate (\texttt{agents:SecurityControlReference}).

An ontology-based evaluation is defined as an activity that applies deductive inferences using an OWL Reasoner as tool.

Finally, an ontology-based certificate may have one expiration date and may indicate the time interval when the assessment took place.

**Fulfilment of requirements**

This section describes how the ASSERT-O model presented above fulfils the requirements stated in [38] for ASSERT-O certificates (i.e. supports the answer to the competency questions concerning ontology-based certificates).

The ASSERT-O class fulfils requirement \texttt{AO1.FUN} (representation of ontology-based certificates) since:

- the \texttt{isCarrierFor} object property allows to retrieve the \texttt{ConformanceExpression} associated to an ASSERT-O. A \texttt{ConformanceExpression} (see Figure 11) represents a security statement by linking Security Properties (via the \texttt{conformsTo} object property) to service endpoints (via the \texttt{hasSubject} object property).
- the \texttt{isCodomainOf} object property of the \texttt{EBAuthorisedLeadAppraiserBelief} (reachable via the \texttt{attitude} object property of the \texttt{ConformanceExpression} appearing in ASSERT-O) allows to retrieve the information about the proof (\textit{Deduction}) supporting a security statement (i.e. a \texttt{ConformanceExpression}).

The \texttt{ASSERT-O} class fulfils requirement \texttt{AO2.FUN} (representation of security statements) since:

- the \texttt{hasSubject} object property of the \texttt{ConformanceExpression}, referenced via the \texttt{isCarrier} property, allows to retrieve the certified service;
- the \texttt{conformsTo} object property of the \texttt{ConformanceExpression}, referenced via the \texttt{isCarrier} property, allows to retrieve the security property asserted;
- the \texttt{isCodomainOf} object property of the \texttt{EBAuthorisedLeadAppraiserBelief} (reachable via the \texttt{attitude} object property (defined in \texttt{mental-states} module) of the \texttt{ConformanceExpression} appearing in ASSERT-O) allows to retrieve the information about the proof (\textit{Deduction}) supporting a security statement (i.e. a \texttt{ConformanceExpression}).
The OntologyBasedProof class fulfils requirement AO3.FUN (representation of ASSERT-O proofs) since:
- the targetServiceModel object property of the SecurityControlReference referenced by the hasDomain property, allows to retrieve the service model used in the proof;
- the hasDomain object property allows to retrieve the security controls considered in a proof;
- the useTools of the OntologyBasedEvaluation referenced via the isPerformedIn property, allows to retrieve the reference to the reasoner used to verify the proof (OWLReasoner).

The SecurityControlReference class fulfils requirement AO4.FUN (representation of security controls) since:
- the matchedSecurityPattern object property allows to retrieve the security pattern matched by the security control;
- the OWL ontology referenced by the hasMapping object property describes how the mapping between the security pattern and the service model is realised by means of OWL native constructs (e.g. SubClassOf, EquivalentTo, SubPropertyOf, SubPropertyChain);
- the elements of the service model (referenced by the targetServiceModel property) realising a security control are those mapped by the OWL ontology referenced by the hasMapping property;

3.3 ASSERT-O service model

This section describes the use of the ASSERT4SOA Top Ontology for the description of ASSERT-O Service Model.

A service model is anything performing some behaviour and being both an organisational structure and a service (in the sense defined by the WSDL standard). In the ASSERT4SOA Ontology, service models play the same role that Processes have in OWL-S [27].

A Security Pattern is a service model that is a state evaluating a SecurityPropertySignature to the TRUE value. The definition of the concept of Service is largely borrowed from the WSDL standard.

The ASSERT-O model fulfils requirement MO1.FUN (representation of service implementations) since:
- the interface object property, inherited via the Service class, allows to retrieve the interface offered by the service;
- the performs object property of the activities contained within the behaviour referenced via the performs property, allows to retrieve the functions implemented by the service;
- the hasDepender and hasDependee object properties of the organisational relationships referenced by the hasConstituency property, inherited via the OrganisationalStructure class, allow to retrieve the roles of a service implementation;
the hasConstituency object property, inherited via the OrganisationalStructure class, allows to retrieve the relationships existing between the roles of a service implementation;

- the performs object property allows to retrieve the workflow of a service implementation represented via the Behaviour class (see section 2.2.4);

- the useTools object property of the activities contained within the behaviour referenced via the performs property, allows to retrieve the resources needed by a service implementation;

- the hasContents object property, inherited via the Sequence class by the Behaviour class, allows to retrieve the context of use of resources used by a service implementation;

---

Figure 23 Vocabulary for ASSERT-O Service Model

The SecurityPattern class fulfils requirement AO5.FUN (representation of security patterns) since:

- the (inverse(hasSate) o isInterpretationFor o inverse(hasAssociationRule) o is_SecurityProperty) object property chain of a service model allows to retrieve the security properties provided by a security pattern;

---
the static elements of a security pattern can be retrieved thanks to:
- the `hasConstituency` object property inherited via the `ServiceModel` class (see MO1.FUN above)
- the `useTools` object property of the activities contained within the behaviour referenced via the `performs` property (see MO1.FUN above)
- the `performs` object property inherited via the `ServiceModel` class allows to retrieve the expected behaviour of a security pattern:

The `Interface` class fulfils requirement MO2.FUN (representation of service interface) since:
- the `interfaceOperations` object property allows to retrieve the operation of an interface;
- the URI of the OWL instance or OWL subclass representing an interface operation allows to specify the name of the operation;
- the `style` object property allows to retrieve the style of an operation;
- the `input` sub-property of the `interfaceMessageReference` object property allows to retrieve the formal input parameters of the operation (encoded by means of a Sequence);
- the `output` sub-property of the `interfaceMessageReference` object property allows to retrieve the output of an operation (encoded by means of a Sequence);
- the `interfaceFaultReference` object property allows to retrieve the possible faults of an operation;

The `ServiceModel` class fulfils requirement MO3.FUN (representation of business objectives pursued by services) since:
- the `hasGoal` object property of the activities contained within the Behaviour referenced by the `performs` property, allows to retrieve the business objectives of a service;
- the `hasDomain` object property of the function in each service activity allows to specify the input needed to achieve the objective of the activity;
- the `hasCodomain` object property of the function in each service activity allows to specify the output representing the achievement of the objective of an activity;
- the nesting of activities allowed by means of communication activities (see section 2.2.4) allows to establish the decomposition of objectives: the objectives of the activated behaviour represents the decomposition of the goal of the initiating activity;
- the set of objectives in a sequence of activities connected only via the `hasNext` object property (see section 2.1.1) represents a full decomposition of the initiating activity;
- the order in which the activities appear in a behaviour also specifies the order of achievement of the objectives.
Chapter 4

ASSERT-E

The ASSERT-E ontology models concepts pertaining to evidence based certificates (ASSERT-E), as defined in deliverable D4.1 [21]; the related classes and object properties are listed in details in appendix A.1.21.

The following sections describe the ASSERT-E ontology entities used to model respectively service models, security properties, and certificates. It is also explained how the ASSERT-E ontology addresses the requirements identified in [12].

Since one of the objectives of the ASSERT4SOA Ontology was to investigate the use of Semantic Web technologies to support ASSERT interoperability, this section focuses the encoding of those components (i.e. service models and security properties) of the ASSERT-E that need to be compared across different types of ASSERTs.

4.1 Representing ASSERT-E Service Models in OWL

4.1.1 Symbolic Transition System

Symbolic Transition Systems (STS) are the basis for the definitions of ASSERT-E Services Model. A Symbolic Transition System is a tuple $\langle S, s_0, V, J, Act, \rightarrow \rangle$, where $S$ is a set of states, $s_0$ is the initial state, $V$ is the set of internal variables, $J$ is the set of interaction variables, $Act$ is the set of actions, and $\rightarrow$ is the transition relation.

The following OWL expression is the description of the class of Symbolic Transition Systems using concepts from the ASSERT4SOA top ontology:

```owl
Class: assert-e:SymbolicTransitionSystem
EquivalentTo:
  - wsdl:Service
    and (assert-e:hasInitialState some assert-e:State)

ObjectProperty: assert-e:hasInitialState
```

D.3.4 – Second Version Of The ASSERT Ontology 45 / 165
The encoding chosen for Symbolic Transition System within the ASSERT4SOA Ontology allows to retrieve all the components of the tuple defining a Symbolic Transition System (e.g. see below for examples of such kind of queries):

Example
We have The STS model of an upload function that takes in input the filename name, and the pathname path, and returns in output an integer greater than zero if the upload has been successfully. Upon receiving the input, a transition to state $s_1$ is executed if and only if both input variables do not contain a null value, otherwise an error is returned. Upon generating the output, a transition to state $s_2$ is executed and the upload result returned to the caller [21].

The STS model of the above function is described by the following tuple:

$$\langle \{s_0, s_1, s_2\}, \{\text{name}, \text{path}\}, \{\text{res}\}, \{\text{req}\}, \{\text{upload}\}, \{s_0, \text{rcv\_req}, s_1\}, \{s_1, \text{send\_res}, s_2\}, (s_2, \varepsilon, s_0)\rangle$$

Where $\mathcal{S} = \{s_0, s_1, s_2\}$, the initial state is $s_0$, $\mathcal{V} = \{\text{res}\}$, $\mathcal{I} = \{\text{req}\}$, $\mathcal{Act} = \{\text{upload}\}$, and $\rightarrow = \{(s_0, \text{rcv\_req}, s_1), (s_1, \text{send\_res}, s_2), (s_2, \varepsilon, s_0)\}$. The above tuple is encoded by the following OWL expressions:

Class: s:STS_Upload
EquivalentTo:
  s:UploadInterface
  and assert-e:hasIntitialState some s:S0

Class: s:S0
EquivalentTo:
  assert-e:State
  and (math:hasContents some s:RCV_Req)
  and (math:hasNext some s:S1)

Class: s:S1
EquivalentTo:
  assert-e:State
  and (math:hasContents some s:SEND_Res)
  and (math:hasNext some s:S2)

Class: s:S2
In the above descriptions the ε-transition \((\sigma_1, \varepsilon, \sigma_2)\) is encoded by asserting the equivalence between the OWL classes representing states \(\sigma_1\) and \(\sigma_2\).

### 4.1.2 Service Model

The ASSERT-E ontology models services by extending both Symbolic Transition Systems (STS) [21] and Web Service Architecture [8] (defined within the \(wsa.owl\) module).

The following OWL expression is the description of the ASSERT-E service model given in the ASSERT-E ontology:

```owl
Class: assert-e:ServiceModel
EquivalentTo:
  assert-e:STS
  and wsa:WebServiceArchitecture
```

The following OWL expression excerpted from the \(wsa\) module, defines the `WebServiceArchitecture`:

```owl
Class: wsa:WebServiceArchitecture
EquivalentTo:
  mereology:hasConstituency some wsa:RequestRelationship
```

Class: wsa:RequestRelationship
EquivalentTo:
4.2 Representing Container-level Security Properties

This section presents how the ASSERT4SOA ontology can be used to represent in OWL some of the Container-level security properties defined in [21]. Container-level security properties (e.g. Authenticity, Integrity) are provided by services deployed in a container because of the compliance of the container with web services security standards (i.e. XML Encryption). In particular, in the chosen approach the container-level security properties presented in [21] are described as subsets of the ASSERT-E services (i.e. assert-e:ServiceModel) complying with the Web Services Architectures [8] and implementing the WS-Security standard [9].

4.2.1 Confidentiality

ASSERT-E services achieve the Confidentiality property by encrypting SOAP messages using the set of mechanisms defined by the WS-Security standard (i.e. XML Encryption) [21]. ASSERT4SOA ontology allows to describe the ASSERT-E Confidentiality property as the set of web services complying with the Web Services Architectures [8] and implementing the XML Encryption processing [40] as expressed by the following OWL expression:

Class: assert-e:Confidentiality
EquivalentTo:
assert-e:ServiceModel
and (activity:performs some xmlenc:XMLEncryptionProcessing)
Example
The set of architectures corresponding to the concrete security property
\( p_1 = (confidentiality, \{algorithm = DES\}) \) is described by the following OWL
expression:

```
Class: s:Confidentiality_DES
EquivalentTo:
    wsa:WebServiceArchitecture
    and activity:performs some s:XMLEncryptionProcessing_DES
```

```
Class: s:XMLEncryptionProcessing_DES
EquivalentTo:
    math:Sequence
    and (math:hasContents some s:EncryptData_DES)
```

```
Class: s:EncryptData_DES
EquivalentTo:
    activity:Activity
    and activity:hasOutcome only s:EncryptedData_DES
```

```
Class: s:EncryptedData_DES
EquivalentTo:
    xmlenc:EncryptedData
    and xmlenc:algorithm value xmlenc:tripledes-cbc)
```

### 4.2.2 Integrity

ASSERT-E services achieve the *Integrity* property by signing SOAP messages
using the set of mechanisms defined by the WS-Security standard (i.e. XML
Signature) \[21\]. The ASSERT4SOA ontology describes the ASSERT-E Integrity
property as the set of web services complying with the Web Services
Architectures \[8\] and implementing the XML Signature processing \[40\] as
expressed by the following OWL expression:

```
Class: assert-e:Integrity
EquivalentTo:
    assert-e:ServiceModel
    and (activity:performs some xmldsig:XMLSignatureProcessing)
```

Example
The set of architectures corresponding to the concrete security property
\( p_1 = (integrity, \{algorithm = RSA, |key| = 1024bit\}) \) is described by the following
OWL expression:

```
Class: s:Integrity_RSA_1024bit
EquivalentTo:
    wsa:WebServiceArchitecture
    and activity:performs some s:XMLDSigProcessing_RSA_1024bit
```
In the above expression \textit{s:GenerateReference} denotes the description of the activity producing the reference element (\textit{xmlsig:Reference}) of an XML signature [40].

### 4.3 Certificates Model

The concepts used to model an evidence-based certificate are reported in Figure 24, showing the definition of an ASSERT-E.

<table>
<thead>
<tr>
<th>Class: ASSERT-E</th>
</tr>
</thead>
<tbody>
<tr>
<td>SubClassOf: communication:Document</td>
</tr>
<tr>
<td>EquivalentTo:</td>
</tr>
<tr>
<td>communication:isCarrierFor some</td>
</tr>
<tr>
<td>(mental-states:attitude some ASSERT-E_Assertion and</td>
</tr>
<tr>
<td>(mental-states:attitude some</td>
</tr>
<tr>
<td>(EvaluationBodyAgentBelief and (math:isCodomainOf some</td>
</tr>
<tr>
<td>(mental-states:Induction and</td>
</tr>
<tr>
<td>(activity:isPerformedIn some TestBasedEvaluation))))</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class: ASSERT-E_Assertion</th>
</tr>
</thead>
<tbody>
<tr>
<td>SubClassOf: communication:Assertion</td>
</tr>
<tr>
<td>EquivalentTo:</td>
</tr>
<tr>
<td>communication:assert some SecurityStatement and</td>
</tr>
<tr>
<td>communication:asserted_by some</td>
</tr>
<tr>
<td>(CertificationAuthorityAgent and</td>
</tr>
<tr>
<td>roles:count-as some assert-e:LegalRepresentative)</td>
</tr>
</tbody>
</table>

| Class: SecurityStatement                            |
The **ASSERT-E ontology** fulfils requirement **AE1.FUN** (representation of evidence-based certificates) because:

- the `isCarrierFor` object property allows to retrieve the **ASSERT-E Assertions** associated to an ASSERT-E. An **ASSERT-E_Assertion** asserts some security statements relating a service (through the `hasSubject` object property) to a concrete security property (via the `hasQualification` object property);
- the service model considered during the certification process can be obtained via the `hasSubject` object property of the **SecurityStatement**;
- the `isCodomainOf` object property of the **EvaluationBodyAgentBelief** allows to retrieve the information on the inductive process (**TestBasedEvaluation**) followed by an Evaluation Body to prove the security statements included in the **ASSERT-E**. The **TestBasedEvidence**, i.e. the proof supporting the certificate, can be obtained via the `useTools` object property of the **TestBasedEvaluation**.  
- the `evaluatedBy` object property of **TestBasedEvidence** allow to retrieve the metrics used to evaluate the testing activities.

The concepts used to model a test-based evidence are reported in **Figure 25**.

---

---

---

---

---

---

---
The ASSERT-E ontology fulfills requirement AE2.FUN (representation of evidence-based proofs) because:

- the believed_by object property of EvaluationBodyAgentBelief allows to know the EvaluationBodyAgent, that is related to the EvaluationBody he/she belongs to via the isMemberOf object property,
- the isCodomainOf object property of the EvaluationBodyAgentBelief allows to retrieve the TestGeneration and TestExecution activities,
- the TestGenerationMethod (reachable via the useTools object property of TestGeneration) link test cases (hasCodomain object property) to the tests used to generate them (hasDomain object property). The test generation model used to generate test case can be obtained through the hasAssociationRule object property,
- tests used to generate test cases are instances of the Test class, that represent the class of test considered in the test based evidence
- a TestExecution activity performs some TestCaseMeasure using (useTools object property) some TestCase. The TestCaseMeasure is a measure with a nominal scale of measure with only two possible values (results): "Passed" and "Failed". This scale of measure is called BooleanNominalScale and is reachable via the hasCodomain object property of TestCaseMeasure.

Concepts related to tests and test cases are depicted in Figure 26.
The ASSERT-E ontology fulfils requirement **AE3.FUN** (representation of tests) because:

- the *Test* class represents tests that can be executed to prove that certain properties hold for a given services; tests (called test types in [12] and [21]) are therefore instances of the *Test* class. Tests specify how to produce the test cases required to certify a property basing on a test generation model. The *TestGenerationMethod* (reachable from the *TestEvidence*, as explained above) link test cases (*hasCodomain* object property) to the tests used to generate them (*hasDomain* object property). The test generation model used to generate test case can be obtained through the *hasAssociationRule* object property;
- each *Test* can be related to another *Test* via the *testStrongerThan* or *testWeakerThan* object properties that allow to specify if the given test implies (or is implied by) another test;
- a *Test* is related to the test attributes defining it via the *hasTestAttribute* object property. A test attribute is defined as a pair having a textual name and a textual value;
- *TestCase* is a subclass of *FunctionalRequirement*. Test case preconditions and postconditions are reachable respectively via the *hasPrecondition* and *hasPostcondition* object properties of *FunctionalRequirement*. The other attributes of a test case can be obtained via object properties of *TestCase*: hidden communications (through *hasHiddenCommunication*), input and outputs (pair reachable via the *hasAssociationRule* object property).
Chapter 5

ASSERT-M

This section describes the use of the ASSERT4SOA Top Ontology for the description of the concepts specific to the ASSERT-M model, as described in [11], while the detailed list of classes and object properties can be found in appendix A.1.24. Each sub-section describes how the different concepts fulfil the requirements stated in deliverable D3.1 [12] for ASSERT-M certificates (i.e. supports the answer to the competency questions concerning Model-based certificates).

In the following sections we will adopt the following typographical conventions within formulas:
- Lower case strings represent symbols
- Capital greek letters represent set of symbols

5.1 Representing SeMF in OWL

This section describes the use of the ASSERT4SOA Top Ontology and OWL primitives for the description of the concepts specific to the SeMF, as described in [11]. The detailed list of classes is reported in appendix A.1.23.

5.1.1 Agents

Within the ASSERT4SOA ontology the set concept of SeMF agents [11] is mapped onto the concepts of organisations:OrganisationalRole defined in the ASSERT4SOA Top Ontology by means of the following OWL assertion stating that SeMF agent are a subset of organizational role:

```
Class: semf:Agent
SubClassOf:
    organisations:OrganisationalRole
```
A set of SeMF agents can be described by means of the following class assertion:

Class: s:Users
SubClassOf: semf:Agent

5.1.2 Actions

SeMF describes systems in terms of actions performed by agents [11]. The ASSERT4SOA ontology models actions performed by SeMF agents as a specialisation of the concepts of Activity described within the activity module of the top ontology as shown by the following OWL assertion:

Class: semf:Action
EquivalentTo:
  activity:Activity
  and activity:hasGoal exactly 1
    (mental-states:Goal
     and
     mental-states:aimed_by exactly 1
     mental-states:CongnitiveAgent)

the above definition uses the chaining of object properties activity:hasGoal and mental-states:aimed_by to describe that a SeMF action is assigned to exactly one SeMF agent [11].

Example
The following OWL expressions uses the semf:Action class to represent the set of actions $\Sigma=\{\text{send-login, rec-login}\}$:

Class: Sigma
EquivalentTo:
  (s:rec-login, s:send-login)

Individual: s:send-login
  Types: Sigma

Individual: s:rec-login
  Types: Sigma

5.1.3 Traces

The set of all finite sequences of SeMF actions (traces) is defined by the Trace class:
Class: semf:Trace
EquivalentTo:
  math:EmptyList
or
  (math:Sequence
   and math:hasContents only semf:Action
   and math:isFollowedBy only
    (math:EmptyList
     or
      (math:Sequence
       and math:hasContents only semf:Action
      )))

The definition relies on the concept of sequence (math:Sequence) defined in the Math module of the ASSERT4SOA ontology.

Example
The following OWL class expression uses the semf:Trace class to describe that s:Gamma is a set of SeMF traces:

Class: s:Gamma
SubClassOf:
  semf:Trace

The set Σ* of all finite sequences of element of Σ, including the empty sequence ε is defined by the following expression:

Class: s:Sigma-star
EquivalentTo:
  math:EmptyList
or
  (math:Sequence
   and math:hasContents only s:Sigma
   and math:hasNext only s:Sigma-star)

The set of traces containing at least one s:send-login action is described by the OWL expression:

Class: s:TracesWith_send-login
EquivalentTo:
  (semf:Trace
   and math:hasContents value s:send-login
  )
or
  (semf:Trace
   and
    (math:isFollowedBy some
     (semf:Trace
      and math:hasContents value s:send-login
     ))
  )


The above definition can be generalized to describe the set of traces containing some action from the set of actions \texttt{s:Gamma} as follow:

\begin{verbatim}
Class: s:Gamma
SubClassOf:
   semf:Action

Class: s:TracesWith_Gamma
EquivalentTo:
   (math:Trace and math:hasContents some s:Gamma)
   or
   (math:Trace
   and (math:isFollowedBy some
   (math:Trace and math:hasContents some s:Gamma)
   )
   )
\end{verbatim}

5.1.4 Alphabetic language homomorphism

SeMF uses alphabetic language homomorphism $h^*: \Sigma^* \rightarrow \Sigma'^*$ to describe systems’ abstractions. An alphabetic language homomorphism is a mapping $h^*: \Sigma^* \rightarrow \Sigma'^*$ such the following hold [11]:

- $h'(xy) = h'(x)h'(y)$
- $h'(\epsilon) = \epsilon$
- $h'(\Sigma') \subseteq (\Sigma'\cup\{\epsilon\})^*$

This kind of homomorphisms are uniquely defined by their corresponding action mapping $h: \Sigma \rightarrow \Sigma'$ [11]. The action mapping $h$ between the actions of two sets of actions (e.g. \texttt{Sigma} and \texttt{Sigma1}) can be described by means of the OWL SameAs primitive.

The third condition can be described by asserting that the OWL class describing the image of the homomorphism is a subclass of the class describing the abstraction (i.e. the right hand side of the expression).

\textbf{Example}

This example shows how to assert in OWL and the ASSERT4SOA ontology the fact that a sequence \texttt{s1:Seq_x_y} is an abstraction of another sequence \texttt{s:Seq_send-login_rec-login} under the specific mapping $h=\{ \texttt{s:send-login} \rightarrow \texttt{s1:y}, \texttt{s:rec-login} \rightarrow \texttt{s1:x} \}$.  

The following expressions represent respectively the language containing the traces having the action \texttt{send-login} immediately followed by the action \texttt{rec-login} and the sequences of traces having the symbol \texttt{x} immediately followed by an \texttt{y}.

---

D.3.4 − Second Version Of The ASSERT Ontology 58 / 165
Class: s:Seq_send-login_rec-login
EquivalentTo:
  (math:Trace and math:hasContents value s:send-login)
  and
  (math:hasNext some
    (math:Trace and math:hasContents value s:rec-login)
  )

Class: s1:Seq_x_y
EquivalentTo:
  (math:Trace and math:hasContents value s1:y)
  and
  (math:hasNext some
    (math:Trace and math:hasContents value s1:x)
  )

the action mapping \( h = \{ s:\text{send-login} \rightarrow s1:y, s:\text{rec-login} \rightarrow s1:x \} \) is represented by:

Individual: s:send-login
  Types: Sigma
  SameAs: s1:y

Individual: s:rec-login
  Types: Sigma
  SameAs: s1:x

Individual: s1:y
  Types: Sigma1
  SameAs: s:send-login

Individual: s1:x
  Types: Sigma1
  SameAs: s:rec-login

The fact that \( s1:Seq_x_y \) is an abstraction for \( s:Seq_send-login_rec-login \) under the mapping \( h = \{ s:\text{send-login} \rightarrow s1:y, s:\text{rec-login} \rightarrow s1:x \} \) can be expressed by asserting that \( s:Seq_send-login_rec-login \) is a subclass of \( s1:Seq_x_y \):

Class: s:Seq_send-login_rec-login
SubClassOf: s1:Seq_x_y

Note that by asserting the alternative mapping (e.g. \( h = \{ s:\text{send-login} \rightarrow s1:x, s:\text{rec-login} \rightarrow s1:y \} \)) would result in an inconsistency detected by the OWL reasoner.
5.1.5 Agents’ Knowledge

Within SeMF agents’ knowledge about the global system behavior is integral part of a system specification [11]. An agents’ knowledge of the global system behavior is the set of traces that the agent P believes possible within the system.

The ASSERT-M ontology describes the association between an SeMF agent and the set of traces it believes possible using concepts from the communication and mental-states modules of the ASSERT4SOA top ontology. The following OWL expression is the description of SeMF agents’ knowledge within the ASSERT4SOA ontology:

Class: semf:AgentKnowledge
EquivalentTo:
  semf:Trace
  and (communication:isQualificationIn some
      (communication:Qualificatory_Expression
       and (mental-states:attitude some
        (mental-states:Belief
         and (mental-states:held_by some
          (mental-states:CongnitiveAgent
           and roles:count-as some semf:Agent)
          )
        )
      )
    )
  )

The above expression qualifies as agent knowledge any trace playing the role of qualification (communication:isQualificationIn) in a qualificatory expression (communication:Qualificatory_Expression) believed by an agent (mental-states:held_by) playing a role (roles:count-as) of SeMF agent.

Example

The knowledge ($W_p$) of an agent $P$ is described by the following OWL class expression:

Class: s:Wp
EquivalentTo:
  semf:Trace
  and (communication:isQualificationIn some
      (communication:Qualificatory_Expression
       and (mental-states:attitude some
        (mental-states:Belief
         and (mental-states:held_by some
          (mental-states:CongnitiveAgent
           and roles:count-as value s:p)
          )
        )
      )
    )
It has to be noted that the expression states that $s:W_p$ is a subclass of $s:Sigma-star$ (denoting the set of all possible sequence of the service being considered) to state that $W_p \subseteq \Sigma^*$. Further, the system behavior $B$ is a subclass of $Sigma^*$. 

5.1.6 Agents’ Local View

Within the SeMF framework the local view of an agent represents the set of all traces that look exactly the same for $P$ after a trace $\omega$ has happened [11]. As in the case of modeling of the agent knowledge, the ASSERT-M ontology describes the association between a SeMF agent and the set of traces believes possible using concepts from the communication and mental-states modules of the ASSERT4SOA top ontology. In general the set of sequences that look exactly the same from some agent’s local view after some trace has happen is described by the following class expression:

Class: $s$:EquivalentTraces-Agent_Trace
EquivalentTo:
  semf:Trace
  and
  (communication:isQualificationIn some
   (communication:Qualificatory_Expression
    and
    (statement:hasSubject some semf:Trace)
    and
    (mental-states:attitude some
     (mental-states:Belief
      and
      (mental-states:held_by some
       mental-states:CognitiveAgent
       and
       roles:count-as some semf:Agent)
     )
    )
   )
  )
)

Example
The set $\lambda_p^{-1}(\lambda_p(\omega))$ of all sequences that look exactly the same from $P$'s local view after the specific sequence $\omega$ has happened is described by the following OWL class expression:
Class: s:EquivalentTraces-P_Omega
EquivalentTo:
  semf:Trace
  and
  (communication:isQualificationIn some
   (communication:Qualificatory_Expression
    and
    (statement:hasSubject value s:omega)
    and
    (mental-states:attitude some
     (mental-states:Belief
      and
      (mental-states:held_by some
       mental-states:CognitiveAgent
       and
       roles:count-as value s:p)
     )
    )
  )
)
)

In the previous class expression s:omega and s:p denote two individuals of type semf:Trace and semf:Agent respectively. Using the class expression s:EquivalentTraces-P_Omega the set \( \lambda_P^{-1}(\lambda_P(\omega)) \cap W_P \) of all sequences of actions \( P \) considers to have possibly happened when \( \omega \) has happened is defined by:

Class: s:PossibleTraces-P_Omega
EquivalentTo:
  s:EquivalentTraces-P_Omega
  and
  s:Wp

5.1.7 System

A SeMF system is a tuple \( S=(\Sigma,P,B,W,V) \) consisting of a set of agents \( P \), a language \( B \) over an alphabet of actions \( \Sigma \), a set \( V \) of agent’s local view, and a set \( W \) of agents’ initial knowledge [11]. The following OWL expression is the description of the class of SeMF systems using concepts from the ASSERT4SOA top ontology:

Class: semf:System
EquivalentTo:
  wsdl:Service
  and (activity:performs some semf:Trace)
  and (mereology:hasConstituency some
   (organisations:OrganisationalRelationship
    and (dependency:hasDependee some semf:Agent))

---

D.3.4 − Second Version Of The ASSERT Ontology 62 / 165
The benefit of the encoding chosen for SeMF system within the ASSERT4SOA ontology is twofold since it allows:

- to retrieve all the five components of the tuple defining a SeMF system (e.g. see below for examples of such kind of queries);
- to compare SeMF system models with ASSERT-E and ASSERT-O service models since like the other models it specialises the concept of Service defined within the wsdl module.

Example

The following class expression describes an instance of SeMF system comprising a client $C$, a secure storage service $S$ and its subcontractor $Sub$ managing persistence of data on behalf of the storage service [11]:

```
Individual: s:SecureStorageSystem
Types: semf:System
Facts:
  mereology:hasConstituency s:ClientServiceRelationship
  mereology:hasConstituency s:ServiceSubcontractorRelationship

Individual: s:ClientServiceRelationship
Types: organisations:OrganisationalRelationship
Facts:
  dependency:hasDepender s:Client
  dependency:hasDependee s:Service

Individual: s:ServiceSubcontractorRelationship
Types: organisations:OrganisationalRelationship
Facts:
  dependency:hasDepender s:Service
  dependency:hasDependee s:Subcontractor

Individual: s:Client
Types: semf:Agent
Facts:
  roles:played-by s:c

Individual: s:Service
Types: semf:Agent
Facts:
  roles:played-by s:s

Individual: s:Subcontractor
Types: semf:Agent
Facts:
  roles:played-by s:sub
```
The following OWL expressions shows sample queries aimed to retrieve the five components defining a SeMF system relying on the definition given above.

The alphabet of actions of secure storage ($\Sigma$):

Class: s:SecureStorageAction
EquivalentTo:
  Wsdl:InterfaceMessageReference
  and
  ((wsdl:isInputOf some s:SecureStorageOperation)
   Or
   (wsdl:isOutputOf some s:SecureStorageOperation)
  )

Class: s:SecureStorageOperation
EquivalentTo:
  wsdl:InterfaceOperation
  and wsdl:parent (wsdl:Interface
   and wsdl:parent value s:SecureStorageSystem)

The set of agents ($P$):

Class: s:SecureStorageAgent
EquivalentTo:
  semf:Agent
  and
  ((roles:isDependerOf some s:SecureStorageSystemRelationship)
   Or
   (roles:isDependeeOf some s:SecureStorageSystemRelationship))

Class: s:SecureStorageSystemRelationship
EquivalentTo:
  organizations:OrganisationalRelationship
  and mereology:isConstituencyOf value s:SecureStorageSystem

The language $B$ over $\Sigma$:

Class: s:SecureStorageTrace
EquivalentTo:
  semf:Traces
  and activity:isPerformedBy value s:SecureStorageSystem

The set $V$ of agents’ local view

Class: s:SecureStorageView
EquivalentTo:
  semf:Trace
  and
  (communication:isQualificationIn some
   (communication:Qualificatory_Expression
    and
    ...)
The set of agents' initial knowledge $W$

Class: $s$: SecureStorageAgentKnowledge

EquivalentTo:

$\text{semf:Trace}$

and

$\text{communication:isQualificationIn some}$

$\text{communication:Qualificatory_Expression}$

and

$\text{mental-states:attitude some}$

$\text{mental-states:Belief}$

and

$\text{mental-states:held_by some s:SecureStorageAgent}$

$5.2$ Representing Basic Properties Definitions

In general a ASSERT-M security property $P$ is a class of SeMF systems (see 5.1.7) fulfilling a specific requirement. Formally the definition of a ASSERT-M security property $P$ takes the following form

$$P \equiv \{ S \in \mathcal{S} | F(S) \}$$

where $F$ is a first order logic (FOL) well-formed formula expressing the requirement of interest and $\mathcal{S}$ is the set of possible SeMF systems.

This section shows how the terms and concepts introduced by the ASSERT4SOA ontology can be used to translate some the security properties possibly referenced in ASSERT-M certificates [11] into equivalent OWL Class expressions using equivalences between OWL axioms and FOL sentences [41]:

- class and property inclusion axioms are equivalent to FOL sentences consisting of an implication between two formulae with the free variable universally quantified at the outer level (e.g. $\forall x. C(x) \supset D(x)$);
- individuals and facts axioms correspond to ground atoms (e.g. $C(a) \land P(a, b)$);
- and axioms correspond to conjunctions of formulae;
- or axioms correspond to disjunction of formulae;
- not axioms correspond to negation of formulae.
5.2.1 Authenticity

The authenticity property [11] expresses the requirement that in a system $S$ (e.g. a client-server system) each time an agent $P$ (e.g. the service provider) observes the occurrence of some type of sequence of actions (e.g. a sequence terminated by the receipt of a login message) a specific action $a$ (e.g. the sending of a login message by a client) actually occurred in the past (i.e. $S$ belongs to the class of client-server systems that guarantee that the service provider observes a login message action by a consumer only if that consumer sent it sometime in the past).

[11] generalizes the concept of authenticity of an action to the case of a set of actions $\Gamma$ and formalizes the requirement as:

$$\forall x. x \in \lambda_P^{-1}(\lambda_P(\omega)) \cap W_P \Rightarrow \text{alph}(x) \cap \Gamma \neq \emptyset$$

The above requirement is the basis to express the authenticity property as class of systems. The authenticity of a set of actions $\Gamma$ for an agent $P$ is defined as the set of systems such that $\Gamma$ is authentic for $P$ each time it observes a trace belonging to $B$ (where $B$ denotes the behavior of system $S$ (see 5.1.7)):

$\text{Authenticity}(P, \Gamma) \equiv \{S | \forall \omega. \omega \in B \Rightarrow (\forall x. x \in \lambda_P^{-1}(\lambda_P(\omega)) \cap W_P \Rightarrow \text{alph}(x) \cap \Gamma \neq \emptyset)\}$

The formula defining the set $\text{Authenticity}(P, \Gamma)$ can be rewritten as:

$$\forall x. \forall \omega. x \in \{t | (\omega \in B) \land (t \in \lambda_P^{-1}(\lambda_P(\omega)) \cap W_P)\} \Rightarrow x \in \{s | \exists y. y \in \text{alph}(s) \cap \Gamma\}$$

The above formula, matching the scheme $\forall x. A(x) \supset B(x)$, is translated by the following OWL expression:

Class: s:PossibleTraces-P_Omega
SubClass:
  s:TracesWith_Gamma

where $s$:PossibleTraces-P_Omega is the identifier of the OWL class describing the set of traces that $P$ considers to have possibly happened when a trace in $B$ has happened (i.e. the set $\{t | (\omega \in B) \land (t \in \lambda_P^{-1}(\lambda_P(\omega)) \cap W_P)\}$) and $s$:TracesWith_Gamma is the identifier of the OWL class describing the set of traces containing at least one action from $\Gamma$ (i.e. $\{s | \exists y. y \in \text{alph}(s) \cap \Gamma\}$).

Example

The requirement that for a system $S$ the sending of a login message by the user $U$ shall be authentic for the service provider $SP$ each time it observes a trace containing a login request is expressed formally by [42]:

$$\forall \omega. (\text{rec_login}, SP, U) \in \text{alph}(\omega) \Rightarrow$$

$$\forall x. x \in \lambda^{SP}_S(\lambda_{SP}(\omega)) \cap W_{SP} \Rightarrow x \in \{s | \exists y. y \in \text{alph}(s) \cap \{(\text{send_login}, U)\}\}$$

as in the general case the above formula can be rewritten as:

$$\forall x. \forall \omega. x \in \{t | (t \in \lambda^{SP}_S(\lambda_{SP}(\omega)) \cap W_{SP}) \land (\text{rec_login}, SP, U) \in \text{alph}(\omega)\} \Rightarrow x \in \{s | \exists y. y \in \text{alph}(s) \cap \{(\text{send_login}, U)\}\}$$

D.3.4 − Second Version Of The ASSERT Ontology 68 / 165
By observing that the transformation $\lambda_\rho \circ \lambda_\rho^{-1}$ preserves characters in $\Sigma/\rho$ and that $\text{rec\_login} \in \Sigma/\rho$ the above formula can be further rewritten as:

$$\forall x. \forall \omega. x \in \{t | t \in \lambda_{\text{SP}}^{-1}(\lambda_{\text{SP}}(\omega)) \cap W_{\text{SP}} \} \cap \{t | \text{rec\_login} \in \text{alph}(t) \} \Rightarrow x \in \{s | \exists y. y \in \text{alph}(s) \cap \{(\text{send\_login}, U)\}\}$$

The above condition is translated by the following OWL expressions:

Class: $s:\text{PossibleTraces-SP\_rec-login}$
EquivalentTo:
- $s:\text{TracesWith\_rec-login}$
  and
- $s:\text{PossibleTraces-SP}$
SubClass:
- $s:\text{TracesWith\_send-login}$

where $s:\text{TracesWith\_rec-login}$ is the OWL expression defining the set of traces containing the receiving of a login message (i.e. $t|\text{rec\_login} \in \text{alph}(\omega)$), $s:\text{PossibleTraces-SP}$ is the OWL expression defining the set of traces that $\text{SP}$ considers possible in $S$ (i.e. $t|t \in \lambda_{\text{SP}}^{-1}(\lambda_{\text{SP}}(\omega)) \cap W_{\text{SP}}$) and $s:\text{TracesWith\_send-login}$ the class of traces containing the sending of a login message (i.e. $s|\exists y. y \in \text{alph}(y) \cap \{(\text{send\_login}, U)\})$.

### 5.2.2 Authenticity with Respect to a Phase

The property of authenticity with respect to a phase strengthens the concept of authenticity by requesting that an authentic action occurs within a phase [11]: for an agent a set of actions is authentic with respect to a phase if it is authentic and they occur within a phase.

**Phase**

A phase is a sequence of actions started by a given symbol and terminated by a given number of terminating symbols [11]. The symbol $\Phi$ denotes the constructor of phase structures. As an example the expression $\Phi(a, \{b(2)\})$ denotes the phase started by the starting symbol $a$ and terminated by the second occurrence of $b$. Note that the definition of phase allows other actions (but $a$) to be interleaved between the terminating symbols.

The above phase can be expressed by means of the following OWL expressions:

Class: $s:\text{TerminatedBy\_b}$
EquivalentTo:
- $(\text{semf:Trace and math:hasContents value s:b})$
  or
- $(\text{semf:Trace and not (math:hasContents some s:Delimiter) and (math:hasNext only s:TerminatedBy\_b)})$

Class: $s:\text{TerminatedBy\_b2}$
EquivalentTo:
(semf:Trace
    and (math:hasContents value s:b)
    and (math:hasNext only s:TerminatedBy_b))
or
(semf:Trace
    and not (math:hasContents some s:Delimiter)
    and (math:hasNext only s:TerminatedBy_b2))
Class: s:Phase_a_b2
EquivalentTo:
  semf:Trace and (math:hasContents value s:a)
  and math:hasNext only s:TerminatedBy_b2

In the above expression the symbol s:Delimiter denotes the OWL class
enumerating the possible starting and terminating symbols used to delimit
phases.
The class s:TerminatedBy_b describes the set of traces containing the value
s:b. It is defined recursively as the traces having s:b as first element or the
traces not having a delimiter (s:Delimiter) as first element and immediately
followed by a trace containing the value s:b.
The class s:TerminatedBy_b2 describes the set of traces having s:b as first
element and terminated by a s:TerminatedBy_b trace or the traces not
starting with a delimiter and terminated by a s:TerminatedBy_b2 trace.
The class s:Phase_a_b2 encodes the phase Φ, 22. It is described as the
traces having s:a as first element and immediately followed by a trace of type
s:TerminatedBy_b2.

**Authenticity with respect to a phase**
The requirement that a set of actions Γ ⊂ Σ is authentic for agent P ∈ ℙ after a
sequence of actions x ∈ B with respect to W_p and λ_p and a phase Φ is formalized
by [11]:

\[
\text{Authenticity}(P, \Gamma, x) \land \forall y \in \lambda^{-1}_p(\lambda_p(x)) \cap W_p \Rightarrow \exists u, v, w. (y = uvw) \land (v \in \Phi) \land (\alpha\phi(v) \cap \Gamma \neq \emptyset)
\]

The formula above can be rewritten as:

\[
\text{Authenticity}(P, \Gamma, x) \land (\forall y \in \lambda^{-1}_p(\lambda_p(x)) \cap W_p \Rightarrow y \in \{s | \exists u, v, w. (s = uvw) \land (v \in \Phi) \land (\alpha\phi(v) \cap \Gamma \neq \emptyset))\})
\]

The formula above is translated by the following OWL expressions:

Class: s:AuthenticWrt-Phase_Gamma_P_Omega
EquivalentTo:
  s:Authentic-Gamma_P_Omega
  and s:PossibleTraces-P_Omega

Class: s:PossibleTraces-P_Omega
SubClassOf: s:PhaseWith_Gamma_Terminated
Class: s:PhaseWith_Gamma_Terminated
EquivalentTo:
  (semf:Trace and not (math:hasContents some s:Delimiter)
   and math:hasNext only s:PhaseWith_Gamma)
  or
  (semf:Trace and not (math:hasContents some s:Delimiter)
   and math:hasNext only s:PhaseWith_Gamma_Terminated)

Class: s:PhaseWith_Gamma
EquivalentTo:
  s:Phase and s:TracesWith_Gamma

where s:Authentic-Gamma_P_Omega is the identifier of the set of actions
authentic for P after x (i.e. Authenticity(P, Γ, x)), s:PossibleTraces-P_Omega
the set of traces P considers to have possibly happened when x has happened
(i.e. λ⁻¹(λ_p(ω)) ∩ W_p), s:PhaseWith_Gamma_Terminated encodes the set
{s|∃u,v,w.s = uvw} containing the traces started by a sequence of non delimiters
and immediately followed by a phase containing some action from Γ. The set of
phases containing some action from Γ is defined as the intersection of the set of
phases s:Phase (i.e. φ) and s:TracesWith_Gamma the set of traces
containing some action from Γ (i.e. {s|alph(s) ∩ Γ ≠ ∅}).

Example
The requirement, taken from [42], that service requests from a user U should
only be answered when they have been sent during a current session which has
been started with a login and has not yet terminated is formalized stating that
action (send-request, U) must be authentic for service provider SP with respect
to the phase started by (rec-login, SP, U) action and terminated by a (rec-
logoff, SP, U) action (i.e. the phase denoted by φ(rec_login, rec_logoff(1))):

Authenticity(SP, Γ, x)
∧ (∀y.y ∈ λ⁻¹(λ_p(x))) ∩ W_SP ⇒ ∃u,v,w.(y = uvw)∧(v ∈ V ∧ (alph(v) ∧ Γ ≠ ∅))
where Γ = ((send_request, U), V = φ(rec_login, rec_logoff, rec_logoff(1))). As
in general case the above formula can be rewritten as:

Authenticity(SP, Γ, x)
∧ (∀y.y ∈ λ⁻¹(λ_p(x))) ∩ W_SP ⇒ y ∈ {s|∃u,v,w.(s = uvw) ∧ (v ∈ V) ∧ (alph(v) ∧ Γ ≠ ∅))

The above condition is translated by the following OWL expressions:

Class: s:AuthenticWrt-V_Gamma_SP_X
EquivalentTo:
  s:Authentic-Gamma_SP_X
  and s:PossibleTraces-SP_X

Class: s:PossibleTraces-SP_X
SubClassOf: s:TerminatedBy_PhaseWith_Gamma

Class: s:TerminatedBy_PhaseWith_Gamma
EquivalentTo:
(smf:Trace and not (math:hasContents some s:Delimiter) 
and math:hasNext only s:PhaseWith_Gamma) 
or
(smf:Trace and not (math:hasContents some s:Delimiter) 
and math:hasNext only s:TerminatedBy_PhaseWith_Gamma)

Class: s:PhaseWith_Gamma

EquivalentTo:
s:V and s:TracesWith_Gamma

where s:Authentic-Gamma_SP_X is the OWL expression defining the set of traces that are authentic for SP after x (i.e. the OWL expression encoding for Authenticity(SP, Γ, x) as shown in 5.2.1), is the set of traces that SP considers possible in S (i.e. \( \lambda_{sp}^{\gamma}(\lambda_{sp}(\omega)) \cap W_{sp} \)), s:TerminatedBy_PhaseWith_Gamma are the traces having a phase V including an action from Γ (i.e. \( \{s|\exists u,v,w.(s = u w) \land (v \in V) \land (\text{alp}(v) \cap \Gamma \neq \emptyset)\})

The set of traces belonging to phase V = \( \Phi((\text{rec_login,SP, U}),(\text{rec_logoff,SP, U}))(1) \) and the set of traces containing an action in Γ = \( \{(\text{send_request, U})\} \) are described in OWL using the approach presented in sections 5.2.2 and 5.1.3 respectively.

5.2.3 Confidentiality

The parameter confidentiality [11] property expresses the requirement that in a system an agent that monitors a sequence of actions cannot distinguish between possible values (e.g. \( v_1,v_2 \)) of a certain parameter (e.g. password) of a specific action (e.g. send_login) or set of actions. In other words the requirement states that an agent R shall not be able to discard any of the possible values \( M \) for a certain parameter \( \text{par} \) of a set of actions.

The formalization is given by stating that the possible values for parameter \( \text{par} \) that R believes possible (after observing \( \omega \)) is a superset of the set of possible values for parameter \( \text{par} \).

The SeMF framework expresses parameter confidentiality in terms of sequences of \((\text{action type, parameter value})\) pairs resulting from the mapping of traces of the system.

Expressing the requirement of parameter confidentiality requires the ability to express dependencies between parameter values occurring within sequences of actions [11]. The notion of \((L,M)\)-completeness is aimed to capture the dependencies allowed to be known by agents that shall not know the parameter value within a set of sequences of actions [11]. An \((L,M)\)-complete set is the set of all possible sequences of \((\text{action type}, \text{parameter value})\) pairs matching sequence patterns \( L \) and having \text{parameter value} from set \( M \) [11].

The requirement that a set of parameters \( M \) is parameter confidential for agent \( R \in \mathbb{P} \) with respect to a \((L,M)\)-complete set \( K \) is formalized by:

\[
\forall \omega. \omega \in B \supset P_{\delta}(\omega) \cap K \subseteq Q_{\delta}(\omega)
\]
where $P_R(\omega)$ denotes the set all of possible sequences of pairs resulting from the mapping of the set of traces that $R$ considers to have possibly happened when $\omega$ has happened, $Q_R(\omega)$ denotes the set of sequence of pairs mapping the set of traces that $R$ considers to have possibly happened when $\omega$ has happened. The intersection is aimed to restrict the set of all of possible sequences of pairs to those satisfying the dependencies between parameter values allowed to be known. The FOL sentences defining $P_R(\omega)$ and $Q_R(\omega)$ are given in [11].

The confidentiality property of a set of parameters $M$ for an agent $R \in \mathcal{P}$ is then defined as the set of systems such that $M$ is parameter confidential for agent $R \in \mathcal{P}$ with respect to the $(L,M)$-complete set $K$:

$$Confidentiality(K,R) \equiv \{ S | \forall \omega. \omega \in B(S) \Rightarrow P_R(\omega) \cap K \subseteq Q_R(\omega) \}$$

The above formula is translated by the following OWL expressions:

\begin{verbatim}
Class: s:ParameterConfidentiality_K-R
EquivalentTo:
  s:PossibleSeqOfPairs_R and s:K
SubclassOf:
  s:BelievedSeqOfPairs_R
\end{verbatim}

where $s$:PossibleSeqOfPairs_R is the set all of possible pairs resulting from the mapping of traces (i.e. $P_R(\omega)$), $s$:K is the set of all sequences of pairs satisfying dependencies between parameters values, and $s$:BelievedSeqOfPairs_R is the set of sequence of pairs mapping the set of traces that $R$ considers to have possibly happened when $\omega$ has happened (i.e. $Q_R(\omega)$)

**Example**

The requirement that a subcontractor $Sub$ managing persistence of data on behalf of a secure storage service $S$ shall not be able to read the data and to distinguish between different data being stored [11] is expressed by the following OWL expressions:

\begin{verbatim}
Class: s:ParameterConfidentiality_MultipleStore-G-Sub
EquivalentTo:
  s:PossibleSeqOfPairs_Sub and s:MultipleStore-G_Complete
SubclassOf:
  s:BelievedSeqOfPairs_Sub

Class: s:PossibleSeqOfPairs_Sub
EquivalentTo:
  (Math:Sequence
   and (math:hasContents only (math:first value s:store_Sub
    and math:second only s:G))
   or
   (Math:Sequence
    and (math:hasContents only (math:first value s:store_Sub
     and math:second only s:G))
    and math:hasNext only s:PossibleSeqOfPairs_Sub)
\end{verbatim}

D.3.4 − Second Version Of The ASSERT Ontology 71 / 165
Class: s:K
EquivalentTo:
(Math:Sequence
 and (math:hasContents only (math:first value s:store_Sub
 and math:second only s:G)))
or
(Math:Sequence
 and (math:hasContents only (math:first value s:store_Sub
 and math:second only s:G))
 and math:hasNext only s:K)

Class: s:BelivedSeqOfPairs_Sub-Omega
EquivalentTo:
math:Sequence
and (math:hasContents only (math:first value s:store_Sub
 and math:second only s:G))
and math:hasNext only (
math:Sequence
and (math:hasContents only
 (math:first value s:store_Sub
 and math:second only s:G))
)

Class: s:G
EquivalentTo: {s:gd1, s:gd2}

where s: s:PossibleSeqOfPairs_Sub is the set of all possible sequence of
(action type, parameter value) pairs resulting from the mapping on Sub,
s:MultipleStore-G_Complete is the set of all sequences of traces
containing multiple store operations, s:BelivedSeqOfPairs_Sub is the set of
sequences of pairs mapping the set of traces that Sub considers to have possibly
happened when $\omega$ has happened.

5.2.4 Enforcing system behavior

The *enforce-behaviour* property expresses the requirement that the set of
possible sequences of actions of a system $S$ must be a subset of a specific set $L$
and is formalised as [11]:

$$B \subseteq L$$

where $B$ is the set of actual traces of $S$ and $L$ is the particular required
behaviour. The *enforce-behaviour* security property is defined as the set of
systems such that the behaviour of $S$ is a subset of $L$:

$$Enforce\_Behaviour(L) \equiv \{S | B(S) \subseteq L\}$$

where $B(S)$ denotes the behaviour of system $S$. The above formula is translated
by the following OWL expression:
Class: s:B
SubClass: s:L

where $s:B$ is the identifier of the OWL class expression describing the actual behaviour of system $S$ and $s:L$ is the name of the class describing the requested behaviour.
Chapter 6

Semantic Similarity

In this section we explore the problem of comparing two Ontology-based certificates, and in particular to define a metric and a measurement algorithm to express their similarity, and in general to compare concepts of an OWL ontology. The comparison is based on the definitions of ASSERT-O concepts as expressed in the ASSERT4SOA ontology.

6.1 Background

In literature, there are different approaches to the problem of assessing the similarity of concepts in two ontologies. A similarity $\sigma: C \times C \to [0, 1]$ is a function from a pair of concepts to a real number between zero and one expressing the degree of similarity between two concepts such that:

11. $\forall x \in C, \sigma(x, x) = 1$
12. $\forall x, y \in C, \sigma(x, y) = \sigma(y, x)$

Considering the ontology as a graph where concepts are nodes, and IS-A relationships are edges, the first attempt to assess the similarity of two concepts is counting the edges of the shortest path between the two concepts. The similarity would thus be the inverse of the edge-count.

The problem in this approach lies in the fact that in ontologies (and taxonomies in general), there is a wide variability in the “distance” covered by a single taxonomic link (non-uniform density of the taxonomy), so the edge count cannot be an accurate metric on its own.

A first evolution of these methods can be found in [44] [43], where the notions of weighted edges and *information content* were introduced.

The idea is to give weights to concepts based on their depth in the ontology with the following formula:

$$\omega(n) = \frac{1}{k(n)+1}$$
where \( l(n) \) is the length of the longest path from the root concept to node \( n \) in the taxonomy and \( k \) is a predefined factor larger than 1 indicating the rate at which the weight values decrease along the taxonomy (usually \( k \) is set to 2).

The similarity is then computed as \( \sigma(c_1, c_2) = 1 - \delta(c_1, c_2) \), being \( \delta(c_1, c_2) \) the distance between the concepts. The distance is computed as:

\[
\delta(c_1, c_2) = \left( \omega(ccp(c_1, c_2)) - \omega(c_1) \right) + \left( \omega(ccp(c_1, c_2)) - \omega(c_2) \right)
\]

Where \( ccp(x,y) \) is the sum of the distances to their closest common parent.

An alternative to this approach is based on the information content. It is computed associating probabilities of encountering each concept in the ontology, allowing to capture the same idea of edge distance, but avoiding its unreliability. This implies that the probability grows as one moves up the taxonomy: if \( c_1 \) is-a \( c_2 \) then \( p(c_1) \leq p(c_2) \)

Moreover, if the taxonomy has a unique top node then its probability is 1. Given this assignment of probabilities, the information content of a concept \( c \) can be quantified as negative the log likelihood, i.e. \( -\log p(c) \). The similarity is then computed as:

\[
\sigma(c_1, c_2) = \max_{c \in S(c_1, c_2)} \left[ -\log p(c) \right]
\]

where \( S(c_1, c_2) \) is the set of concepts that subsumes both \( c_1 \) and \( c_2 \).

These approaches count on the fact that most ontologies build upon primitive concepts, and focus on IS-A relationships to link concepts. OWL ontologies, and the ASSERT4SOA Ontology in particular, define concepts that also have a number of datatype properties and a number of object properties. These properties somehow determine the characteristics of the concept, therefore these properties can be seen as features of the concept.

More common features and less non-common features two concepts have, the more similar they are. When a concept is a subclass of another concept, it inherits all the object and datatype properties of the superclasses. If they are not explicitly declared, a reasoner can infer them.

In [43] we can find a reference to an approach that is feature-based. The approach is originally proposed in [45], but can easily applied to the ontology domain.

The following is a semantic similarity measure which takes into account the feature sets of concepts. If \( F(c) \) is the set of features of a concept \( c \), the similarity of two concepts \( c_1 \) and \( c_2 \) can be expressed as:

\[
\sigma(c_1, c_2) = \frac{2 \times |F(c_1) \cap F(c_2)|}{|F(c_1) \cap F(c_2)| + |F(c_1) \cup F(c_2)|}
\]

In this case the similarity measures the degree of overlap in the set of features of the compared concepts.
The success of this measure depends on the degree to which the features of concepts are specified in ontologies. In most of the current available ontologies only the subsumption relation between concepts is specified and the other relations are neglected. In these kinds of ontologies, the feature-based similarity measure cannot be a useful measure and the results are often not good enough. The ASSERT4SOA Ontology, on the contrary, is suitable for this method, and in the following paragraph we propose an extension elaborated for the purposes of the project.

6.2 Adopted approach

Concepts in the ASSERT4SOA Ontology are described by means of their object properties. Among the possibilities described above, a feature-based approach can be a good candidate for comparing ASSERT-Os. For the purposes of the problem of assessing the degree of similarity of ASSERT-Os, the feature-based method will be extended.

A first consideration that needs to be made is that in the case of ontologies two classes can share an object property, but still be different, because the property points to different concepts.

Consider two classes characterized only by these assertions:

- Class A hasObjProp some C
- Class B hasObjProp some D

The presence of the the same object property does not say much on the similarity of A and B, but we can say that the more C and D are similar, the more A and B are as well.

The hierarchy of the concepts cannot be ignored in the computation of the similarity. For this reason the direct superclasses comparison is used, so that part of the similarity measure depends on the SubclassOf statement. For example if:

- Class A subclassOf E
- Class B subclassOf F

Part of the similarity measure is given by the similarity of E and F. The choice in this version of the algorithm is to balance equally the impact of the
similarity of superclasses and the similarity of features, but this is a choice that can be further elaborated. A discussion on this point can be found in section 6.4.

Another aspect to take into account is that two features generally do not have the same impact on the assessment of the similarity between concepts. For each pair of concepts, the properties are examined and weighted by an expert in the domain of the ontology. If the properties link to the same instance of the feature, then it is said that there is a match, otherwise the distance between the instances of the pointed concepts is computed.

Let us consider two concepts \( c_1 \) and \( c_2 \) and name object properties as \( p_i \) and the range of \( p_i \) for \( c_j \) as \( R_{i,c_j} \) (at the moment we do not consider the multiplicity of the object properties).

Given two concepts \( c_1 \) and \( c_2 \), the similarity \( \sigma(c_1, c_2) \) is computed as follows:

Let us call \( S_i \) the direct superclass of \( c_j \). The direct superclass can be either declared or inferred by the reasoner. Note that the reasoner is able to infer more refined hierarchies, thus giving the opportunity to assess a more accurate similarity. Since OWL allows multiple inheritance, if more than one class is declared as superclass, the closest pair of superclasses is chosen.

Let \( P \) be the set of object properties of both concepts. A weight is assigned to each \( p_i \in P \) such that:

\[
\sum_{p_i \in P} w(p_i) = 1
\]

Then, we define the similarity of \( c_1 \) or \( c_2 \) as:

\[
\sigma(c_1, c_2) = 0.5\sigma(S_1, S_2) + 0.5 \frac{2 \times \sum_{p_i \in P_1 \cap P_2} (w(p_i) \cdot \sigma(R_{i,c_1}, R_{i,c_2}))}{\sum_{p_i \in P_1 \cap P_2} (w(p_i) \cdot \sigma(R_{i,c_1}, R_{i,c_2})) + 1}
\]

The first part of the addition is the part related on the similarity of superclasses, the second part is related to the similarity of the features. If \( c_1 \) and \( c_2 \) are equivalent, even without object properties, then \( \sigma(c_1, c_2) = 1 \).

### 6.3 Comparing ASSERT-Os

In this section the general formula presented above is explained through an example of comparison of two ASSERT-Os. We can see the structure of ASSERT-Os in Figure 28, where arrows are object properties and boxes are classes:\(^4\)

Being ASSERT-Os members of the same superclass, they share all features of the superclass.

---

\(^4\)If not specified otherwise "\( A \op \rightarrow B \)" is the equivalent of the OWL assertion "\( A \op \text{some } B \)."
As depicted in Figure 28, an ASSERT-O is characterized by an expiration date, and a conformance expression through the `assert-o:expirationDate` and `communication:isCarrierFor` object properties. To compare the asserts, for each of those properties, the related concepts need to be examined as well. In the case of expiration date, the `time:Instant` concepts pointed by the property are compared, and similarly the `assert-o:OntologySupportedConformanceExpression`.

In the following we give an example on how to compare two ASSERT-Os present in the ASSERT4SOA Ontology.
In this example we compare ASSERT isticom:ID_5fd0dab4-18a0-4e7e-9389-524d1f720b8c (A1 in the following) with isticom:ID_e88279bf-4f5b-4064-848c-9d13ee4dc871 (A2) that can be found in the second version of the ASSERT4SOA Ontology. The two corresponding OWL classes are depicted in Figure 29.

As shown by the figure, the ASSERTs share the same object properties. In particular they have the same expiration date and differ in the Conformance Expression pointed by the communication:isCarrierFor object property.

To compute the similarity, we denote $p_1= \text{assert-o:expirationDate}$ and $p_2= \text{communication:isCarrierFor}$, and assign the weights as follows:

- $w(p_1) = 0.3$
- $w(p_2) = 0.7$

The similarity is then:

$$
\sigma(A_1, A_2) = 0.5 + \frac{0.3 \cdot 1 + 0.7 \sigma(CE_1, CE_2)}{0.3 \cdot 1 + 0.7 \sigma(CE_1, CE_2)} + 1
$$

Where CE1 and CE2 are the two conformance expressions. As it can be seen from the formula above, the similarity of the ASSERTs depends on the similarity of Conformance Expressions. Figure 30 depicts the structure of the two conformance expressions.
The two conformance expressions have the same list of object properties:

- \( p1: \text{assert-o:conformsTo} \rightarrow w1=0.3 \)
- \( p2: \text{communication:hasSubject} \rightarrow w2=0.3 \)
- \( p3-p4: \text{mental-states:attitude} \rightarrow w3, w4=0.2 \)

We also know that the two endpoints are not further specified, and nothing can be said except they are both subclasses of \( \text{wsdl:Endpoint} \). Moreover, the reasoner inferred that \( \text{assert-o_1:MarkScottAssertion} \) is declared to be a \( \text{assert-o:ConformanceExpression} \), the reasoner is able to infer that it actually is an \( \text{assert-o:OntologySupportedConformanceExpression} \).

For conciseness, the computation of \( \sigma(\text{secure-pipe:EncryptingSistem, authentication-enforcer:AuthenticatingSystem}) \) and \( \sigma(\text{assert-o_1:JonhSmithOntologySupportedBelief, assert-o_1:JonhSmithOntologySupportedBelief}) \) are omitted. They are, respectively:

- \( \sigma(\text{EncryptingSistem, AuthenticatingSystem}) = 0.5 \)
- \( \sigma(\text{JSOntologySupportedBelief1, JOSntologySupportedBelief2}) = 0.98 \)

The similarity of the conformance expressions is thus

\[
\sigma(CE_1, CE_2) = 0.5 + \frac{(0.3 \cdot 0.5 + 0.2 \cdot 0.98 + 0.2 \cdot 1 + 0.3 \cdot 0.5)}{(0.3 \cdot 0.5 + 0.2 \cdot 0.98 + 0.2 \cdot 1 + 0.3 \cdot 0.5) + 1} = 0.91
\]

And consequently the similarity of the two asserts is:
Considerations on the chosen approach

The algorithm presented in this section is aimed at giving a possible ordering of ASSERT-Os based on their similarity.

This approach is representative of the intuition of what could be the similarity between ASSERTs. In fact, the introduction of weights over object properties allows balancing attributes and giving higher results to ASSERTs with similar conformance expressions than expiration date. It is worth noting that going down the features tree, all relevant characteristics of an ASSERT are present, including the service, the security property and the organization issuing the ASSERT. The domain expert can balance the weights to emphasize the aspects she is more interested in, so to calibrate the algorithm and get more accurate results.

The measurement method presented in this section is a first attempt to formalize similarity of OWL classes, taking as a basis intuitions from the literature, and has several dimensions of improvement. To make it more accurate, more aspects of OWL Ontologies can be taken into consideration. One dimension of evolution can be to consider the hierarchy of Object Properties, and their multiplicity.

One other dimension would be to use weighs representing the depth of concepts presented in section 6.1 to refine the impact of the similarity of superclasses, that now it is set to 0.5. The idea would be that going down the ontology graph, the similarity of superclasses have a greater impact. For example, if C1 and C2 are both subclasses of Document, the impact of this information should be lower than if they inherit from more detailed classes, such as ASSERT.
Chapter 7

Conclusions

7.1 Conclusions

In this document we described the second and final version of the ASSERT4SOA Ontology, supporting the description of ASSERT-E, ASSERT-O and ASSERT-M security properties and enabling the interoperability and comparison of heterogeneous certificates in the ASSERT4SOA Framework. This second version is an update and extension of the previous version, that also considers the feedbacks on the evaluation task (Task 3.3 of Workpackage 3)).

This document is meant to be self contained, so it gives a complete description of the ontology, but it highlights the extensions and the differences with respect to the first version.

The more relevant additions in the ontology presented in this deliverable are modules related to WS security standards, Topological spaces and entities representing concepts of the ASSERT4SOA Core part of the language. The ASSERT-E and ASSERT-M modules have been revised, and it was presented how the models of the systems in these types of ASSERTs is represented in OWL, by means of concepts coming from the Top Ontology.

Some considerations on the possible mappings between representation of the models in ASSERT-E service models (assert-e:ServiceModel) and ASSERT-M system models (semf:System) are also reported.

In the document we also presented a proposal for an algorithm meant to measure the similarity between OWL concepts, created by adapting some proposals from the literature to best suit the peculiarities of the ASSERT4SOA Ontology. A discussion on the applications of the algorithm to assess the similarity between ASSERT-Os is also reported.

Finally the appendix gives a detailed description of the content of each module in the ASSERT4SOA Ontology.
Appendix A

Modules

A.1. Modules and dependencies

The following figure shows the dependencies existing between ASSERT4SOA Top Ontology modules. In the figure each package represents a different OWL file of the ASSERT4SOA Ontology.

![Dependencies between ASSERT4SOA top Ontology modules](image)

**Figure 31** Dependencies between ASSERT4SOA Top Ontology modules
All modules included in the ASSERT4SOA Ontology are described in details in the following sections.

A.1.1. Statement

<table>
<thead>
<tr>
<th>Direct Imports</th>
</tr>
</thead>
<tbody>
<tr>
<td>math</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Indirect Imports</th>
</tr>
</thead>
<tbody>
<tr>
<td>mereology</td>
</tr>
</tbody>
</table>

Short Description

This module contains the vocabulary to describe object properties as OWL entities applying the Statement ontology pattern (see section 2.1.3). The vocabulary is borrowed from the RDF built-in vocabulary intended for describing RDF statements [19].

Classes

**Class statement:Statement**

<table>
<thead>
<tr>
<th>Subclass of:</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Represent an owl:ObjectProperty as an OWL:Class</td>
</tr>
<tr>
<td>Examples:</td>
<td>The class Property in the mental-states module is an example of Statement.</td>
</tr>
<tr>
<td>Similar to:</td>
<td>rdf:Statement</td>
</tr>
</tbody>
</table>

Object Properties

**ObjectProperty statement:hasObject**

<table>
<thead>
<tr>
<th>Subproperty of:</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>The object of the statement</td>
</tr>
<tr>
<td>Similar to:</td>
<td>rdf:object</td>
</tr>
</tbody>
</table>

**ObjectProperty statement:hasSubject**

<table>
<thead>
<tr>
<th>Subproperty of:</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>The subject of the statement</td>
</tr>
<tr>
<td>Examples:</td>
<td>Properties are statements about persistent items.</td>
</tr>
<tr>
<td>Similar to:</td>
<td>rdf:subject</td>
</tr>
</tbody>
</table>
A.1.2. Mereology

<table>
<thead>
<tr>
<th>Direct Imports</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Indirect Imports</td>
<td></td>
</tr>
</tbody>
</table>

**Short Description**
This module contains a vocabulary of different types of composition (meronymic) relationships. This vocabulary is based on [46]. This vocabulary presents six kinds of composition based on particular combinations of three basic properties:

- **Configuration** - whether or not the parts bear a particular functional or structural relationship to one another or to the object they constitute
- **Homeomerous** - whether or not the parts are the same kind of thing as the whole
- **Invariance** - whether or not the parts can be separated from the whole

**ObjectProperties**

**ObjectProperty mereology:hasComponent**

<table>
<thead>
<tr>
<th>Subproperty of:</th>
<th>mereology:hasPart</th>
</tr>
</thead>
<tbody>
<tr>
<td>InverseProperty</td>
<td>mereology:isComponentOf</td>
</tr>
<tr>
<td>Description</td>
<td>Defines a non-transitive configuration of parts within a whole.</td>
</tr>
<tr>
<td>Examples:</td>
<td>-</td>
</tr>
</tbody>
</table>

**ObjectProperty mereology:hasConstituency**

<table>
<thead>
<tr>
<th>Subproperty of:</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>InverseProperty</td>
<td>mereology:isConstituencyOf</td>
</tr>
<tr>
<td>Description</td>
<td>Defines an invariant configuration of parts within a whole.</td>
</tr>
<tr>
<td>Examples:</td>
<td>A cappuccino is partly milk. The clay is a constituent of a statue.</td>
</tr>
<tr>
<td>Similar to:</td>
<td>LKIF:</td>
</tr>
</tbody>
</table>

**ObjectProperty mereology:hasLocation**

<table>
<thead>
<tr>
<th>Subproperty of:</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>InverseProperty</td>
<td>mereology:isLocationOf</td>
</tr>
<tr>
<td>Description</td>
<td>Defines a homeomorphic and invariant configuration of parts within a whole.</td>
</tr>
<tr>
<td>Examples:</td>
<td>San Francisco is part of California. A peak is part of a mountain. The 50-yard line is part of a football field</td>
</tr>
<tr>
<td>Similar to:</td>
<td>LKIF</td>
</tr>
</tbody>
</table>
ObjectProperty mereology:hasMember

Subproperty of: -
InverseProperty mereology:isMemberOf
Description A relation between collections and entities.
Examples: An employee is part of a union
My collection of saxophones includes an old Adolphe Sax original alto.

ObjectProperty mereology:hasPart

Subproperty of: -
InverseProperty mereology:isPartOf
Description Defines a configuration of parts within a whole.
Examples: Wheels are part of a car
Scenes are parts of films

ObjectProperty mereology:hasPartner

Subproperty of: mereology:hasMember
InverseProperty mereology:isPartnerOf
Description Defines an invariant collection of parts as a whole.
Examples: Ginger and Fred are a waltz couple
Stan Laurel is part of Laurel and Hardy.

ObjectProperty mereology:isPortionOf

Subproperty of: -
Description Defines a homeomorphic configuration of parts within a whole.
Examples: A slice of bread is a portion of a loaf of bread.
A meter is part of a kilometre.

A.1.3. Dependency

<table>
<thead>
<tr>
<th>Direct Imports</th>
</tr>
</thead>
<tbody>
<tr>
<td>mereology</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Indirect Imports</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Short Description
This module contains a vocabulary for dependencies as defined in the i* model [47].
Classes

Class dependency:Dependency

Subclass of: -
Description “The intuitive meaning of a dependency is that a depender by depending on someone else (the dependee) for something (the dependum) can accomplish some goal or objective that it would otherwise be unable to achieve (or not as well).” [47]
Examples: Certification Authorities depend on Evaluation Bodies for appraisal.

Class dependency:System

Subclass of: -
Description A collection of dependency relationships among actors.
Examples: A Software Certification domain model presenting the relationships among service designers, certification authorities, evaluation bodies and service providers
Similar to: Strategic Dependency model in [47].

ObjectProperties

ObjectProperty dependency:hasDepender

Subproperty of: -
Description The depender in a Dependency
Examples: Certification Authorities in “Certification Authorities depend on Evaluation Bodies for appraisal.”

ObjectProperty dependency:hasDependeee

Subproperty of: -
Description The dependee in a Dependency
Examples: Evaluation Bodies in “Certification Authorities depend on Evaluation Bodies for appraisal.”

ObjectProperty dependency:hasDependum

Subproperty of: -
Description The dependum in a Dependency
Examples: Appraisal in “Certification Authorities depend on Evaluation Bodies for appraisal.”
A.1.4. Time

**Direct Imports**

- math

**Indirect Imports**

- mereology

**Short Description**

This module models temporal concepts and implements the relations between times interval considered in the theory of time by Allen [48]. As related work, the following ontologies modelling time related concepts have been investigated: LKIF, OWL-Time [49] and CIDOC [16].

**Classes**

**Class** time:BC-AD_Transition

**Subclass of:** time:Instant

**Description**

Time instant indicating the transition from the BC (Before Christ) era to the AD (Anno Domini) era. It is based on the traditionally reckoned year of the conception or birth of Jesus of Nazareth, with AD counting years after the start of this epoch, and BC denoting years before the start of the epoch. Indeed, year zero is only a convention, because in this scheme the year AD 1 immediately follows the year 1 BC.


**Class** time:BigBangTime

**Subclass of:** time:Instant

**Description**

Time instant denoting the beginning of the development of the Universe according to the Big Bang theory.

**Class** time:Calendar

**Subclass of:**

**Description**

A system of organizing units of time for the purpose of reckoning time over extended periods [50]. Calendars are usually based on astronomical cycles.

**Examples:** Examples of calendars are the Julian calendar, the Gregorian calendar, the Chinese calendar and so on.

**Class** time:DateTimeValue

**Subclass of:**

**Description**

A reference to a particular instant of time represented within a calendar system.

**Examples:** The first instant of 2011 in the Gregorian calendar system is
related to the “1 January 2011 at 00:00” value.

Similar to: OWL-Time: DateTimeDescription

Class time:Epoch

Subclass of: time:Instant
Description Time instant indicating the day whose date according to the Gregorian Calendar is 1st January 1970 at 00:00.

Class time:Instant

Subclass of: time:TemporalEntity
Description Point-like (i.e. without extent) temporal entity or, alternatively, a temporal entity whose start and end coincide.
Similar to: LKIF: time:Moment
OWL-Time: Instant

Class time:ProperTimeInterval

Subclass of: time:TimeInterval
Description A time interval having a start and an end.
Examples: Year 2011 is a time interval with a start (“1 January 2011 at 00:00”) and an end (“31 December 2011 at 23:59”).
Similar to: OWL-Time: ProperInterval

Class time:SolarCalendar

Subclass of: time:Calendar
Description A calendar whose dates indicate the position of the earth on its revolution around the sun.
Examples: The Julian calendar, the Gregorian calendar and the Coptic calendar are solar calendars.

Class time:TemporalEntity

Subclass of:
Description Entity representing all time related concepts.
Similar to: OWL-Time: TemporalEntity

Class time:TimeInterval

Subclass of: time:TemporalEntity
Description Temporal Entity with an extent, i.e. a temporal entity whose start and end are different (and start<end). A time interval must have a start but can have no end, i.e. time intervals can be infinite.
Examples: The period corresponding to the age of the universe has a
start (approximately 15 billions of years ago) and, at the moment, no end.

Similar to: LKIF: time:Temporal_Occurrence
OWL-Time: Interval

Object Properties

Object Property time:after

Subproperty of: math:follows
InverseProperty time:before
Description Property indicating that a temporal entity occurs after another temporal entity.
Examples: Let to be the instant when a person P is born and ti a temporal entity related to another instant or period during P’s life; it always holds that ti after to.
Similar to: OWL-Time: after

Object Property time:before

Subproperty of: math:precedes
InverseProperty time:after
Description Property indicating that a temporal entity occurs after another temporal entity.
Examples: Let ti be the instant when a person P dies and ti a temporal entity related to another instant or period during P’s life; it always holds that ti before ti.
Similar to: OWL-Time: before

Object Property time:endsAt

Subproperty of: -
Description Property that relates a temporal entity to its start
Examples: Let T represent year 2011 and t2 the instant with date time “31 December 2011 at 23:59”; T endsAt t2.
Similar to: OWL-Time: hasEnd

Object Property time:hasDateTime

Subproperty of: -
Description Property relating an instant of time to a date and a time that uniquely identify that specific instant in a calendar system.
Examples: If t1 is the first instant of 2011 in the Gregorian calendar system, we can say that t1 hasDateTime “1 January 2011 at 00:00”.
Similar to: OWL-Time: hasDateTimeDescription
Object Property time:inside

*Subproperty of:* -

*Description:* Property relating an (extended) time interval to one of its interior points (instants).

*Examples:* Let $T$ represent year 2011, $t_1$ the instant with date time “15 June 2011 at 17:02” and $t_2$ the instant with date time “4 February 2012 at 20:34”; it is true that $t_1$ inside $T$ while $t_2$ inside $T$ is false.

*Similar to:* OWL-Time: inside

Object Property time:intervalAfter

*Subproperty of:* time:after

*InverseProperty:* time:intervalBefore

*Description:* Property representing the inverse of the X before Y Allen’s relation (>) [48].

*Examples:* Let $T_1$ represent year 2010 and $T_2$ denote year 2012; $T_2$ intervalAfter $T_1$.

*Similar to:* LKIF: time:after
OWL-Time: intervalAfter
CIDOC: occurs before (occurs after) (P120)

Object Property time:intervalBefore

*Subproperty of:* time:before

*InverseProperty:* time:intervalAfter

*Description:* Property representing the X before Y Allen’s relation (<) [48].

*Examples:* Let $T_1$ represent year 2010 and $T_2$ denote year 2012; $T_1$ intervalBefore $T_2$.

*Similar to:* LKIF: time:before, time:preceeds
OWL-Time: intervalBefore
CIDOC: occurs before (occurs after) (P120)

Object Property time:intervalContains

*Subproperty of:* time:intervalOverlappedBy

*InverseProperty:* time:intervalDuring

*Description:* Property representing the inverse of the X during Y Allen’s relation (di) [48].

*Examples:* Let $T_1$ represent year 2011 and $T_2$ denote June 2011; $T_1$ intervalContains $T_2$.

*Similar to:* OWL-Time: intervalOverlappedBy
CIDOC: occurs during (includes) (P117)
**Object Property time:intervalDuring**

*Subproperty of:* time:intervalOverlaps  
*InverseProperty:* time:intervalContains  
*Description:* Property representing the \( X \) during \( Y \) Allen’s relation (d) \[48\].  
*Examples:* Let \( T_1 \) represent year 2011 and \( T_2 \) denote June 2011; \( T_2 \) intervalDuring \( T_1 \)  
*Similar to:* LKIF: time: during  
OWL-Time: intervalDuring  
CIDOC: occurs during (includes) (P117)

**Object Property time:intervalEquals**

*Subproperty of:* time:intervalStarts, time:intervalStartedBy,  
time:intervalFinishes, time:intervalFinishedBy,  
time:intervalDuring, time:intervalContains  
*Description:* Property representing the \( X \) equal \( Y \) Allen’s relation (=) \[48\].  
*Examples:* Let \( T_1 \) denote June 2011 and \( T_2 \) represent the time interval starting at “1 June 2011 at 00:00” and ending at “30 June 2011 at 23:59”; \( T_1 \) intervalEquals \( T_2 \)  
*Similar to:* OWL-Time: intervalEquals  
CIDOC: is equal in time (P114)

**Object Property time:intervalFinishedBy**

*Subproperty of:* time:intervalOverlappedBy  
*InverseProperty:* time:intervalFinishes  
*Description:* Property representing the inverse of the \( X \) finishes \( Y \) Allen’s relation (fi) \[48\].  
*Examples:* Let \( T_1 \) represent December 2011 and \( T_2 \) denote year 2011; \( T_2 \) intervalFinishedBy \( T_1 \)  
*Similar to:* OWL-Time: intervalFinishedBy  
CIDOC: finishes (is finished by) (P115)

**Object Property time:intervalFinishes**

*Subproperty of:* time:intervalOverlaps  
*InverseProperty:* time:intervalFinishedBy  
*Description:* Property representing the \( X \) finishes \( Y \) Allen’s relation (f) \[48\].  
*Examples:* Let \( T_1 \) represent December 2011 and \( T_2 \) denote year 2011; \( T_1 \) intervalFinishes \( T_2 \)  
*Similar to:* LKIF: time:finishes  
OWL-Time: intervalFinishes  
CIDOC: finishes (is finished by) (P115)
<table>
<thead>
<tr>
<th>Property</th>
<th>Subproperty of:</th>
<th>InverseProperty</th>
<th>Description</th>
<th>Examples</th>
<th>Similar to:</th>
</tr>
</thead>
<tbody>
<tr>
<td>time:intervalMeets</td>
<td>time:intervalBefore, math:directlyPrecedes</td>
<td>time:intervalMetBy</td>
<td>Property representing the X meets Y Allen’s relation (m) [48].</td>
<td>Let (T_1) represent year 2011 and (T_2) denote year 2012: (T_1) intervalMeets (T_2)</td>
<td>LKIF: time:immediately_before, OWL-Time: intervalMeets, CIDOC: meets in time with (is met in time by) (P119)</td>
</tr>
<tr>
<td>time:intervalMetBy</td>
<td>time:intervalAfter, math:directlyFollows</td>
<td>time:intervalMeets</td>
<td>Property representing the inverse of the X meets Y Allen’s relation (mi) [48].</td>
<td>Let (T_1) represent year 2011 and (T_2) denote year 2012: (T_2) intervalMetBy (T_1)</td>
<td>LKIF: time:immediately_after, OWL-Time: intervalMetBy, CIDOC: meets in time with (is met in time by) (P119)</td>
</tr>
<tr>
<td>time:intervalOverlappedBy</td>
<td>-</td>
<td>time:intervalOverlaps</td>
<td>Property representing the inverse of the X overlaps Y Allen’s relation (oi) [48].</td>
<td>Let (T_1) represent winter 2011-2012 and (T_2) denote year 2012: (T_2) intervalOverlappedBy (T_1)</td>
<td>OWL-Time: intervalOverlappedBy, CIDOC: overlaps in time with (is overlapped in time by) (P118)</td>
</tr>
<tr>
<td>time:intervalOverlaps</td>
<td>-</td>
<td>time:intervalOverlappedBy</td>
<td>Property representing the X overlaps Y Allen’s relation (o) [48].</td>
<td>Let (T_1) represent winter 2011-2012 and (T_2) denote year 2012: (T_1) intervalOverlaps (T_2)</td>
<td>LKIF: time:overlap</td>
</tr>
</tbody>
</table>
Object Property `time:intervalStartedBy`

- **Subproperty of:** `time:intervalOverlappedBy`
- **InverseProperty:** `time:intervalStarts`
- **Description:** Property representing the inverse of the `X starts Y` Allen’s relation (si) [48].
- **Examples:** Let $T_1$ represent January 2011 and $T_2$ denote year 2011: $T_2$ `intervalStartedBy` $T_1$
- **Similar to:** 
  - OWL-Time: `intervalStartedBy`
  - CIDOC: starts (is started by) (P116)

Object Property `time:intervalStarts`

- **Subproperty of:** `time:intervalOverlappedBy`
- **InverseProperty:** `time:intervalStartedBy`
- **Description:** Property representing the `X starts Y` Allen’s relation (s) [48].
- **Examples:** Let $T_1$ represent January 2011 and $T_2$ denote year 2011: $T_1`intervalStarts` $T_2$
- **Similar to:** 
  - LKIF: `time:starts`
  - OWL-Time: `intervalStarts`
  - CIDOC: starts (is started by) (P116)

Object Property `time:isRelatedToCalendar`

- **Subproperty of:**
- **Description:** Property relating a date to the calendar system it is represented in.
- **Examples:** The date time “12 March 2012 at 16:42” is related to the Gregorian calendar system.

Object Property `time:startsAt`

- **Subproperty of:** `time:temporalProperty`
- **Description:** Property that relates a temporal entity to its end
- **Examples:** Let $T$ represent year 2011 and $t_i$ the instant with date time “1 January 2011 at 00:00”; $T`startsAt` $t_i$
- **Similar to:** OWL-Time: `hasBeginning`
A.1.5. Primitive-types

<table>
<thead>
<tr>
<th>Class</th>
<th>primitive-types:NegativeNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subclass of</td>
<td>primitive-types:Number</td>
</tr>
<tr>
<td>Description</td>
<td>this class comprises all instances of Number, with a value lesser than 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class</th>
<th>primitive-types:Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subclass of</td>
<td>primitive-types:PrimitiveValue</td>
</tr>
<tr>
<td>Description</td>
<td>class comprising all computable values</td>
</tr>
<tr>
<td>Examples</td>
<td>5, 3.64, 0.9f</td>
</tr>
<tr>
<td>Similar to</td>
<td>CIDOC Number (E60)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class</th>
<th>primitive-types:PositiveNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subclass of</td>
<td>primitive-types:Number</td>
</tr>
<tr>
<td>Description</td>
<td>this class comprises all instances of Number, with a value greater than 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class</th>
<th>primitive-types:PrimitiveValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subclass of</td>
<td>-</td>
</tr>
<tr>
<td>Description</td>
<td>class representing the primitive values</td>
</tr>
<tr>
<td>Examples</td>
<td>3.4, “dog”, 30.5f, 2012-02-03</td>
</tr>
<tr>
<td>Similar to</td>
<td>CIDOC Primitive Value (E59)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class</th>
<th>primitive-types:Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subclass of</td>
<td>primitive-types:PrimitiveValue</td>
</tr>
<tr>
<td>Description</td>
<td>this class comprises all instances used for documentation, such as text strings, bitmaps, etc.</td>
</tr>
<tr>
<td>Examples</td>
<td>“the quick brown fox jumps over the lazy cat”</td>
</tr>
<tr>
<td>Similar to</td>
<td>CIDOC String (E62)</td>
</tr>
</tbody>
</table>
A. Class primitive-types:TimePrimitive

Subclass of: primitive-types:PrimitiveValue
Description: this class comprises all instances of representation of time intervals
Examples: 13 May 1985, 85th Century BC
Similar to: CIDOC Time Primitive (E61)

A.1.6. Roles

Direct Imports
mereology

Indirect Imports

Short Description
This module contains a vocabulary for the representation of roles as proposed in [25].

Classes

Class roles:Configuration:

Subclass of:
Description: Anything having as component some configuration item.
Examples: The organisational structure of a University comprising students, professors, assistants, etc..
Similar to:

Class roles:ConfigurationItem

Subclass of:
Description: Anything playing some role in a Configuration (i.e. counting as ... within ...)
Examples: Any person playing some role (in the organisational structure) in a University.
Similar to:

Class roles:Role

Subclass of:
Description: A (generic) role is something imposed on some item in the context of some configuration.
Examples: The Student role (in the organisational structure) of a University.
Similar to: LKIF role:Role
Object Properties

ObjectProperty roles:count-as

Subproperty of:  
InverseProperty roles:imposed-on
Description Specifies that some role is played by something
Examples:  Mark count-as student
Similar to:  LKIF role:play

ObjectProperty roles:is_role

Subproperty of:  
InverseProperty Description A utility property to mark OWL classes representing a Role.
Examples:  The OWL Class Student is intended to represent a role.
Similar to:  

A.1.7. Math

<table>
<thead>
<tr>
<th>Direct Imports</th>
</tr>
</thead>
<tbody>
<tr>
<td>mereology</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Indirect Imports</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>This module contains all classes and properties pertaining to general mathematics, from first order logics definitions to Sets. It also contains the definition of a concept used to describe sequences, following the approach from [23]. General mathematical concepts are formalized using the description given in Wolfram [17].</td>
</tr>
</tbody>
</table>

Classes

Class math:AtomicFormula

Subclass of:  math:Sequence
Description sequence of elements of the type PT*, where P is a PredicateSymbol and T are Terms (used to describe Formulas in FOL)
Examples:  P(t1, t2), Q(t)
Similar to:  Wolfram: First-Order Logic
## Class math:BinaryConnective

**Subclass of:** math:LogicalConnective  
**Description:** Class representing the logical connectives that relate two elements  
**Examples:** →

## Class math:BinaryConnective_Formula

**Subclass of:** math:Sequence  
**Description:** Sequence describing formula of the form $A \ op \ B$ where $A$ and $B$ are formulas and $\ op$ is a binary connective.  
**Examples:** $A \to C$

## Class math:Constant

**Subclass of:** math:FunctionSymbol  
**Description:** Class representing constants in First Order Logics  
**Similar to:** Wolfram: First-Order Logic

## Class math:EmptyList

**Subclass of:** math:Sequence  
**Description:** Class representing the empty sequence  
**Similar to:** EmptyList, as described in [23]

## Class math:Exists

**Subclass of:** math:QuantifierSymbol  
**Description:** Class representing the logical symbol of existence  
**Examples:** $\exists$

## Class math:Exists_Formula

**Subclass of:** math:Sequence  
**Description:** Sequence describing an existential quantifier followed by a formula, used to describe Formulas in FOL  
**Examples:** $\exists x. f(x)$

## Class math:Expression

**Subclass of:** math:Sequence  
**Description:** Sequence describing a First Order Logic function of the form $F(t_1, .., t_n)$, where $F$ is a Function symbol and $t$ are terms  
**Examples:** $f(t_1,t2), C$  
**Similar to:** Wolfram: First-Order Logic
Class **math:FOLAlphabet**

*Subclass of:* -
*Description:* Superclass for all First Order Logic elements
*Examples:* P(x), for all x, then P(x)
*Similar to:* Wolfram: First-Order Logic

Class **math:ForAll**

*Subclass of:* math:QuantifierSymbol
*Description:* Class representing the *for-all* logical symbol
*Examples:* ∀

Class **math:ForAll_Formula**

*Subclass of:* math:Sequence
*Description:* Sequence describing formula of the form ∀x.F, where x is a variable and F is a formula.
*Examples:* ∀x.F(x)

Class **math:Formula**

*Subclass of:* -
*Description:* Inductive Definition:
- Predicate symbols. If P is an n-ary predicate symbol and t1, ..., tn are terms then P(t1,...,tn) is a formula.
- Equality. If t1 and t2 are terms, then t1 = t2 is a formula.
- Negation. If φ is a formula, then not φ is a formula.
- Binary connectives. If φ and ψ are formulas, then φ → ψ is a formula. Similar rules apply to other binary logical connectives.
- Quantifiers. If φ is a formula and x is a variable, then ∀x.φ and ∃x.φ are formulas.

Class **math:FunctionSymbol**

*Subclass of:* math:NonLogicalSymbol
*Description:* Class representing function symbols in First Order Logics
*Similar to:* Wolfram: First-Order Logic

Class **math:IdentitySymbol**

*Subclass of:* math:LogicalSymbol
*Description:* Class representing the logical symbol of identity
*Examples:* =
Class `math:LogicalConnective`

*Subclass of:* `math:LogicalSymbol`
*Description:* Class representing the logical connectives in First Order Logic
*Examples:* →, AND

Class `math:LogicalSymbol`

*Subclass of:* `math:FOLAlphabet`
*Description:* Class representing the logical symbols needed to express formulas in first order logic
*Examples:* →, ∀, =
*Similar to:* Wolfram: First-Order Logic

Class `math:NegationSymbol`

*Subclass of:* `math:LogicalSymbol`
*Description:* Class representing the logical symbol for negation
*Examples:* not

Class `math:NonEmptySequence`

*Subclass of:* `math:Sequence`
*Description:* Sequence containing at least one element
*Examples:* A+, BDFGF

Class `math:NonLogicalSymbol`

*Subclass of:* `math:FOLAlphabet`
*Description:* Class representing the non logical symbols needed to express formulas in first order logic
*Similar to:* Predicates or function symbols in Wolfram [17]: First-Order Logic

Class `math:Not_Formula`

*Subclass of:* `math:Sequence`
*Description:* Sequence describing a negated formula.
*Examples:* not(F)

Class `math:Pair`

*Subclass of:* `math:Sequence`
*Description:* Sequence of two ordered elements
*Examples:* <i, o>, AB
Class math:PredicateSymbol

Subclass of: math:NonLogicalSymbol
Description: Class representing predicate symbols in First Order Logics
Similar to: Wolfram: First-Order Logic

Class math:PropositionalVariable

Subclass of: math:PredicateSymbols
Description: Class representing propositional variables in First Order Logics
Similar to: Wolfram: First-Order Logic

Class math:PunctuationSymbol

Subclass of: math:LogicalSymbol
Description: Class representing punctuation in First Order Logics
Similar to: Wolfram: First-Order Logic

Class math:QuantifierSymbol

Subclass of: math:LogicalSymbol
Description: Class representing the logical quantifiers
Examples: forAll, exists, ∀, ∃

Class math:Seq_Terms

Subclass of: math:Sequence
Description: unbounded sequence of elements of type Term (used to describe Formulas in FOL)
Examples: <t1,t2,t3>, <t1>

Class math:Set

Subclass of: -
Description: a generic set
Examples: <1, 3, 5>, <dog, cat, tree>
Similar to: ODP:Set [18]

Class math:SetOfSequences

Subclass of: math:Set
Description: A set of elements of type Sequence
Examples: {ABC, T*}
Class math:Sequence

Subclass of: -
Description: A sequence of elements, similar to the Java List data structure. Structured following the principles of [23], it allows to represent regular expressions.
Examples: A*, ABB, A+C*
Similar to: OWLList, as described in [23]

Class math:T1_equal_T2

Subclass of: math:Sequence
Description: sequence used to represent the expression T1=T2, where T1 and T2 are Terms, and = is one instance of the math:IdentitySymbol.
Examples: T=P, A=B
Similar to: OWLList, as described in [23]

Class math:Term

Subclass of: math:NonLogicalSymbol
Description: a math:Expression or math:VariableSymbol, representing terms in First Order Logic.

Class math:VariableSymbol

Subclass of: math:LogicalSymbol
Description: class representing variables in First Order Logics
Similar to: Wolfram: First-Order Logic

Object Properties

Object Property math:directlyFollows

Subproperty of: math:follows
InverseProperty: math:directlyPrecedes
Description: functional property to assign a direct ordering (adjacent items)
Examples: B directlyFollows A, second item directlyFollows first item
Similar to: ODP:Sequence [18]

Object Property math:directlyPrecedes

Subproperty of: math:precedes
InverseProperty: math:directlyFollows
Description: functional property to assign a direct ordering (adjacent items).
**Examples:**  A directlyPrecedes B, first item directlyPrecedes second item

**Similar to:**  ODP:Sequence [18]

**Object Property math:first**

*Subproperty of:*  math:hasContents

*Description:*  identifies the first element of a pair

**Object Property math:follows**

*Subproperty of:*  -

*InverseProperty:*  math:precedes

*Description:*  Transitive property to assign ordering

**Examples:**  B follows A, third item follows first item

**Similar to:**  ODP:Sequence [18]

**Object Property math:hasContents**

*Subproperty of:*  math:hasSequenceMember

*Description:*  identifies the content of an element of a sequence

**Similar to:**  hasContents in [23]

**Object Property math:hasSequenceMember**

*Subproperty of:*  mereology:hasMember

*Description:*  identifies all the elements of a Sequence, as members of a set

**Object Property math:hasNext**

*Subproperty of:*  math:isFollowedBy, math:directlyPrecedes

*Description:*  equivalent to math:directlyPrecedes, but constraints the domain to the type Sequence

**Similar to:**  hasNext in [23]

**Object Property math:isFollowedBy**

*Subproperty of:*  math:precedes

*Description:*  equivalent to math:precedes, but constraints the domain to the type Sequence

**Similar to:**  isFollowedBy in [23]

**Object Property math:precedes**

*Subproperty of:*  -

*InverseProperty:*  math:follows
Description: Transitive property to assign ordering.
Examples: A precedes B, first item precedes second item
Similar to: ODP:Sequence [18]

Object Property math:second

Subproperty of: math:hasSequenceMember
Description: identifies the second element of a Pair

Data Properties

Data Property math:arity

Domain: math:NonLogicalSymbol
Range: xsd:nonNegativeInteger
Description: property used to define the arity of non logical symbols.
Examples: ConstantSymbols are FunctionSymbols with arity=0
A binary function is a Function with arity=2
PropositionalVariables are PredicateSymbols with arity=0

A.1.8. Math-calculus

<table>
<thead>
<tr>
<th>Direct Imports</th>
</tr>
</thead>
<tbody>
<tr>
<td>math</td>
</tr>
<tr>
<td>primitive-types</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Indirect Imports</th>
</tr>
</thead>
<tbody>
<tr>
<td>mereology</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module with definitions of functions and properties related to these concepts. The concepts were mainly derived from Wolfram [17].</td>
</tr>
</tbody>
</table>

Classes

Class math:Function

Subclass of: -
Description: the class representing a Function, as defined in [17], through its domain, codomain and the association function that map elements of the first set to elements of the second set.
Examples: add(x, y); int int_part(float)
Similar to: Wolfram: Function

Class math:PlottedFunction

Subclass of: math:Function
Description: the class representing a function described through a set
(Plot) of pairs, mapping possible inputs from the domain to the corresponding result in the codomain.

**Examples:**

\[ f(x) = \{(1,2), (2,3), (3,4)\} \]

**Similar to:** Wolfram: Function

### Class math:Plot

**Subclass of:** math:Set

**Description:** class representing a set of math:Pair, used to define math:PlottedFunction

### ObjectProperties

#### Object Property math:hasAssociationRule

**Subproperty of:** mereology:hasComponent

**Description:** property that links a function to its expression, or map

**Examples:** Function \( f \) hasAssociationRule “\( x+1 \)”

**Similar to:** Wolfram: Function

#### Object Property math:hasCodomain

**Subproperty of:** mereology:hasComponent

**Description:** property that links a function to its codomain

**Examples:** Function \( f \) hasCodomain \( \mathbb{N} \)

**Similar to:** Wolfram: Function

#### Object Property math:hasDomain

**Subproperty of:** mereology:hasComponent

**Description:** property that links a function to its domain

**Examples:** Function \( f \) hasDomain \( \mathbb{R} \)

**Similar to:** Wolfram: Function

### A.1.9. Math-algebra

<table>
<thead>
<tr>
<th><strong>Direct Imports</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>math-calculus</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Indirect Imports</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>math</td>
</tr>
<tr>
<td>mereology</td>
</tr>
<tr>
<td>primitive-types</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Short Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Module containing the concepts and properties relative to algebras, as defined in Wolfram [17].</td>
</tr>
</tbody>
</table>

---

D.3.4 - Second Version Of The ASSERT Ontology 107 / 165
Classes

Class math:Algebra

Subclass of: math:Set
Description: concept representing an algebra as a set plus some Operations on it
Examples: Natural numbers with addition
Similar to: Wolfram: Algebra

Class math:CartesianProduct

Subclass of: math:SetOfSequence
Description: The Cartesian product of two sets \( A \) and \( B \) (also called the product set, set direct product, or cross product) is defined to be the set of all \((a,b)\) points where \( a \in A \) and \( b \in B \)
Examples: a plane cartesian product \( RxR \)
Similar to: Wolfram: Cartesian Product

Class math:Homomorphism

Subclass of: math:Function
Description: homomorphism is a mapping between two Algebraic structures
Examples: If \( G \) and \( H \) are groups, a homomorphism from \( G \) to \( H \) is a function \( f:G\rightarrow H \) such that \( f(g_1\ast g_2) = f(g_1)\ast' f(g_2) \), for any elements \( g_1, g_2 \in G \), where \( \ast \) denotes the operation in \( G \) and \( \ast' \) denotes the operation in \( H \).
Similar to: Wikipedia [51]: Homomorphism

Class math:IdentityFunction

Subclass of: math:Function
Description: a particular type of function with domain = codomain that assigns every element of the domain to the same element of the codomain
Examples: \( id(x)=x \)
Similar to: Wolfram: Identity Function

Class math:MathematicalProperty

Subclass of: math:Function
Description: a mathematical property is a structural characteristic of a mathematical object.
Examples: transitivity, monotonicity
Class math:OrderingFunction

Subclass of: math:Function
Description function that given a set of elements returns the ordered sequence of the same elements.
Examples: (1,3)\rightarrow(1,3) (3,4)\rightarrow(1,3,4)

ObjectProperties

Object Property math:hasOperation

Subproperty of: mereology:hasComponent
Description this property links an operation to a set, so to define an algebra
Examples: (N,+) hasOperation addition

Object Property math:hasNeutralElement

Subproperty of: mereology:hasComponent
Description property defining the neutral element of an algebra
Examples: (N, +) hasNeutralElement 0

A.1.10. Math-discrete

<table>
<thead>
<tr>
<th>Direct Imports</th>
<th>math-calculus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indirect Imports</td>
<td>math</td>
</tr>
<tr>
<td></td>
<td>mereology</td>
</tr>
<tr>
<td></td>
<td>primitive-types</td>
</tr>
</tbody>
</table>

Short Description
Module containing concepts related to Computer Science

Classes

Class math: Algorithm

Subclass of: -
Description concept representing a specific set of instructions for carrying out a procedure or solving a problem, usually with the requirement that the procedure terminate at some point.
Similar to: Wolfram : Algorithm
**Class math: Computation**

*Subclass of:* -  
*Description:* concepts representing an operation that begins with some initial conditions and gives an output which follows from a definite set of rules  
*Examples:* computations performed by computers, in which the fixed set of rules may be the functions provided by a particular programming language  
*Similar to:* Wolfram: Computation

**Class math: ComputingMachine**

*Subclass of:* -  
*Description:* concepts representing model of a computer system (considered either as hardware or software) constructed to allow a detailed and precise analysis of how the computer system works

**Class math: TuringMachine**

*Subclass of:* math: ComputingMachine  
*Description:* A Turing machine is a theoretical computing machine to serve as an idealized model for mathematical calculation  
*Similar to:* Wolfram: TuringMachine

**Class math: ComputableFunction**

*Subclass of:* math: Function  
*Description:* Any computable function can be incorporated into a program using while-loops (i.e., "while something is true, do something else").  
*Similar to:* Wolfram: ComputableFunction

**Class math: PrimitiveRecursiveFunction**

*Subclass of:* math: ComputableFunction  
*Description:* A function that can be implemented using only do-loops is called primitive recursive  
*Examples:* power, greatest common divisor etc.  
*Similar to:* Wolfram: PrimitiveRecursiveFunction

**Class math: Instruction**

*Subclass of:* -  
*Description:* -
Class math: For-loop

Subclass of: math: Instruction
Description -

Class math: ProgrammingLanguageInstruction

Subclass of: math: Instruction
Description -

Class math: While-loop

Subclass of: math: Instruction
Description -

Class math: Program

Subclass of: -
Description A precise sequence of instructions designed to accomplish a given task
Similar to: Wolfram: Program

Class math: AlgorithmImplementation

Subclass of: math: Program
Description -

Class math: ProgrammingLanguage

Subclass of: -
Description -

Class math: ProgrammingLanguageContract

Subclass of: math: ProgrammingLanguage
Description -

ObjectProperties

Object Property math:followsFrom

Subproperty of: -
Description -
A.1.11. Agents

<table>
<thead>
<tr>
<th>Direct Imports</th>
<th>Indirect Imports</th>
</tr>
</thead>
<tbody>
<tr>
<td>math-calculus</td>
<td>mereology</td>
</tr>
<tr>
<td>roles</td>
<td>math</td>
</tr>
<tr>
<td></td>
<td>primitive-types</td>
</tr>
</tbody>
</table>

Short Description
An ontology introducing the concepts related to agents acting and sensing in an environment.

Classes

Class agents:Environment

Subclass of: math:Function
Description: Anything being constituted by a collection of signals.

Class agents:Action

Subclass of: math:Function
Description: Any function mapping signals to stimulus

Class agents:PersistentItem

Subclass of:
Description: This class comprises items that have a persistent identity, sometimes known as “endurants” in philosophy.
Similar to: CIDOC-CRM: E77.Persistent_Item
Class agents:Agent

Subclass of: agents:PersistentItem
Description: Anything having its own resources, possessing skills, able to act and sense its environment and driven by some tendency. Agent (active objects) are disjoint from (passive) objects.
Similar to: CIDOC-CRM E39_Actor
LKIF: action:Agent

Class agents:IndividualAgent

Subclass of: agents:Agent
Description: An individual agent (disjoint from Organisations)
Examples: My dog
Napoleon

Class agents:Organisation

Subclass of: agents:Agent
Description: An organisation is an agent (i.e. something having goals, resources and skills) having as members other agents.
Examples: City University of London
Fondazione Ugo Bordoni
Similar to: LKIF: action:Organisation
CIDOC-CRM: E74_Group

Class agents:Appellation

Subclass of: agents:PersistentItem
Description: This class comprises all proper names, words, phrases or codes, either meaningful or not, that are used or can be used to identify a specific instance of some class within a certain context
Examples: E41_Appellation
Similar to: CIDOC-CRM: E41.Appellation

Class agents:Agent_Appellation

Subclass of: agents:Appellation
Description: This class comprises any sort of name, number, code or symbol characteristically used to identify an Agent
Examples: Mary
Similar to: CIDOC-CRM: E82 Actor_Appellation
Class agents:Object_Identifier

Subclass of: agents:PersistemItem
Description: This class comprises codes assigned to objects in order to identify them UNIQUELY within the context of one or more organisations.
Examples: Colosseum
Similar to: CIDOC-CRM: E.42 Object_Identifier

Class agents:Object

Subclass of: agents:PersistentItem
Description: An object is something that allows some transformation. They can be either intellectual products or physical things, and are characterized by relative stability. They may for instance either have a solid physical form, an electronic encoding, or they may be logical concept or structure.
Examples: A pen
Similar to: CIDOC-CRM: E70.Thing

Class agents:Actuator

Subclass of: agents:Object
Description: A component of an agent allowing some action.
Examples: A hand

Class agents:Sensor

Subclass of: agents:Object
Description: A component of an agent allowing some perception.
Examples: An ear

Class agents:Signal

Subclass of: -
Description: Anything in an agent’s environment and perceivable by an agent.
Examples: The mechanical waves produced by a loudspeaker.

Class agents:Stimulus

Subclass of: -
Description: The result of agent’s perception.
Examples: The sound patterns resulting in the brain as result of the hearing process.
Class agents:State

Subclass of: 
Description Anything driving an agent’s behaviour. Objectives or satisfaction/survival functions which an agent try to optimise.
Examples: To avoid obstacles while walking.
          To become a billionaire.

Class agents:Action

Subclass of: math:Function
Description Any function mapping stimulus to signals.
Examples: The transformation of an electrical stimulus into movement.

Class agents:Decision

Subclass of: math:Function
Description Any function mapping an agent’s state to stimuli.
Examples: The transformation of a mechanical wave into electrical stimuli.

Class agents Perception

Subclass of: math:Function
Description Any function mapping signals to stimulus
Examples: The transformation of a mechanical wave into a electrical stimulus.

Object Properties

ObjectProperty agents:allows

Subproperty of: 
Description The operations allowed by an object.
Examples: Turn on a light
          Withdraw a sum from an account

ObjectProperty agents:hasResource

Subproperty of: 
Description The resources possessed by an agent.
Examples: Money on Mark's bank account.

ObjectProperty agents:canPerform

Subproperty of: 

**ASSERT4Soa**

`Description` The action an agent is able to perform.

`Examples:` Grasping an object.
Decoding a message.

**ObjectProperty agents:canSense**

`Subproperty of:` -
`Description` The perceptions an agent is able to sense.

**ObjectProperty agents:hasBody**

`Subproperty of:` -
`Description` The object representing the embodiment of an agent

**ObjectProperty agents:hasSkill**

`Subproperty of:` -
`Description` The skills possessed by an agent

**ObjectProperty agents:hasTendency**

`Subproperty of:` -
`Description` The tendencies driving an agent

**ObjectProperty agents:bearsFeature**

`Subproperty of:` -
`Description` The features of an object

**ObjectProperty agents:hasScope**

`Subproperty of:` -
`Description` The scope of an identifier

**ObjectProperty agents:has_alternative_form**

`Subproperty of:` -
`Description` The alternative form of an identifier

**ObjectProperty agents:identifies**

`Subproperty of:` -
`InverseProperty` agents:isIdentifiedBy
`Description` The items identified by an identifier
### Assert4Soa

**ObjectProperty agents::isPerceivedBy**

*Subproperty of:* -  
*Description:* The agents perceiving a behaviour

**ObjectProperty agents::isShownBy**

*Subproperty of:* -  
*Description:* The behaviour shown by an agent

### A.1.12. WSDL

<table>
<thead>
<tr>
<th>Direct Imports</th>
</tr>
</thead>
<tbody>
<tr>
<td>agents</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Indirect Imports</th>
</tr>
</thead>
<tbody>
<tr>
<td>mereology</td>
</tr>
<tr>
<td>math</td>
</tr>
<tr>
<td>math-calculus</td>
</tr>
<tr>
<td>primitive-types</td>
</tr>
<tr>
<td>roles</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>An OWL model of the W3C WSDL specifications.</td>
</tr>
</tbody>
</table>

#### Classes

**Class wsdl::MessageExchangePattern**

*Subclass of:* math:Sequence  
*Description:* Define the sequence and cardinality of abstract messages listed in an operation  
*Examples:* IN, IN-OUT, OUT

**Class wsdl::OperationStyle**

*Subclass of:* -  
*Description:* The style of an operation.  
*Examples:* RPC, IRI, Multipart.

**Class wsdl::Endpoint**

*Subclass of:* agents::Agent_Appellation  
*Description:* Defines the particulars of a specific endpoint at which a given service is available.
Class `wsdl:Interface`

*Subclass of:* agents:Sensor, agents:Actuator  
*Description*  
An interface is a set of operations.

Class `wsdl:Service`

*Subclass of:* agents:Agent  
*Description*  
describes a set of endpoints at which a particular deployed implementation of the service is provided.

Class `wsdl:InterfaceOperation`

*Subclass of:* agents:Perception, agents:Action  
*Description*  
An operation is an interaction with the service consisting of a set of (ordinary and fault) messages exchanged between the service and the other parties involved in the interaction.

Class `wsdl:SafeOperation`

*Subclass of:* math:InterfaceOperation  
*Description*  
The class of operations that are safe (e.g. no cost) for users to invoke.

Class `wsdl:InterfaceFaultReference`

*Subclass of:* agents:Signal  
*Description*  
Defines the fault message exchanged in an operation.

Class `wsdl:InterfaceMessageReference`

*Subclass of:* math:Signal  
*Description*  
Defines the content, or payload, of a message exchanged in an operation.

**Object Properties**

**ObjectProperty `wsdl:input`**

*Subproperty of:* math:hasDomain, math:interfaceMessageReference  
*InverseProperty*  
*Description*  
A message coming to the Service.

**ObjectProperty `wsdl:XMLInput`**

*Subproperty of:* `wsdl:input`  
*Description*  
An xml message coming to the service.
ObjectProperty `wsdl:output`

*Subproperty of:* `math:hasCodomain`, `math:interfaceMessageReference`

*Description:* A message going from the service

ObjectProperty `wsdl:interface`

*Subproperty of:* `agents:allows`

*Description:* The interface of a service.

ObjectProperty `wsdl:interfaceOperations`

*Subproperty of:* `agents:allows`

*Description:* The operations within an interface.

ObjectProperty `wsdl:endpoints`

*Subproperty of:* `agents:isIdentifiedBy`

*Description:* The endpoints of a service.

ObjectProperty `wsdl:extendedInterfaces`

*Subproperty of:* `mereology:hasComponent`

*Description:* The interfaces extended by an interface.

ObjectProperty `wsdl:interfaceFaultReference`

*Subproperty of:* `mereology:hasComponent`

*Description:* The faults of an interface.

ObjectProperty `wsdl:inputFault`

*Subproperty of:* `math:hasDomain`, `wsdl:interfaceFaultReference`

*Description:* An input fault of an operation

ObjectProperty `wsdl:outputFault`

*Subproperty of:* `wsdl:interfaceFaultReference`

*Description:* An output fault of an operation

ObjectProperty `wsdl:interfaceMessageReference`

*Subproperty of:* `mereology:hasComponent`

*Description:* The messages exchanged by an operation.
ObjectProperty `wsdl:style`

**Subproperty of:** mereology:hasComponent

**Description**
The style of an operation.

### A.1.13. Measurements

<table>
<thead>
<tr>
<th>Direct Imports</th>
</tr>
</thead>
<tbody>
<tr>
<td>primitive-types</td>
</tr>
<tr>
<td>math:algebra</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Indirect Imports</th>
</tr>
</thead>
<tbody>
<tr>
<td>merology</td>
</tr>
<tr>
<td>math</td>
</tr>
<tr>
<td>math:calculus</td>
</tr>
</tbody>
</table>

**Short Description**
An ontology for concepts related to the measurement theory.

### Classes

**Class `measurement:Measure`**

**Subclass of:** math:Function

**Description**
Functions assigning numbers to some property (e.g. “speed”) of an entity according to some rule.

**Examples:** Measurement of the speed of a body.

**Class `measurement:ScaleOfMeasure`**

**Subclass of:** math:Algebra, measurement:NumericSet

**Description**
The class of different type measurements.

**Examples:** Interval Scale, Ratio Scale.

**Class `measurement:IntervalScale`**

**Subclass of:** Measurement:ScaleOfMeasure

**Description**
Scale point are totally ordered. The distance between scale points is equal. The zero point does not means lack of quality being measured (e.g. the zero in the Celsius scale does not means lack of movement of molecules).

**Examples:** Fahrenheit or Celsius scale.
Class measurement: NominalScale

Subclass of: Measurement: ScaleOfMeasure
Description: Standard Set Structure. Points are not ordered.
Examples: Classifying people according to gender is a common application of a nominal scale e.g. assigning the number "1" to "female" and "2" to male.

Class measurement: OrdinalScale

Subclass of: measurement: ScaleOfMeasure
Description: Totally Ordered Set. The distance between scale points is not equal.
Examples: The Mohs scale of mineral hardness

Class measurement: RatioScale

Subclass of: measurement: ScaleOfMeasure
Description: One-dimensional vector space The measurement is the estimation of the ratio between a magnitude of a continuous quantity and a unit magnitude of the same kind. The zero point means lack of quality being measured (e.g. the zero in the Kelvin scale does means lack of movement of molecules).
Examples: Kelvin temperature scale.

Class measurement: UnitOfMeasurement

Subclass of: measurement: ScaleOfMeasure
Description: A unit of measurement is a definite magnitude of a physical quantity, defined and adopted by convention and/or by law.
Examples: Meter, Ampere
Similar to: CIDOC-CRM : E58_MeasurementUnit

Class measurement: SystemOfMeasurement

Subclass of: measurement: ScaleOfMeasure
Description: A set of units which can be used to specify anything which can be measured.
Examples: International System of Measurement

Class measurement: NumericSet

Subclass of: math: Set
Description: Any set containing only numbers.
Examples: The set \{1, 345, 234\}
Class `measurement:MeasurementValue`

*Subclass of:* `math:Number`

*Description* The ratio of a physical quantity, such as a length, time, temperature etc., to a unit of measurement.

*Examples:* 3 meters, 1 Hz

**Object Properties**

**ObjectProperty** `measurement:hasMeasure`

*Subproperty of:* -

*InverseProperty* `measurement:isMeasuredBy`

*Description* The scale of a measurement values.

**ObjectProperty** `measurement:hasScalse`

*Subproperty of:* -

*Description* The unit of measurement of a measurement value.

**ObjectProperty** `measurement:hasUnit`

*Subproperty of:* -

*Description* The unit of measurement of a measurement value.

**A.1.14. Mental-states**

<table>
<thead>
<tr>
<th><strong>Direct Imports</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>statement</td>
</tr>
<tr>
<td>agents</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Indirect Imports</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>math-calculus</td>
</tr>
<tr>
<td>mereology</td>
</tr>
<tr>
<td>math</td>
</tr>
<tr>
<td>roles</td>
</tr>
<tr>
<td>primitive-types</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Short Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>An ontology introducing mental states, cognitive agents and related concepts. This ontology reuse concepts form the LKIF Ontology [15]. Descriptions of classes imported by LKIF are partially carbon-copied for the reader's convenience.</td>
</tr>
</tbody>
</table>
Classes

Class mental_states:Mental_Object

Subclass of:
Description: [LKIF] “Metaphor of physical concepts, i.e. the things we mentally manipulate, either in thought or memory”
Similar to: LKIF Mental_Object

Class mental_states:Proposition

Subclass of: mental-states:Mental_Object
Description: [LKIF] “A (non logical) proposition is a proposition qualified by a propositional attitude. NB: The proposition used here does not correspond to a proposition in proposition logics.”
Similar to: LKIF Proposition

Class mental_states:Propositional_Attitude

Subclass of: mental-states:Mental_Object
Description: [LKIF] “A propositional attitude connects a person (the holder of the attitude) to some proposition, in fact it expresses some qualification over the proposition.”
Examples: The belief held by Mark that web service from www.assert4soa.eu are secure.
Similar to: LKIF Propositional_Attitude

Class mental_states:Belief

Subclass of: mental-states:Mental_Object
Description: [LKIF] “Something an agent 'believes in', i.e. holds as true”
Examples: The belief held by Mark that web service from www.assert4soa.eu are secure.

Class mental_states:Desire

Subclass of: mental-states:Mental_Object
Description: [LKIF] “A feeling of wanting”
Similar to: LKIF Desire

Class mental_states: Intention

Subclass of: mental-states:Mental_Object
Description: [LKIF] “Intention is where the agent expects certain consequences of his or her actions and desires those consequences to occur.”
Examples: The intention of John to verify that web service from
www.assert4soa.eu are secure.

**Similar to:** LKIF Intention

**Class mental states:Quality**

**Subclass of:** 

**Description:** A quality is an attribute or a property. Attributes are ascribable, by a subject, whereas properties can be possessed.

**Class mental states:Attribute**

**Subclass of:** mental-states:Quality

**Description:** Attributes are ascribable, by a subject.

**Class mental states:Property**

**Subclass of:** statement:Statement

**Description:** A property is possessed by something.

**Examples:** Redness of a red object.

**Class mental states:Legal_Person**

**Subclass of:** agents:Agent

**Description:** LKIF “A legal entity is a natural person or a legal construct through which the law allows a group of natural persons to act as if it were a single composite individual for certain purposes.”

**Similar to:** LKIF Legal_Person

**Class mental states:Private_Legal_Person**

**Subclass of:** mental-states:Legal_Person

**Description:** LKIF “A legal person as defined in private law”

**Similar to:** LKIF Legal_Person

**Class mental states:Public_Body**

**Subclass of:** mental-states:Legal_Person

**Description:** LKIF “A public body or body created by an act of law to serve a public interest”

**Similar to:** LKIF Public_Body

**Class mental states:Legislative_Body**

**Subclass of:** mental-states:Public_Body

**Description:** LKIF “A legislature is a type of (representative) deliberative assembly with the power to adopt laws.”
Similar to: LKIF Public_Body

Class mental_states:CognitiveAgents

Subclass of: agents:IndividualAgent
Description Any individual agent that hold some mental object.

Class mental_states:Inference

Subclass of: math:Function
Description Derivation of logical conclusions from premises known or assumed to be true.

Class mental_states:Abduction

Subclass of: mental-states:Inference
Description Inference by abduction
Examples: Given evidence $E$ and candidate explanations $H_1, ..., H_n$ of $E$, if $H_i$ explains $E$ better than any of the other hypotheses, infer that $H_i$ is closer to the truth than any of the other hypotheses.

Class mental_states:Deduction

Subclass of: mental-states:Inference
Description Inference by deduction

Class mental_states:Induction

Subclass of: mental-states:Inference
Description Inference by Induction

Object Properties

ObjectProperty mental_states:attitude

Subproperty of: 
InverseProperty mental-states:towards
Description LKIF “A towards relation between a propositional attitude and a proposition expresses that the attitude is held towards that proposition”
Similar to: LKIF attitude

ObjectProperty mental_states:held_by

Subproperty of: 
InverseProperty mental-states:holds
**ObjectProperty mental-states:belived_by**

- **Subproperty of:** mental-states:held_by
- **InverseProperty:** mental-states:believes
- **Description:** LKIF “Relates a belief to the agent holding the belief"
- **Similar to:** LKIF believed_by

**ObjectProperty mental-states:observer**

- **Subproperty of:** 
- **InverseProperty:** mental-states:observes
- **Description:** LKIF “Relates a believed observation to the agent observing it"
- **Similar to:** LKIF observer

**ObjectProperty mental-states:intended_by**

- **Subproperty of:** mental-states:held_by
- **InverseProperty:** mental-states:intends
- **Description:** LKIF “Specifies that some intention is held by some agent"
- **Similar to:** LKIF intended_by

**A.1.15. Communication**

<table>
<thead>
<tr>
<th><strong>Direct Imports</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>mental-states</td>
</tr>
<tr>
<td>wsdl</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Indirect Imports</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>mereology</td>
</tr>
<tr>
<td>math</td>
</tr>
<tr>
<td>agents</td>
</tr>
<tr>
<td>math-calculus</td>
</tr>
<tr>
<td>statement</td>
</tr>
<tr>
<td>roles</td>
</tr>
<tr>
<td>primitive-types</td>
</tr>
</tbody>
</table>

**Short Description**

An ontology defining concepts related to communication between cognitive agents. This ontology reuse concepts form the LKIF Ontology [15]. Descriptions of classes imported by LKIF are partially carbon-copied for the reader's convenience.
Classes

Class communication:Medium

Subclass of: -
Description: Anything being, among the other thing, a carrier of some Expression.
Similar to: LKIF Medium

Class communication:Document

Subclass of: communication:Medium
Description: Any document is a carrier of Statements in writing.
Similar to: LKIF Document

Class communication:LegalDocument

Subclass of: communication:Document
Description: LKIF "A legal document is a document bearing norms or normative statements."
Examples: Code, Contract, Directive, Statute
Similar to: LKIF norm:Legal_Document

Class communication:Expression

Subclass of: mental-states-proposition
Description: An expression is a proposition carried by some medium.
Similar to: LKIF expression:Expression

Class communication:Legal_Expression

Subclass of: communication:Expression
Description: LKIF "Legal expressions are created by some legal speech act and qualified by a communicated attitude"
Similar to: LKIF Legal_Expression

Class communication:Definitional_Expression

Subclass of: communication:Legal_Expression
Description: LKIF "A definition in a legal context (for example “x means y” or “by x it is meant y”)"
Similar to: LKIF Definitional_Expression

Class communication:Evaluative_Expression

Subclass of: communication:Legal_Expression
**Class communication:Existential_Expression**

*Description*
LKIF “Establishes or terminates the existence of a legal entity”

*Examples:* “the company ceases to exist”

*Similar to:* LKIF Existential_Expression

**Class communication:Potestative_Expression**

*Description*
LKIF “Attributes a power to some agent”

*Examples:* “a worker has the power to terminate his work contract”

*Similar to:* LKIF Potestative_Expression

**Class communication:Qualificatory_Expression**

*Description*
LKIF “Ascribes a legal role to a person or an object”

*Examples:* “x is a citizen”, “x is an intellectual work”, “x is a technical invention”

*Similar to:* LKIF Qualificatory_Expression

**Class communication:Communicated_Attitude**

*Description*
LKIF “A communicated attitude is a propositional attitude involved in an act of communication.”

*Similar to:* LKIF Communicated_Attitude

**Class communication:Assertion**

*Description*
LKIF “It's propositional content can be true of false.”

*Similar to:* LKIF Assertion

**Class communication:Declaration**

*Description*
LKIF “Searle: the successful performance of a declaration is
sufficient to bring about the fit between words and world, to
make the propositional content true.”

Similar to: LKIF Declaration

Class communication:Promise

Subclass of: communication:Communicated_Attitude
Description LKIF “A promise is a communicated attitude about some
future action or state”
Similar to: LKIF Promise

Class communication:Statement_In_Writing

Subclass of: communication:Communicated_Attitude
Description LKIF “Not to be confused with the actual writing/document
itself, which is the medium of the statement.”
Similar to: LKIF Statement_In_Writing

Object Properties

ObjectProperty communication:hasMedium

Subproperty of: -
InverseProperty communication:isCarrierFor
Description Relates an expression to the medium it is borne in.
Similar to: LKIF medium

ObjectProperty communication:stated_by

Subproperty of: metal-states:attitude
Description Relates a statement to its author

ObjectProperty communication:asserted_by

Subproperty of: communication:stated_by
InverseProperty Relates an expression being asserted to the assertion
Description

ObjectProperty communication:declared_by

Subproperty of: communication:stated_by
Description Relates a declared expression to the attitude to the
declaration
Assert4Soa

ObjectProperty communication:promised_by

Subproperty of: communication:stated_by
Description Relates an expression to the promise over the expression

ObjectProperty communication:utters

Subproperty of: mental-states:holds
InverseProperty communication:utterer
Description Relates an agent to its utterance(s)

ObjectProperty communication:hasQualification

Subproperty of: communication:statement:hasObject
Description Relates a qualificatory expression to is qualification.

A.1.16. Organisations

<table>
<thead>
<tr>
<th>Direct Imports</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>communication</td>
<td></td>
</tr>
<tr>
<td>dependency</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Indirect Imports</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>mereology</td>
<td></td>
</tr>
<tr>
<td>math</td>
<td></td>
</tr>
<tr>
<td>math-calculus</td>
<td></td>
</tr>
<tr>
<td>agents</td>
<td></td>
</tr>
<tr>
<td>primitive-types</td>
<td></td>
</tr>
<tr>
<td>statement</td>
<td></td>
</tr>
<tr>
<td>roles</td>
<td></td>
</tr>
<tr>
<td>mental-states</td>
<td></td>
</tr>
<tr>
<td>wsdl</td>
<td></td>
</tr>
</tbody>
</table>

| Short Description | An ontology introducing concepts for organizational structures (e.g. roles and dependencies between roles). |

Classes

Class organisation:PredefinedOrganisation

Subclass of: agents:Organisation
Description

Class organisation:OrganisationalStructure

Subclass of: roles:Configuration
Description
### Class `organisation:OrganisationalRelationship`

<table>
<thead>
<tr>
<th>Subclass of:</th>
<th><code>dependency:Dependency</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td></td>
</tr>
</tbody>
</table>

### Class `organisation:Acquaintance`

<table>
<thead>
<tr>
<th>Subclass of:</th>
<th><code>organisation:OrganisationalRelationship</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>The depender has some representation (dependum) of the dependee.</td>
</tr>
</tbody>
</table>

### Class `organisation:Communication`

<table>
<thead>
<tr>
<th>Subclass of:</th>
<th><code>organisation:OrganisationalRelationship</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Depender can send messages to Dependee. Depender depends on the dependee for the provision/availability of a communication channel.</td>
</tr>
</tbody>
</table>

### Class `organisation:Concurrency`

<table>
<thead>
<tr>
<th>Subclass of:</th>
<th><code>organisation:OrganisationalRelationship</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Depender needs to coordinate with dependee to access a shared resource (dependum).</td>
</tr>
</tbody>
</table>

### Class `organisation:Incompatibility`

<table>
<thead>
<tr>
<th>Subclass of:</th>
<th><code>organisation:OrganisationalRelationship</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Depender has a conflicting goal (dependum) with the dependee.</td>
</tr>
</tbody>
</table>

### Class `organisation:Provision`

<table>
<thead>
<tr>
<th>Subclass of:</th>
<th><code>organisation:OrganisationalRelationship</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Depender depends on dependee for the provisioning of a resource (dependum).</td>
</tr>
</tbody>
</table>

### Class `organisation:Subordination`

<table>
<thead>
<tr>
<th>Subclass of:</th>
<th><code>organisation:OrganisationalRelationship</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Depender depends on the dependee for the execution of an activity (dependum).</td>
</tr>
</tbody>
</table>

### Class `organisation:Trust`

<table>
<thead>
<tr>
<th>Subclass of:</th>
<th><code>organisation:OrganisationalRelationship</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Depender places its faith in dependee in respect of a certain</td>
</tr>
</tbody>
</table>
belief (dependum).

Class organisation:OrganisationalRole

Subclass of: organisation:OrganisationalRelationship

Description

Object Properties

ObjectProperty organisation:hasStructure

Subproperty of: -

Description

A.1.17. Activity

Direct Imports
communication

Indirect Imports
mereology
math
math-calculus
agents
statement
mental-states
primitive-types
roles
wsdl

Short Description
An ontology for Activity Theory [30].

Classes

Class activity:Activity

Subclass of: agents:Persistent_Item
Description The concept of Activity according to Activity Theory
Examples: Forging some steel with an hammer to make a knife to cut bread.

Class activity:CommunicationActivity

Subclass of: activity:Activity
Description Any activity using some medium as tool to transmit some request aimed to produce a desired behaviour from addressee.
Examples: Using a web page to request the delivery of a book

Class activity:Behaviour

Subclass of: math:Sequence
Description Any sequence containing only CommunicationAction

Class activity:CommunicationProtocol

Subclass of: activity:Behaviour
Description Any sequence containing only Action
Similar to: PSL Occurrence trees [31].

Object Properties

ObjectProperty activity:hasGoal

Subproperty of: activity:hasConstituency
Description The objective of an activity

ObjectProperty activity:hasObject

Subproperty of: activity:hasConstituency
Description The object of the transformation performed in the context of an activity.

ObjectProperty activity:hasOutcome

Subproperty of: activity:hasConstituency
InverseProperty activity:isOutcomeOf
Description The outcome of the transformation performed in the context of an activity.

ObjectProperty activity:hasSubject

Subproperty of: activity:hasConstituency
Description The agent performing the activity.

ObjectProperty activity:performs

Subproperty of: activity:hasConstituency
InverseProperty activity:isPerformedIn
Description hasConstituency
Examples: The transformation performed in the context of an activity.
ObjectProperty activity:useTools

Subproperty of: activity:hasConstituency
Description: The tools used by agent to perform a transformation. The artefact mediating the interaction between the agent and the object of the transformation.

A.1.18. Requirements

<table>
<thead>
<tr>
<th>Direct Imports</th>
<th>Direct Imports</th>
<th>Direct Imports</th>
<th>Direct Imports</th>
</tr>
</thead>
<tbody>
<tr>
<td>communication</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Indirect Imports</th>
<th>Indirect Imports</th>
<th>Indirect Imports</th>
<th>Indirect Imports</th>
</tr>
</thead>
<tbody>
<tr>
<td>mereology</td>
<td>math</td>
<td>math-calculus</td>
<td>agents</td>
</tr>
<tr>
<td>statement</td>
<td>mental-states</td>
<td>primitive-types</td>
<td>roles</td>
</tr>
<tr>
<td>wsd1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Short Description
An ontology for requirements and Denotational Semantic [36].

Classes

Class requirement:Requirement

Subclass of: requirement:Communication:Expression
Description: A primitive class for expressions intended to represent requirements.

Class requirement:FunctionalRequirement

Subclass of: requirement:Requirement
Description
Examples: The interpretation of the expression “SQUARE(X)” is $X^2$
Similar to: Denotation in [36]

Object Properties

ObjectProperty requirement:isInterpretationFor

Subproperty of: 
InverseProperty hasInterpretation
Description: Reference to the math:Expression denoted by a FunctionalRequirement
ObjectProperty requirement::hasPostconditions

Subproperty of:  -
Description:  Reference to the post-conditions of functional requirements

ObjectProperty requirement::hasNextInput

Subproperty of:  requirement::hasPostconditions
Description:  Reference to the input resulting from the evaluation of a math::Expression.

ObjectProperty requirement::hasNextState

Subproperty of:  requirement::hasPostconditions
Description:  Reference to the state resulting from the evaluation of a math::Expression.

ObjectProperty requirement::hasOutput

Subproperty of:  requirement::hasPostconditions
Description:  Reference to the output resulting from the evaluation of a math::Expression.

ObjectProperty requirement::throwsException

Subproperty of:  requirement::hasPostconditions
Description:  Reference to the exception thrown by the evaluation of a math::Expression.

ObjectProperty requirement::hasPrecondition

Subproperty of:  -
Description:  Reference to the pre-condition for the evaluation of a math::Expression

ObjectProperty requirement::hasInput

Subproperty of:  requirement::hasPrecondition
Description:  Reference to the input assumed for the interpretation of a math::Expression.

ObjectProperty requirement::hasState

Subproperty of:  requirement::hasPrecondition
Description:  Reference to the state assumed for the interpretation of a math::Expression.
**A.1.19. A4S Language**

<table>
<thead>
<tr>
<th>Direct Imports</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indirect Imports</td>
<td>None</td>
</tr>
<tr>
<td>Short Description</td>
<td>Module containing concepts related to the ASSERT4SOA Language [10]. This module supports definition of ASSERT artifacts related to the ASSERTCore part of the language.</td>
</tr>
</tbody>
</table>

**Classes**

**Class a4s-language: Asset**

| Subclass of | - |
| Description | Represents information, application, user of the Information System or even the IT equipment that needs to be secure. |

**Class a4s-language: Application**

| Subclass of | a4s-language: Asset |
| Description | A type of asset. |

**Class a4s-language: InputParameter**

| Subclass of | a4s-language: Asset |
| Description | A type of asset. |

**Class a4s-language: Operation**

| Subclass of | a4s-language: Asset |
| Description | A type of asset. |

**Class a4s-language: OutputParameter**

| Subclass of | a4s-language: Asset |
| Description | A type of asset. |
Class `a4s-language: PropertyContext`  

*Subclass of:* -  
*Description:* Represents a context in which the security property is applied.

Class `a4s-language: InStorage`  

*Subclass of:* `a4s-language: PropertyContext`  
*Description:* A type of property context.

Class `a4s-language: InTransit`  

*Subclass of:* `a4s-language: PropertyContext`  
*Description:* A type of property context.

Class `a4s-language: InUsage`  

*Subclass of:* `a4s-language: PropertyContext`  
*Description:* A type of property context.

Class `a4s-language: Scope`  

*Subclass of:* -  
*Description:* Represents a scope to the security problem specification.

Class `a4s-language: SecurityObjective`  

*Subclass of:* -  
*Description:* Represents a security objective.

Class `a4s-language: SecurityPolicy`  

*Subclass of:* -  
*Description:* Represents a security policy.

Class `a4s-language: TargetOfCertification`  

*Subclass of:* -  
*Description:* Represents the certified entity.

Class `a4s-language: Infrastructure-as-a-service`  

*Subclass of:* `Class a4s-language: TargetOfCertification`  
*Description:* A type of service.

Class `a4s-language: Platform-as-a-service`
Subclass of: Class a4s-language: TargetOfCertification
Description: A type of service.

Class a4s-language: Service-application

Subclass of: Class a4s-language: TargetOfCertification
Description: A type of service.

Class a4s-language: Software-as-a-service

Subclass of: Class a4s-language: TargetOfCertification
Description: A type of service.

Class a4s-language: Threat

Subclass of: -
Description: Represents a threat.

A.1.20. USDL-SEC

<table>
<thead>
<tr>
<th>Direct Imports</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indirect Imports</td>
<td>None</td>
</tr>
<tr>
<td>Short Description</td>
<td>Module containing concepts representing Security Features</td>
</tr>
</tbody>
</table>

Classes

Class foaf:Person

Subclass of: -
Description: Represents people.

Class skos:Concept

Subclass of: -
Description: Represents an idea or notion.

Class skos:ConceptScheme

Subclass of: -
Description: Represents an aggregation of one or more skos:Concept.
**Class usdl-sec:SecurityGoal**

*Subclass of:* -

*Description:* Represents a security topic.

*Examples:* It encompasses well known security concepts like Anonymity, Confidentiality, Privacy, Authentication etc.

**Class usdl-sec:SecurityMechanism**

*Subclass of:* -

*Description:* Represent a security solutions that can achieve a security goal. The solutions can be applied under three realization levels: the network level, the application level, and the service level (expressed with the relation usdl-sec:hasSecurityRealizationType).

**Class usdl-sec:Access_Control**

*Subclass of:* usdl-sec:SecurityMechanism

**Class usdl-sec:Auditing**

*Subclass of:* usdl-sec:SecurityMechanism

**Class usdl-sec:BiometricData**

*Subclass of:* usdl-sec:SecurityMechanism

**Class usdl-sec:Certificate**

*Subclass of:* usdl-sec:SecurityMechanism

**Class usdl-sec:CertificateExchange**

*Subclass of:* usdl-sec:SecurityMechanism

**Class usdl-sec:Certification**

*Subclass of:* usdl-sec:SecurityMechanism
Class usdl-sec: Challenge-Response

Subclass of: usdl-sec:SecurityMechanism

Description

Class usdl-sec: Checksum

Subclass of: usdl-sec:SecurityMechanism

Description

Class usdl-sec: Control_Code

Subclass of: usdl-sec:SecurityMechanism

Description

Class usdl-sec: Cryptography

Subclass of: usdl-sec:SecurityMechanism

Description

Class usdl-sec: Delegation

Subclass of: usdl-sec:SecurityMechanism

Description

Class usdl-sec: Digest

Subclass of: usdl-sec:SecurityMechanism

Description

Class usdl-sec: Filtering

Subclass of: usdl-sec:SecurityMechanism

Description

Class usdl-sec: KeyManagement

Subclass of: usdl-sec:SecurityMechanism

Description

Class usdl-sec: Load_Balancing

Subclass of: usdl-sec:SecurityMechanism

Description

Class usdl-sec: Logging

Subclass of: usdl-sec:SecurityMechanism

Description

Class usdl-sec: Monitoring

Subclass of: usdl-sec:SecurityMechanism

Description

Class usdl-sec: Obfuscation

Subclass of: usdl-sec:SecurityMechanism

Description
A
SSERT
4S
OA

D.3.4 − Second Version Of The ASSERT Ontology

Description

Class usdl-sec: Obligation
Subclass of: usdl-sec:SecurityMechanism
Description

Class usdl-sec: PasswordExchange
Subclass of: usdl-sec:SecurityMechanism
Description

Class usdl-sec: Pseudonym
Subclass of: usdl-sec:SecurityMechanism
Description

Class usdl-sec: Recommendation
Subclass of: usdl-sec:SecurityMechanism
Description

Class usdl-sec: Replication
Subclass of: usdl-sec:SecurityMechanism
Description

Class usdl-sec: Reputation
Subclass of: usdl-sec:SecurityMechanism
Description

Class usdl-sec: SharedSecret
Subclass of: usdl-sec:SecurityMechanism
Description

Class usdl-sec: Signature
Subclass of: usdl-sec:SecurityMechanism
Description

Class usdl-sec: Steganography
Subclass of: usdl-sec:SecurityMechanism
Description

Class usdl-sec: Token
Subclass of: usdl-sec:SecurityMechanism
Description

Class usdl-sec: Usage_Control
Subclass of: usdl-sec:SecurityMechanism
Description
Class `usdl-sec: SecurityProfile`

*Subclass of:* -

*Description:* Represents the root node of the model and the entry point from a service description language to USDL-SEC. This node can conceptually be similar to a pointer element from the general service description, to the security properties of the service.

Class `usdl-sec: SecurityTechnology`

*Subclass of:* -

*Description:* Represents a set of concrete implementations and tools that realizes the Security Measures.

*Examples:* The encryption on the network level is implemented by IPSec.

**Object Properties**

**ObjectProperty `usdl-sec: hasSecurityGoal`**

*Subproperty of:* -

*Description:* The property connects a Security Profile with an high-level Security Goal.

**ObjectProperty `usdl-sec: hasSecurityProfile`**

*Subproperty of:* -

*Description:* The property binds a USDL service description with its USDL-SEC security profile.

**ObjectProperty `usdl-sec: hasSecurityRealizationType`**

*Subproperty of:* -

*Description:* The property describes at which level a Security Mechanism operates. It ideally refers to a protocol stack, in order to understand in which layer a certain mechanism is active.

**ObjectProperty `usdl-sec: isRealizedByTechnology`**

*Subproperty of:* -

*Description:* The property expresses how a certain Security Mechanism is implemented, by means of a Security Technology.

**ObjectProperty `usdl-sec: securityMeasureType`**

*Subproperty of:* -

*Description:* The property expresses which Security Mechanism is used by a service (and thus, is associated with its Security...
**ObjectProperty** usdl:sec: usesMeasure

*Subproperty of:* -

*Description:* The property expresses which Security Mechanism is used by a service (and thus, is associated with its Security Profile).

### A.1.21. ASSERT-E

<table>
<thead>
<tr>
<th><strong>Direct Imports</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>activity</td>
</tr>
<tr>
<td>measurement</td>
</tr>
<tr>
<td>organisations</td>
</tr>
<tr>
<td>requirements</td>
</tr>
<tr>
<td>wsa</td>
</tr>
<tr>
<td>wsd1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Indirect Imports</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>agents</td>
</tr>
<tr>
<td>communication</td>
</tr>
<tr>
<td>dependency</td>
</tr>
<tr>
<td>math</td>
</tr>
<tr>
<td>math-algebra</td>
</tr>
<tr>
<td>math-calculus</td>
</tr>
<tr>
<td>mental-states</td>
</tr>
<tr>
<td>mereology</td>
</tr>
<tr>
<td>primitive-types</td>
</tr>
<tr>
<td>roles</td>
</tr>
<tr>
<td>statement</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Short Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>This module models concepts related to the ASSERT-E model, defined in deliverable D4.1 [21].</td>
</tr>
</tbody>
</table>

**Classes**

**Class assert-e:AbstractSecurityProperty**

*Subclass of:* communication:NonFunctionalRequirement

*Description:* A generic security requirement for the service under evaluation. It is used to define concrete security property included in the certificate [21].

*Examples:* Authenticity, Robustness, Confidentiality and so on.
Class assert-e:Action

**Subclass of:** activity:Activity  
**Description** An Action is the content of a state of a Symbolic Transition System and it is an activity performing a wsdl:InterfaceOperation.  
**Examples:** In the example reported in section 4.1.1, the initial state of the STS has the REC_Req action as content. Such action is an activity performing an upload operation.

Class assert-e:ASSERT-E

**Subclass of:** communication:Document  
**Description** An evidence-base certificate in the ASSERT4SOA framework. An ASSERT-E aims to i) prove that some test-related assurance activities have been carried out on a service, and ii) specify which security properties these activities were meant to support [21]. An ASSERT-E includes some assertions (ASSERT-E_Assertion).

Class assert-e:ASSERT-E_Assertion

**Subclass of:** communication:Assertion  
**Description** An assertion stating some SecurityStatement made by a CertificationAuthorityAgent acting as legal representative.

Class assert-e:Attribute

**Subclass of:** math:Pair  
**Description** A name-value pair denoting attributes.

Class assert-e:Authenticity

**Subclass of:** assert-e:AbstractSecurityProperty  
**Description** The need of ensuring that the users or objects are genuine [21].

Class assert-e:AuthorisedLeadAppraiser

**Subclass of:** organisations:OrganisationalRole  
**Description** Who is in charge of reporting the results of the appraisals they conduct.

Class assert-e:BooleanNominalScale

**Subclass of:** measurement:NominalScale  
**Description** A nominal scale of measure that has only two possible labels ("Passed" and "Failed", or "Ok" and "Not Ok" or "0" and "1") to
indicate that a measure has been successful or not.

**Class assert-e:CertificationAuthority**

*Subclass of:* mental-states:Public_Body  
*Description:* An entity trusted by one or more users to create and assign certificates [5].

**Class assert-e:CertificationAuthorityAgent**

*Subclass of:* mental-states:CognitiveAgent  
*Description:* A cognitive agent member of a CertificationAuthority organisation.

**Class assert-e:CertificationAuthorityAgentBelief**

*Subclass of:* mental-states:Belief  
*Description:* Something a CertificationAuthorityAgent believes in.

**Class assert-e:ClassAttribute**

*Subclass of:* assert-e:Attribute  
*Description:* Attribute that refers to a threat the service proves to counteract, a security function supporting the property or a specific characteristic of the security function that is certified [21].

*Examples:* Class attribute that has name="algorithm" and value="RSA".

**Class assert-e:Confidentiality**

*Subclass of:* assert-e:AbstractSecurityProperty  
*Description:* The need of limiting information access and disclosure to authorized users only [21].

**Class assert-e:ConcreteSecurityProperty**

*Subclass of:* assert-e:AbstractSecurityProperty  
*Description:* Let $p$ an abstract security property and $A$ the set of class attributes, a concrete security property $p_i$ is a pair $(p, A)$ [21].

*Examples:* Integrity using an RSA algorithm (see ClassAttribute).

**Class assert-e:EvaluationBody**

*Subclass of:* mental-states:Public_Body  
*Description:* Laboratory carrying out evaluations [5]. It is a public body accredited by a National Authority.
Class assert-e:EvaluationBodyAgent

Subclass of: mental-states:CognitiveAgent
Description A cognitive agent member of an EvaluationBody organisation.

Class assert-e:EvaluationBodyAgentBelief

Subclass of: mental-states:Belief
Description Something an EvaluationBodyAgent acting as AuthorisedLeadAppraiser believes in.

Class assert-e:FunctionalityTest

Subclass of: assert-e:Test
Description Tests including functionality tests based on valid inputs [21].

Class assert-e:ImplementationBasedModel

Subclass of: assert-e:WSCLBasedModel
Description Service model extending the WSCL-model with information about the stateful service implementation [21].

Class assert-e:Integrity

Subclass of: assert-e:AbstractSecurityProperty
Description The trustworthiness of information resource [21].

Class assert-e:LegalRepresentative

Subclass of: organisations:OrganisationalRole
Description Who has the official standing to act on behalf of an entity (e.g. a company or an organisation) or another person.

Class assert-e:NationalAuthority

Subclass of: mental-states:PublicBody
Description Public body responsible for accrediting certification authorities and evaluation bodies to be trusted to perform their activities (issue certificates and carry out evaluations).

Class assert-e:NonRepudiation

Subclass of: assert-e:AbstractSecurityProperty
Description Implies that one party of a transaction cannot deny having received a message nor can the other party deny having sent a message [21].
<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>assert-e:PenetrationTest</td>
<td>Tests based on well-known attacks [21].</td>
</tr>
<tr>
<td>assert-e:Robustness</td>
<td>The ability of a computer system to cope with errors during execution or the ability of an algorithm to continue to operate despite abnormalities in input, calculations, and the like [21].</td>
</tr>
<tr>
<td>assert-e:RobustnessTest</td>
<td>Tests based on invalid and malformed input and stress and load tests [21].</td>
</tr>
<tr>
<td>assert-e:SecurityStatement</td>
<td>Assertion on security functionalities of a web service $s$ in the form &quot;$s$ has security property $P$&quot;.</td>
</tr>
<tr>
<td>assert-e:ServiceModel</td>
<td></td>
</tr>
<tr>
<td>assert-e:State</td>
<td>A state of a Symbolic Transition System (STS), represented by a sequence having an assert-e:Action as content and another State as next element of the sequence.</td>
</tr>
<tr>
<td>assert-e:SymbolicTransitionSystem</td>
<td>A tuple $\langle S, s_0, V, I, Act, \rightarrow \rangle$ where $S$ is a set of states, $s_0$ is the initial state, $V$ is the set of internal variables, $I$ is the set of interaction variables, $Act$ is the set of actions (web service operations), and $\rightarrow$ is the transition relation [21].</td>
</tr>
</tbody>
</table>

Examples
See the example reported in section 4.1.1.
**Class assert-e:Test**

*Subclass of:* -

*Description:* Test that can be carried out on a service to prove that certain properties hold. It specifies how to produce test cases.

**Class assert-e:TestAttribute**

*Subclass of:* assert-e:Attribute

*Description:* A name-value pair denoting an attribute of a test.

**Class assert-e:TestBasedEvaluation**

*Subclass of:* activity:Activity

*Description:* The inductive process followed by an Evaluation Body when proving a security property based on a test-based approach. It is supported by a test based evidence.

**Class assert-e:TestBasedEvidence**

*Subclass of:* -

*Description:* Evidence that tests carried out on a service have given certain results [21]. It is the proof supporting the ASSERT-E and is provided by an Evaluation Body after having generated and executed tests on a service.

**Class assert-e:TestCase**

*Subclass of:* communication:FunctionalRequirement

*Description:* It is composed of all values necessary for a complete execution and evaluation of the software under test. In ASSERT4SOA a test case $tc$ is a 4-uple $(Pr, HC, \{(I_1, EO_1), \ldots, (I_n, EO_n)\}, Po)$ where $Pr$ is the set of pre-conditions, $HC$ a set of hidden communications, $\{(I_1, EO_1), \ldots, (I_n, EO_n)\}$ a set of pairs (input, expected output), and $Po$ the set of post-conditions [21].

**Class assert-e:TestCaseMeasure**

*Subclass of:* measurement:Measure

*Description:* The process of execution of test case that, given a test case indicating an input an expected output, assigns to the execution of such a test one of two possible values: “1” (Successful) or “0” (Failed).

**Class assert-e:TestExecution**

*Subclass of:* activity:Activity
Description: The activity performed by an Evaluation Body when executing tests on services to prove that certain properties hold for that service. It is defined as performing some TestCaseMeasure and using test cases.

**Class assert-e:TestGeneration**

Subclass of: activity:Activity
Description: The activity performed by an Evaluation Body when generating test cases from tests and using test generation methods.

**Class assert-e:TestGenerationMethod**

Subclass of: math:Function
Description: Method driving and managing the test case generation process and defining a test generation model [21]. It is seen as a function having tests as domain and test cases as codomain.

**Class assert-e:TestGenerationModel**

Subclass of: 
Description: Model defined on the basis of the available service model specifying which input has to be submitted to which service operation and where to measure the corresponding output operation [21].

**Class assert-e:TestMetrics**

Subclass of: measurement:Measure
Description: Metrics used to evaluate the testing activities.

**Class assert-e:WSCLBasedModel**

Subclass of: assert-e:WSDLBasedModel
Description: Service model extending the WSDL model with information about the client-service interactions and the overall communication flow [21].

**Class assert-e:WSDLBasedModel**

Subclass of: assert-e:ServiceModel
Description: The basic model for a service that is generated from the information in the WSDL interface [21].
Object Properties

Object Property `assert-e:attributeName`

*Subproperty of:* `math:first`
*Description:* A text representing the attribute name.

Object Property `assert-e:attributeValue`

*Subproperty of:* `math:second`
*Description:* A text representing the attribute value

Object Property `assert-e:evaluatedBy`

*Subproperty of:* `-`
*Description:* Property relating a test based evidence to the metrics used to assess the testing activities.

Object Property `assert-e:hasClassAttribute`

*Subproperty of:* `-`
*Description:* Property relating a concrete security property to the class attributes defining it

Object Property `assert-e:hasHiddenCommunication`

*Subproperty of:* `mereology:hasComponent`
*Description:* Property relating a test case to the hidden communications.

Object Property `assert-e:hasTestAttribute`

*Subproperty of:* `-`
*Description:* Property relating a test to the test attributes related to it

Object Property `assert-e:measurableWith`

*Subproperty of:* `-`
*Description:* Property linking a concrete security property to tests that can be used to prove such a property holds

Object Property `assert-e:models`

*Subproperty of:* `roles:count-as`
*Description:* Property relating a service to the model considered when issuing the ASSERT-E
Object Property assert-a:propertyStrongerThan

Subproperty of:  math:follows
InverseProperty assert-a:propertyWeakerThan
Description Property denoting that a concrete security property implies another concrete security property. It is defined as subproperty of math:follows because a partial order can be seen as an ordered sequence.

Object Property assert-a:propertyWeakerThan

Subproperty of:  math:precedes
InverseProperty assert-a:propertyStrongerThan
Description Property denoting that a concrete security property is implied by another concrete security property.

Object Property assert-a:testStrongerThan

Subproperty of:  math:follows
InverseProperty assert-a:testWeakerThan
Description Property denoting that a test is stronger than another one, i.e. a certificate with the former is better than a certificate with the latter.

Object Property assert-a:testWeakerThan

Subproperty of:  math:precedes
InverseProperty assert-a:testStrongerThan
Description Property denoting that a test is weaker than another one

A.1.22.ASSERT-O

<table>
<thead>
<tr>
<th>Direct Imports</th>
</tr>
</thead>
<tbody>
<tr>
<td>activity</td>
</tr>
<tr>
<td>communication</td>
</tr>
<tr>
<td>organisations</td>
</tr>
<tr>
<td>wsd1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Indirect Imports</th>
</tr>
</thead>
<tbody>
<tr>
<td>mereology</td>
</tr>
<tr>
<td>math-calculus</td>
</tr>
<tr>
<td>math</td>
</tr>
<tr>
<td>agents</td>
</tr>
<tr>
<td>statement</td>
</tr>
<tr>
<td>roles</td>
</tr>
<tr>
<td>dependency</td>
</tr>
<tr>
<td>mental-states</td>
</tr>
</tbody>
</table>

D.3.4 – Second Version Of The ASSERT Ontology 151 / 165
### Short Description

The ASSERT4SOA ontology for the Ontology-based certificates (ASSERT-O).

### Classes

#### Class `assert:o:OntologyBasedEvaluation`

- **Subclass of:** `activity:Activity`
- **Description:** An activity using a OWL Reasoner to perform a deduction starting from an OWL description of the security solution implemented in a SOA service.

#### Class `assert:o:OWLOntologyReference`

- **Subclass of:** `agents:Object_Identifier`
- **Description:** A document containing an OWL Ontology

#### Class `assert-o:SecurityPropertyReference`

- **Subclass of:** `assert-o:OWLOntologyReference`
- **Description:** A document containing an OWL Ontology being the definition of a Security Property.

#### Class `assert-o:CALegalRepresentativeAssertion`

- **Subclass of:** `Communication:Assertion`
- **Description:** An assertion made by an agent playing the role of the Legal Representative of a Certification Authority.

#### Class `assert-o:ASSERT-O`

- **Subclass of:** `communication:Document`
- **Description:** A document carrying, among other things, some conformance expression about a SOA service available from an endpoint and a security property definition.

#### Class `assert-o:ConformanceExpression`

- **Subclass of:** `communication:Qualificatory_Expression`
- **Description:** An expression stating that SOA service available from an endpoint conforms to a security property definition.

#### Class `assert-o:OWLReasoner`

- **Subclass of:** `math:Object_Identifier`
- **Description:** An identifier uniquely identifying an OWL Reasoner.
Class assert-o:EBAAuthorisedLeadAppriserBelief

Subclass of: mental-states:Belief
Description: A belief hold by an agent playing the role of Authorised Lead Appriser in a Evaluation Body.

Class assert-o:CAAgent

Subclass of: mental-states: CognitiveAgent
Description: A (cognitive) agent being member of a Certification Authority.

Class assert-o:EBAAgent

Subclass of: mental-states: CognitiveAgent
Description: A (cognitive) agent being member of an Evaluation Body.

Class assert-o:EvaluationBody

Subclass of: mental-states: Private_Legal_Person
Description: A private legal person acting as Evaluation Body.

Class assert-o:CertificationAuthority

Subclass of: mental-states: Public_Body, organisations: PredefinedOrganisation
Description: A public body acting as certification authority.

Class assert-o:AuthorisedLeadAppriser

Subclass of: organisation: OrganisationalRole
Description: The organisational role in a Evaluation Body having the goal to assure that the conclusions of an apprise is aligned with the state of the affairs.

Class assert-o:LegalRepresentative

Subclass of: organisation: OrganisationalRole
Description: The organisational role having the goal to represent the interests and oversees the legal affairs of an organisation.
ASSERT4Soa

Object Properties

ObjectProperty assert-o:conformsTo

Subproperty of: communication:hasQualification
Description Reference to the qualification of an entity

A.1.23. SeMF

<table>
<thead>
<tr>
<th>Direct Imports</th>
</tr>
</thead>
<tbody>
<tr>
<td>activity</td>
</tr>
<tr>
<td>communication</td>
</tr>
<tr>
<td>dependency</td>
</tr>
<tr>
<td>math</td>
</tr>
<tr>
<td>mental-states</td>
</tr>
<tr>
<td>mereology</td>
</tr>
<tr>
<td>organisation</td>
</tr>
<tr>
<td>roles</td>
</tr>
<tr>
<td>wsdl</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Indirect Imports</th>
</tr>
</thead>
<tbody>
<tr>
<td>agents</td>
</tr>
<tr>
<td>math-calculus</td>
</tr>
<tr>
<td>statement</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module containing concepts used to model SeMF, as described in [11], defined through the concepts included in the top ontology.</td>
</tr>
</tbody>
</table>

Classes

Class semf:Action

Subclass of: Organisations:OrganisationalRole
Description the class representing actions performed by SeMF agents
Examples: send-login, rec-login

Class semf:Agent

Subclass of: -
Description the class representing the agents’ knowledge in SeMF agents
Examples: set Users of SeMF agents

Class semf:AgentKnowledge

Subclass of: -
Description the class representing the set concept of SeMF, i.e. the set of traces that the agent P believes possible within the system.
Examples:

**Class semf:System**

*Subclass of:* -  
*Description:* the class representing a SeMF system, defined as a tuple \( S=(\Sigma, P, B, W, V) \) consisting of a set of agents \( P \), a language \( B \) over an alphabet of actions \( \Sigma \), a set \( V \) of agent’s local view, and a set \( W \) of agents’ initial knowledge.

*Examples:* system including a client \( C \), a secure storage service \( S \) and its subcontractor \( Sub \) managing persistence of data on behalf of the storage service (see example in section 5.1.7)

**Class semf:Trace**

*Subclass of:* -  
*Description:* the class representing the set of all finite sequences of SeMF actions (traces), relying on the `math:Sequence` concept

*Examples:* Set \( \Sigma^* \) of all finite sequences of element of \( \Sigma \), including the empty sequence \( \varepsilon \) (see example in section 5.1.3)

### A.1.24.ASSERT-M

<table>
<thead>
<tr>
<th>Direct Imports</th>
</tr>
</thead>
<tbody>
<tr>
<td>roles</td>
</tr>
<tr>
<td>communication</td>
</tr>
<tr>
<td>semf</td>
</tr>
<tr>
<td>wsdl</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Indirect Imports</th>
</tr>
</thead>
<tbody>
<tr>
<td>mereology.owl</td>
</tr>
<tr>
<td>math-calculus.owl</td>
</tr>
<tr>
<td>math.owl</td>
</tr>
<tr>
<td>agents.owl</td>
</tr>
<tr>
<td>statement.owl</td>
</tr>
<tr>
<td>mental-states.owl</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module containing concepts used to define ASSERT-M, as defined in [11], defined through the concepts included in the top ontology.</td>
</tr>
</tbody>
</table>

**Classes**

**Class assert-m:Action**

*Subclass of:* agents:Action  
*Description:* the class representing actions performed by Agents in the ASSERT-M system model. Actions are defined as having an
originator, a recipient, a list of parameters, and can be linked to interface operations in a WSDL file

Examples: storeSend, StoreReceive

Class assert-m:ASSERT-M

Subclass of: communication:Document
Description: the concept representing the ASSERT-M, as being a document representing a given ASSERT-M model, that proves that a service satisfies a given SecurityPropertyInstance

Class assert-m:ASSERT-M_Model

Subclass of: assert-m:SystemModel
Description: A system model, as defined in SeMF [11], abstracting a verification model, to be included in the ASSERT-M

Class assert-m:CertificationAuthority

Subclass of: mental-states:Legal_Person
Description: the legal person declaring that a service has a certain security property

Class assert-m:CorrectBehaviour

Subclass of: assert-m:SystemBehaviour
Description: the class representing the subset of sequence of actions in the system behaviour that are correct

Class assert-m:FormalDefinition

Subclass of: assert-m:SecurityProperty
Description: the class representing formal definitions of security properties, seen as constraints over the sequence of actions that can be performed by a system
Examples: auth(action1,action2,agent)

Class assert-m:InitialKnowledge

Subclass of: math:Set
Description: the set of sequence of actions initially known and not explicitly excluded by an agent

Class assert-m:LocalView

Subclass of: math:Set
Description: the set of sequences of actions observable by an agent
Class assert-m:SecurityProperty  
Subclass of: math:MathematicalProperty  
Description: the security property comprises both formal definitions and security property instances, and it is seen as a particular type of mathematical property over a system model. Security properties as such are not usually instanced, but are a placeholder to group security property instances and formal definitions.

Class assert-m:Proof  
Subclass of: mental-states:Deduction  
Description: the proof is the deduction process that proves the security property instance to the verification model.

Class assert-m:SecurityPropertyInstance  
Subclass of: assert-m:SecurityProperty  
Description: the concept of security property, realizing a formal definition, with the specific actions from the certification model of the system.

Examples:  
precede-wrt-phase(store(S; C; path; content; device), respRetrieveSend(S; C; path; content), ∃s (store(S; C; path; data; device)); [t])

Class assert-m:Sequence_of_Actions  
Subclass of: math:Sequence  
Description: a generic sequence containing only assert-m:Action individuals, used to model other concepts in the ontology (system behaviour, local view, and others).

Examples:  
send(C; S; d1), ..., store(Sub; g(d1)), send(C; S; d1), ..., store(Sub; g(d1))

Class assert-m:Service  
Subclass of: wsdl:Service  
Description: the concept of service in this context is both bound to its WSDL interface (from which the system model is derived), and considered as being an agent in the system model itself.

Class assert-m:SystemBehaviour  
Subclass of: math:Set  
Description: the set of sequence of actions defining a system behaviour.
Class `assert-m:SystemModel`

Subclass of: -

Description: as in [11], the model representing the target system, comprising the set of agents, the set of actions performed by agents, the system’s behaviour, the initial knowledge and the local view of agents.

Class `assert-m:Verification_model`

Subclass of: `assert-m:SystemModel`

Description: a system model at an abstraction level sufficiently detailed to prove the target security property.

Examples:

Class `assert-m:WSDL_interface_model`

Subclass of: `assert-m:SystemModel`

Description: abstraction of the verification model, that relates to the WSDL description of the target service and, if necessary, to additional (service internal) actions.

ObjectProperties

Object Property `assert-m:certified_by`

Subproperty of: `communication:stated_by`

Description: this property represents the trust of a certification authority into a security property.

Object Property `assert-m:excluded_by`

Subproperty of: `mental-states:believed_by`

Description: used to mark sequence of actions as explicitly excluded from the initial knowledge by an agent.

Object Property `assert-m:hasOriginator`

Subproperty of: `agents-canPerform, mereology:hasComponent`

Description: this property links an Action to the agent that performs it.

Object Property `assert-m:hasParameters`

Subproperty of: `mereology:hasComponent`
**ASSERT4Soa**

*Description* it specifies the list of parameters of an action

**Object Property assert-m:hasRecipient**

*Subproperty of:* mereology:hasComponent
*Description* the property specifies the recipient of an action, if any

**Object Property assert-m:initially_known_by**

*Subproperty of:* mental-states:believed_by
*Description* used to mark a sequence of actions as included in the initial knowledge by an agent

**Object Property assert-m:models**

*Subproperty of:* roles:count-as
*Description* the property binds the Service to its system model

**Object Property assert-m:realizes**

*Subproperty of:* -
*Description* used to bind the security property instance to its formal definition

**A.1.25. Topology**

<table>
<thead>
<tr>
<th>Direct Imports</th>
</tr>
</thead>
<tbody>
<tr>
<td>math</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Indirect Imports</th>
</tr>
</thead>
<tbody>
<tr>
<td>mereology</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A module introducing concepts from topology.</td>
</tr>
</tbody>
</table>

**Class math-topology:Neighborhood**

*Subclass of:* math:Set
*Description* For a point, an open set containing that point.

**Class math-topology:Space**

*Subclass of:* math:Set
*Description* The general concept of mathematical space

**Class math-topology:TopologicalSpace**

*Subclass of:* math:Space
Description

Object Properties

ObjectProperty math-topology:covers

Subproperty of: mereology:hasLocation

Description

ObjectProperty math-topology:overlaps

Subproperty of: mereology:hasLocation

Description

A.1.26. WSA

<table>
<thead>
<tr>
<th>Direct Imports</th>
</tr>
</thead>
<tbody>
<tr>
<td>organisation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Indirect Imports</th>
</tr>
</thead>
<tbody>
<tr>
<td>mereology</td>
</tr>
<tr>
<td>agents</td>
</tr>
<tr>
<td>primitive-types</td>
</tr>
<tr>
<td>math</td>
</tr>
<tr>
<td>statement</td>
</tr>
<tr>
<td>dependency</td>
</tr>
<tr>
<td>communication</td>
</tr>
<tr>
<td>math</td>
</tr>
<tr>
<td>roles</td>
</tr>
<tr>
<td>math-calculus</td>
</tr>
<tr>
<td>mental-states</td>
</tr>
<tr>
<td>wsdl</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module containing concepts used within the W3C’s Web Service Architecture [8].</td>
</tr>
</tbody>
</table>

Class wsa:MessageProduction

Subclass of: agents:Action

Description

Class wsa:Address

Subclass of: agents:Agent_Appelation

Description
Class `wsa:Body`

*Subclass of:* agents:Object  
*Description*

Class `wsa:Envelope`

*Subclass of:* agents:Object  
*Description*

Class `wsa:Header`

*Subclass of:* agents:Object  
*Description*

Class `wsa:MessageConsumption`

*Subclass of:* agents:Perception  
*Description*

Class `wsa:Message`

*Subclass of:* agents:Signal  
*Description*

Class `wsa:MessageSequence`

*Subclass of:* math:Sequence  
*Description*

Class `wsa:MessageTransport`

*Subclass of:* organisation:OrganisationalRole  
*Description*

Class `wsa:Receiver`

*Subclass of:* organisation:OrganisationalRole  
*Description*

Class `wsa:Sender`

*Subclass of:* organisation:OrganisationalRole  
*Description*
Object Properties

ObjectProperty wsa:hasReceiver

Subproperty of: mereology:hasComponent
Description

ObjectProperty wsa:hasSender

Subproperty of: mereology:hasComponent
Description

A.1.27.WSS

Direct Imports

Indirect Imports

Module containing terms used within WS-Security standard.

Class wss:SecurityToken

Subclass of: Description

Class wss:SecurityTimestamp

Subclass of: Description

Class wss:Signature

Subclass of: Description
Class `wss:SignatureConfirmation`

Subclass of:

Description

Class `wss:TokenReference`

Subclass of:

Description

Class `wss:Security`

Subclass of:

Description
Bibliography


[38] ASSERT4SOA Consortium, "Requirements for an ontology supporting certificates interoperability," 2011.


