ASSERT4SOA

Advanced Security Service cERTificate for SOA

CP - STREP - Grant No. 257351

Evaluation of the Certificate Ontology v1

Deliverable D3.3

.

Project Title	ASSERT4SOA – Advanced Security Service cERTificate for SOA
Grant Number	257351
Project Type	CP - STREP

Title of Deliverable	Evaluation of the Certificate Ontology v1
Subtitle of Deliverable	
Deliverable Number	3.3
Dissemination Level	PU
Internal Rev. No.	008
Contractual Delivery Date	30 November 2012
Actual Delivery Date	14 December 2012
Contributing WPs	WP3
Editor(s)	Claudia Pandolfo (ENG)
Author(s)	Stefania D'Agostini, Valentina Di Giacomo, Claudia Pandolfo, Domenico Presenza (ENG)
Reviewer(s)	Claudio Agostino Ardagna (UNIMI), Sigi Guerguens (SIT)

Executive Summary

This document describes the evaluation of the first version of the ASSERT Ontology, released in deliverable D3.2. The requirements for the ontology had been previously reported in deliverable D3.1. The second version of the ontology will be released in deliverable D3.4.
The objective of the evaluation phase is:

- to evaluate the impact of the ontology on the ASSERT4SOA Framework;
- to check and improve the quality of the ontology.

Evaluation activities have been performed in several phases of the project and provide feedback to the activity of development of the second version of the ASSERT4SOA Ontology and of components of the ASSERT4SOA Framework using the ontology.
This document illustrates the methodology followed during the evaluation of the ASSERT Ontology and the experiments that have been planned and performed during this phase. In particular, the document describes:

- the state of the art on ontology evaluation;
- the scope of the evaluation;
- the feedbacks of the evaluation;
- the experiments performed during the evaluation activity.

This document is structured as follows. The first chapter analyzes on the State of the Art on ontology evaluation, Chapter 2 illustrates the evaluation of the ASSERT Ontology, its lifecycle and feedbacks provided to other activities within the project. Chapter 3 describes in details the experiment executed during the evaluation and the final chapter reports the overall results of the evaluation.

Contents

Chapter 1

State of the art on Ontology Evaluation

This section presents an overview of the state of the art on ontology evaluation, on which the evaluation of the ASSERT4SOA Ontology is partly based.
In this document we use the term "Ontology" to denote "a formal, explicit specification of a shared conceptualization [1]".

1.1 Overview

The need for evaluation methodologies emerged very early in the field of ontology development and reuse because, like any engineering artefact, an ontology needs to be thoroughly evaluated.
With regard to ontology evaluation, the paper in [2] introduces the terms ontology verification and ontology validation. *Ontology verification* deals with checking that the ontology has been built correctly, i.e. that it correctly implements the requirements. *Ontology validation* is concerned with checking whether the meaning of the definitions really models the world for which the ontology is created [2]. In brief, ontology verification checks whether the ontology is built in the right way while ontology validation checks whether the right ontology is built. The paper in [3] is focused on ontology verification and, in particular on automatic approaches. The paper in [4] is a complementing work dealing with ontology validation.
Due to the declarative nature of ontologies, developers cannot just compile and run them like most other software artefacts. Therefore, no comprehensive and global approaches have been proposed to date to address the ontology evaluation problem, and no specific solution exists for security ontologies.
Surveys on evaluation methods are reported in [5], [6] and [7].
According to [8], ontology evaluation approaches are distributed in two major categories: a few principled approaches defining a set of well-studied, high level ontology criteria to be manually assessed, and automatic approaches covering

different evaluation perspectives and levels. Evaluation perspectives are defined by what is considered to be a good "quality" ontology.
Most approaches fall in one of the following categories:

- *Golden standard*: these approaches deal with comparing the ontology to a "Golden standard", usually a manually built in ontology (e.g. [9]). The quality of the ontology is expressed by its similarity to a manually built Gold Standard ontology. One of the difficulties encountered by this approach is that comparing two ontologies is rather complicated;
- *Data driven*: these approaches involve comparisons with a source of data (e.g. collection of documents) about the domain to be covered by the ontology (e.g. [10]). In this scenario the quality of the ontology is represented by its appropriateness to cover the topic of a corpus;
- *Application based*: these approaches based on using the ontology in an application and evaluating the results (e.g. [11]). In an application specific ontology evaluation the quality of an ontology is directly proportional to the performance of an application that uses it;
- *Human assessment*: in this type of approach the evaluation is done by humans who try to assess how well the ontology meets a set of predefined criteria, standard or requirements (e.g. [2] [12]).

Evaluation levels refer to the aspects of the ontology being evaluated. An ontology is a complex and multi-layered resource, therefore it is often more practical to focus on different ontology aspects separately rather than trying to evaluate the ontology as a whole. Individual levels have been identified by different authors but these definitions usually involve the following levels [3] [6]:

- *Vocabulary*: the set of all names in the ontology, be it URI referenced or literals. Evaluation on this level usually involves the comparison with sources of data related to the application domain;
- *Syntax*: ontologies can be defined in several surface syntaxes, as for instance, RDF/XML, OWL Abstract Syntax, Manchester Syntax and so on. The formal language chosen to describe the ontology must match the syntactic requirements of that language (e.g. use of correct keywords). This aspect tends to be evaluated automatically;
- *Taxonomy and other semantic relations*: although various relationships between concepts may be defined, the IS_A relationship is of particular importance and maybe focus of a specific effort. Other semantic relations can be evaluated separately usually including measures such as recall and precision. The paper in [3] introduces the *Structure* level, intended as the RDF graph described by the ontology;
- *Context or application level*: an ontology may be part of a larger collection of ontologies; in this case it is important to consider the context when evaluating it. Another important context is the application where the ontology will be used; rather than assessing the ontology per se it is often more practical to see how the application is affected by the use of the ontology;
- *Structure, architecture and design*: evaluation on this level means checking whether the ontology matches design principles and criteria identified before constructing it and that appropriate natural language documentation is provided. Structural concerns involve the organization of the ontology and its suitability for further development. This evaluation is interesting in manually constructed ontologies and is done by people.

Table 1, taken from [6], summarizes which approaches are commonly used for which of the above levels.

	Evaluation approach			
Level	**Golden standard**	**Data driven**	**Application based**	**Human assessment**
Vocabulary	X	X	X	X
Syntax	X			X
Semantic relations	X	X	X	X
Semantic relations	X	X	X	X
Context, application			X	X
Structure, architecture and design				X

Table 1 Approaches to ontology evaluation on different levels

1.1.1 Evaluation methodologies

OntoClean [13] is meant for application during ontology development. Its main function is the formal evaluation of the properties defined in the ontology by means of a predefined ideal taxonomical structure of meta-properties. Generally, a formal ontology evaluation, such as the one proposed in OntoClean, provides useful insights into semantic models. However, these insights are more structural and formally driven and do not allow to infer anything about the usability of an analysed ontology, therefore the usage of OntoClean for industry is limited.
OntoMetric [12] is a methodology that allows the users to measure the suitability of existing ontologies, regarding the requirements of their systems; it is an adaptation of the Analythic Hierarchy Process (AHP). The application of OntoMetric can only follow the ontology release. The main drawback is its usability: specifying the characteristics of an ontology is complicated and takes time; assessing its characteristics is quite subjective.
EvaLexon's [14] main goal is to evaluate at development time ontologies created by human beings from texts. Well-established recall/precision type metrics are used to evaluate how well ontology triples were extracted from a corpus. EvaLexon is meant for linguistic rather than conceptual evaluation.

1.1.2 Quality criteria and evaluation metrics

Ontology evaluation can target a number of different criteria. The paper in [3] examines several papers describing principles for good ontologies and presents the following concise list of ontology quality criteria:

- *Accuracy* [4]: Does the ontology capture and represent correctly aspects of the real world?
- *Adaptability* [4] (also named *expandability* in [2], *extensibility* in [15] and *flexibility* in [16]): Does the ontology anticipate its uses? Does it offer a conceptual foundation for a range of anticipated tasks? Can it be extended monotonically, i.e. without removing axioms?

- *Clarity* [15]: Does the ontology communicate effectively the intended meaning of the defined terms? Are the definition objective and independent of context? Does the ontology use definitions or partial descriptions? Are definitions documented?
- *Completeness* [2]/*competency*[17](also called *richness* and *granularity* in [4]): Is the domain of interest appropriately covered? Are competency questions defined and can the ontology answer them? Does the ontology include all the relevant concepts?
- *Computational efficiency* [14][16]: How easily and successfully can reasoners process the ontology? How fast can reasoning services (e.g. satisfiability, instance classification) be applied to the ontology?
- *Conciseness* [2]: Does the ontology include irrelevant axioms with regard to the domain to be covered? Does it include redundant axioms? Does it impose a *minimal ontological commitment* [15], that is does it specify the weakest theory possible and define only essential terms?
- *Consistency* [2]/*coherence* [15]: Do the axioms lead to contradictions (logical consistency)?
- *Organizational fitness* [16]: Is the ontology easily deployed within the organization? Was the proper process for creating the ontology used? Was it certified, if required? Does it meet legal requirements? Is it easy to access? Does it align to other ontologies already in use?

These criteria define a good ontology (not how good it is) and guide the creation and evaluation of an ontology; of course a good ontology will not perform equally well with regard to all of them. Hardly any of these criteria can be directly measured. Concrete evaluation methods should assess specific features of an ontology and provide indicators and metrics, that are usually related to more than one criterion.

The paper in [16] introduces three groups of measures on ontologies:

- *Structural dimensions*, related to the graph representation of an ontology, allow to measure topological and logical properties. According to [3] more than forty different structural metrics are currently described in literature. Examples of these dimensions (extracted from [16]) are[1]:
  - Measures for depth (the property related to the cardinality of paths in a graph), such as *Absolute depth* (M1), *Average depth*(M2) and *Maximal depth* (M3). The metrics can serve as indicators for the clarity of an ontology;
  - Measures for modularity, such as *Modularity rate* (M22) and *Module overlapping rate* (M23). These metrics can be used to evaluate the clarity and the adaptability of an ontology. The modularization of an ontology can also be related to the computational efficiency, because keeping simple and modularized ontology can decrease the reasoning time;
  - Measures for logical adequacy, such as *Consistency ratio* (M24), *Anonymous classes ratio* (M26), *Cycle ratio* (M27), *Class/relation ratio* (M29), *Axiom/class ratio* (M30). These metrics are made on constructs that affect the computational time, therefore can be regarded as indicators for the

[1] Structural dimensions described in [16] are denoted with an identifier consisting in the letter "M" followed by a number. This identifier is reported in brackets after the name of the dimension.

computational efficiency. The *Class/relation ratio* and *Axiom/class ratio* measures can be considered also as indicators for the clarity of an ontology.

- *Functional dimensions* are related to the intended use of an ontology and depend on the context. Specifications of a conceptualization, i.e. the main purpose of an ontology, are always approximate because the relationship between an ontology and a conceptualization is never straightforward. It is therefore necessary to measure the degree of such an approximation. The paper in [16] introduces functional measures that are variants of the ones proposed by [18], which uses an analogy with precision and recall, widely used in information retrieval: *O_Precision* (called coverage in [18]), *O_Recall* and *O_Accuracy;*
- *Usability-related dimensions* deal with the level of annotation of a given ontology.

The structure can also be investigated with regard to certain patterns. For example, the paper in [2] regards cycles within the taxonomic structure of the ontology as an error.

Since the majority of ontologies can be represented as a graph two measures identified by many authors (e.g. [4] [19]) are *semantic similarity* and *semantic distance*. Semantic distance measures how closely two nodes are topologically related in a graph, whereas semantic similarity captures to what extent two nodes might represent the same entity. Various approaches to calculate these dimensions have been proposed, falling mainly in two categories: edge-based methods and node-based methods.

Semantic similarity is considered by [19], together with the following measures, as a metric can be used to rank, and therefore compare, different ontologies by analyzing them separately:

- *Class Match Measure* (CMM), evaluating the coverage of an ontology for the given search term by looking for classes that have labels matching a search term exactly or partially (i.e. containing the search term);
- *Density Measure* (DEM), related to the degree of detail in the representation of a specific concept, including the number of subclasses, superclasses, relations and siblings;
- *Semantic Similarity Measure* (SSM), calculating how close the classes matching the search term are in the ontology;
- The *Betweeness Measure* (BEM), associated to the number of shortest paths passing through each node in the graph. If a class has a high betweeness value it can be assumed to be *central* in the ontology.

These metrics can also be used as indicators of the accuracy and completeness of an ontology.

The paper in [3] argues that there are different types of completeness. *Language completeness* is defined on a given ontology with regard to a specific ontology language and measures the ratio between the knowledge that can be expressed and the knowledge actually given. This is somehow related to the expressivity of the ontology language. Nevertheless, even though the expressivity of the language, e.g. OWL DL, is known, it defines an upper bound on the complexity applicable to reasoning tasks. Most ontologies do not use the

most expressive constructs, therefore the constructs used within the ontology allow to refine the used DL fragment and define a lower complexity bound.
Domain completeness is achieved when the ontology covers the complete domain of interest. This can be measured automatically only when the domain is accessible automatically and comparable to the ontology. Otherwise it is possible to use ontology learning techniques to extract an ontology from the corpus.
The requirements driven by competency questions and ontology use cases can be regarded as ontology evaluation criteria [4] related to completeness. In order to be used in automatic evaluation approaches, they need to be formalized in a query language that can be used with an appropriate tool [3].

Chapter 2

The ASSERT Ontology Evaluation

2.1 Overview

Evaluation activities have been performed in several phases of the project, as described in details in Section 2.3, and provide some feedback and assessment directly to the activity of development of the second version of the ASSERT4SOA Ontology and of components of the ASSERT4SOA Framework using the ontology. Results that will not be able to affect development activities can anyway be considered for future studies and research.

2.2 Scope of the evaluation

In this section the elements (e.g. ontology requirements, ontology itself etc.) considered during the evaluation activity are described. There are a number of elements that the evaluation activity have considered:

1. The ASSERT4SOA Ontology requirements, reported in deliverable D3.1 [20]. Ontology requirements have been identified by relying on the Competency Questions technique[21]. Competency Questions are natural language questions that the ontology to be built should be able to answer; in the ASSERT4SOA context they have been derived from the identified ontology use cases and from the models defined in Work Package 4 (evidence-based certificates) and Work Package 5 (model-based certificates);
2. The first version of the ASSERT4SOA Ontology, described in deliverable D3.2[22]. The second version of the ontology will be evaluated during its modelling, as already done with the first version (see Section 2.3);
3. The components belonging to the ASSERT4SOA Framework using the ASSERT4SOA Ontology. The evaluation activity focused on the first version of the ASSERT4SOA Framework released in deliverable D6.2 [23].

2.3 Evaluation activity lifecycle

The evaluation of the ASSERT4SOA Ontology has been performed during the whole ontology lifecycle. The paper in [7] identifies three important stages at the basis of the evaluation of an ontology:

1. Evaluation during the ontology pre-modelling stage;
2. Evaluation of the ontology in its modelling stage;
3. Evaluation of the ontology after its release.

The flow chart in Figure 1 shows the stages of the evaluation.

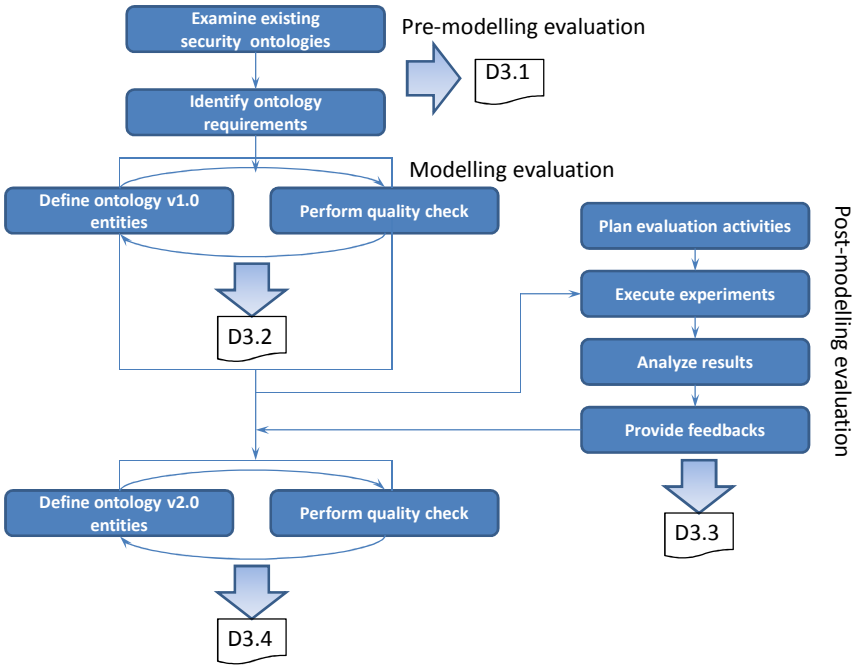


Figure 1 Flowchart of the evaluation process

The first stage includes the evaluation of the material that a human ontology engineer has at his disposal for building the actual ontology. With regard to the ASSERT4SOA Ontology, this activity has been performed in Task 3.1 of Work Package 3, during which the scope of the ontology has been determined, the use cases and user needs have been identified, and existing ontologies to be eventually reused have been examined [20].

Evaluating the ontology during its modelling stage takes place when the ontology engineer(s), at regular intervals, check the quality of the work done so far. This is done by performing consistency checks or other checks on logical errors or errors against the ontology language. Competency questions can be used as a technique to evaluate the ontology content.

The evaluation of the first version of the ASSERT4SOA Ontology during its definition has occurred as follows:

- during meetings regularly scheduled between ontology engineers in order to continuously check the work done;

- when modelling ASSERT-E and ASSERT-M specific concepts (defined respectively in [24] and [25]) through entities defined in the Top Ontology;
- by modelling and designing ontology modules basing on the ontology performance (estimated as order of magnitude of time required by the reasoner to classify);
- by examining ontology representational requirements reported in [20] as competency questions to verify whether all relevant concepts have been included in the ontology [22].

These activities will take place also during the definition of the second version of the ASSERT4SOA Ontology.

The evaluation of an ontology after its release is mainly done by people other than the ontology engineer(s) who released the ontology and this is the scope of Task 3.3 of Work Package 3, whose activity is subdivided into three main tasks:

1. Plan Evaluation activities,
2. Undertake Evaluation,
3. Report and feedback results and findings, as described in the present document.

The evaluation task provide some feedback and assessment mainly to the following activities:

- The definition of the second version of the ASSERT4SOA Ontology;
- The development of the second version of the ASSERT4SOA Framework, in particular of the Ontology Manager.

2.4 Evaluation of the ASSERT Ontology

The evaluation activity has focussed on with the following aspects:

- validate the ontology, i.e. check whether it implemented correctly the requirements reported in deliverable D3.1;
- evaluate the impact of the ontology when used in the ASSERT4SOA Framework;
- improve the quality of the ontology with regard to the quality criteria described in section 1.1.2;
- assess whether the ontology can be used to model concepts pertaining to particular domains/documents, such as Common Criteria, ASSERT4SOA Data Model (the so called A4S Model), A-SerDiQueL and ASLan++.

Chapter 3

Experiments

In order to extensively investigate the evaluation aspects reported in section 2.4, six experiments aimed at assessing these aspects, have been scheduled and executed. These experiments are described in details in the following.
All the evaluators involved in the experiments are experienced in OWL-DL. The evaluators participating in Experiment E2, dealing with Common Criteria, have a background knowledge on CC specifications. The users involved in the other experiments have been previously provided with documentation required to have a background on the focus of the experiment before running it.

3.1 Experiment E1 – Competency Questions

The objectives of this experiment are:

- validate the ontology requirements, i.e. the competency questions reported in the deliverable D3.1[20],
- check whether the ontology is representative for the ASSERT4SOA domain.

3.1.1Description

This experiment is based on a human assessment approach. The type of users that have been involved in its execution are the ones identified in deliverable D3.1.
Questionnaires have been given to the evaluators with the competency questions that the ontology should be able to answer. For this experiment the version ontology (v1.2.6) delivered within the ASSERT4SOA Software Framework v1.0 [26] was used. The evaluators have been asked to verify if the requirements were addressed and asked to evaluate whether it was possible to express the competency questions listed in [20] using the terminology defined by the ontology v1.2.6.
The experiment was conducted by focusing on the competency questions concerning the ASSERT-O Security Certificates [20] because the support on the

model underlying these certificates was easier to provide to the evaluators. ASSERT-E and ASSERT-M models will be evaluated during the definition of the second version of the ontology.

3.1.2Results

This section presents the results of experiment E1, that are the queries formulated by the evaluators to address ontology requirements. Following the organization of deliverable D3.1, requirements (and the associated queries) have been grouped in three categories, corresponding to three different exercises:

- Requirements related to ontology-based security properties;
- Requirements related to ontology-based certificates;
- Requirements related to ASSERT-O service model.

These results provide indications on whether the ontology is able to represent the knowledge of the intended domain and assess to a certain extent its domain completeness and clarity.

Requirements on ontology-based security properties

The goal of this exercise was to express the queries concerning requirement **PO1.FUN** (representation of ontology-based security properties) and informally described in D3.1 as DL Queries. The solution was approached assuming that security properties are described as OWL individuals within the ontology. The resulting OWL Class Expressions were:

- What is the name of a security property?

```
Class: queryNameOf_P
    EquivalentTo:
math:FunctionSymbol
and isMemberOfComponentOf value P
```

- Which are the parameters of a security property?

```
Class: queryParametersOf_P
    EquivalentTo:
math:Sequence
and followsComponentOf value P
```

- What is the definition of a security property?

```
Class: queryDefinitionOf_P
    EquivalentTo:
security:SecurityPropertySignature
and mereology:isComponentOf value P
```

In the queries above the symbol *P* is expected to be replaced with the URI of the security property of interest. In order to formulate the first two queries (`queryNameOf_P` and `queryParametersOf_P`) it was required to define two property chains (`isMemberOfComponentOf` and `followsComponentOf`) not directly available from the ontology, as follows:

```
ObjectProperty: isMemberOfComponentOf
    SubPropertyChain: mereology:isMemberOf o mereology:isComponentOf
```

```
ObjectProperty: followsComponentOf
    SubPropertyChain: math:follows o mereology:isComponentOf
```

Requirements on ontology-based certificates

The goal of this exercise was to verify the expressiveness of the ASSERT4SOA Ontology in answering the competency questions concerning the requirements **AO1.FUN**, **AO2.FUN**, **AO3.FUN**, **AO4.FUN** and **AO5.FUN** described in D3.1 and related to the representation of ontology-based certificates (i.e. ASSERT-O).

In the following the queries considered for each requirements and the resulting OWL Class Expressions are reported.

- **A01.FUN** (representation of ontology-based certificates)
  - Which are the security statements of an ASSERT-O?

```
Class: querySecurityStatementsOf_O
    EquivalentTo:
assert-o:ConformanceExpression
and inverse communication:isCarrierFor value O
```

  - What is the proof supporting an ASSERT-O security statement?

```
Class: queryProofSupporting_ST
    EquivalentTo:
assert-o: OntologyBasedProof
and (inverse math:isCodomainOf some (assert-
o:EBAuthorisedLeadAppriserBelief and inverse mental-
states:attitude value ST))
```

  The solution was approached assuming that the ontology-based certificates and the security statements are described as OWL individuals within the ontology. In the queries above the symbols *O* and *ST* are expected to be replaced with the URI of the ontology-based certificate and security statement of interest, respectively.

- **A02.FUN** (representation of security statements)
  - What is the service certified by a security statement?

```
Class: queryServiceCertifiedBy_ST
    EquivalentTo:
wsdl:Service and agents:isIdentifiedBy some (wsdl:Endpoint and
inverse communication:hasSubject value ST)
```

  - What is the security property asserted by a security statement?

```
Class: querySecurityPropertyAssertedBy_ST
    EquivalentTo:
assert-o:SecurityPropertyReference and
       inverse assert-o:conformsTo value ST
```

  The solution was approached assuming that the security statements are described as OWL individuals within the ontology. In the queries above the symbol ST is expected to be replaced with the URI of the security statement of interest.

- **A03.FUN (representation of ASSERT-O proofs)**

The solution was approached assuming that ASSERT-O proofs are described as OWL individuals within the ontology. In the query below the symbol `Proof` is expected to be replaced with the URI of the proof of interest.

- What is the reasoner used to verify a proof?

```
Class: queryReasonerUsedIn_Proof
    EquivalentTo:
 assert-o:OWLReasoner and inverse activity:useTools some
  (assert-o:OntologyBasedEvaluation
   and inverse activity:isPerformedIn value Proof)
```

During this experiment it was realized that it was not possible to formulate two competency questions of the A03.FUN requirement. In particular, it was not possible to express the queries "What is the service model used in a proof?" and "Which are the security controls considered in a proof?". Indeed, this can still be done by other means (e.g. programmaticaly) via the the `hasDomain` object property. In fact, through the `hasDomain` object property it is possible to retrieve the reference (`SecuritySolutionReference`) to the OWL ontology describing the security solution considered in a proof. In general an OWL ontology describing a security solution will contain (a) the import of the ontology containing the service model, (b) the import of the ontology containing the description of the security pattern and (c) a set of OWL assertions (e.g. `EquivalentTo` or `sameIndividualAs`) relating the two and representing altogether the security solution. In the ASSERT4SOA ontology *security solution* and *security control* are synonyms.

- **A04.FUN – Representation of Security Controls**
  As described in D3.1, A04.FUN refers to competency questions like *"What is the security pattern matched by a security control?"*, *"Which are the elements of a service implementation realising to a security control?"* etc. During this exercise it was realized that it is not possible to express this kind of queries. This was due to the particular choice made to represent security controls. Security controls are represented by means of a collections of OWL assertions relating security

- **A05.FUN – Representation of Security Patterns**
  A05.FUN refers to competency questions like "What is the security property provided by a security pattern?", "What is the expected behaviour of a security pattern?" etc. During this exercise it was realized that it is not possible to express this kind of queries since the security pattern concepts is not defined in the ontology. However, it has to be noted that, since Security Patterns are particular service models known to provide some security property, in the second version of the ASSERT4SOA Ontology it is recommended to introduce a class to explicitly denote them.

Requirements on ASSERT-O service model

This experiment verifies the expressiveness of the ASSERT4SOA Ontology in answering the competency questions regarding the model of a service in the context of an ASSERT-O. The variables representing individuals in the queries are specified after each requirement.

- **MO1.FUN - representation of service implementations**
  The variable *S* used in the queries is an instance of `ws:Service`.
  - What is the interface offered by a service implementation?
    ```
    Class: queryInterfaceOf_S
    EquivalentTo:
    wsdl:Interface and inverse wsdl:interface value S
    ```
  - Which are the functions implemented by service?
    ```
    Class: queryFunctionOf_S
       EquivalentTo:
    wsdl:Function and inverse agents:hasSkill value S
    ```
  - Which are the relationships existing between the roles of a service implementation?
    ```
    Class: RelationshipsOf_S
       EquivalentTo:
       Organisations:OrganisationalRelationship
       and mereology:isConstituencyOf value S
    ```
  - Which are the roles of a service implementation?
    ```
    Class: RolesOf_S
       EquivalentTo:
       organisations:OrganisationalRole
       and ((inverse dependency:hasDepender
            some RelationshipsOf_S)
            or (inverse dependency:hasDependee
            some RelationshipsOf_S))
    ```
  - What is the workflow of a service implementation?
    ```
    Class: WorkflowOf_S
       EquivalentTo:
       assert-o_ws:Behaviour
       and activity:isPerformedIn value S
    ```
  - Which are the resources needed by a service implementation?
    ```
    Class: queryResourcesOf_S
       EquivalentTo:
    agents:Object and inverse agents:hasResource value S
    ```
- **MO2.FUN - representation of service interface**
  The variable *I* used in the queries is an instance of `ws:Interface`, *O* is an instance of `wsdl:InterfaceOperation`, *D* is an instance of `dependency:Dependency`.
  - Which are the operations of an interface?
    ```
    Class: queryInterfaceOpsOf_I
       EquivalentTo:
    wsdl:InterfaceOperations and inverse wsdl:interfaceOperations
    value I
    ```
  - What is the message exchange pattern of an operation?

```
Class: queryMEP_of_O
   EquivalentTo:
wsdl:MessageExchangePattern and
inverse wsdl:messageExchangePattern value O
```

- What is the style of an operation?

```
Class: queryStyle_of_O
   EquivalentTo:
wsdl:OperationStyle and inverse wsdl:style value O
```

- Which are the formal input parameters of an operation?

```
Class: queryInput_of_O
   EquivalentTo:
wsdl:InterfaceMessageReference and inverse wsdl:input value O
```

- What is the output of an operation?

```
Class: queryInput_of_O
   EquivalentTo:
wsdl:InterfaceMessageReference and inverse wsdl:output value O
```

- Which are the possible faults of an operation?

```
Class: queryInput_of_O
   EquivalentTo:
wsdl:InterfaceFaultReference and
inverse wsdl:outputFault value O
```

- **MO3.FUN - representation of business objectives pursued by services**
  During this exercise it was realized that it is not possible to model this kind of queries since the business objective concept is not defined in the ontology.

- **MO4.FUN - representation of roles and organisational relationships**
  The variable *D* is an instance of `dependency:Dependency`.
  - What is the depender role?

```
Class: queryDependerOf_D
   EquivalentTo:
inverse dependency:hasDepender value D
```

  - What is the dependee role?

```
Class: queryDependerOf_D
   EquivalentTo:
inverse dependency:hasDependee value D
```

  - What is the type of relationship existing between two roles?
    Subclasses of Dependency concept represent the types of relationships between two roles. Currently those defined are: `Acquaintance`, `Communication`, `Concurrency`, `Cooperation`, `Incompatibility`, `Provision`, `Subordination`, *and* `Trust`. All these concepts are defined under the `organisations` namespace.

  - What is the dependum creating a dependency between a depender and a dependee?

```
Class: queryDependerOf_D
   EquivalentTo:
inverse dependency:hasDependum value D
```

- **MO5.FUN – Representation of Workflows**
  The variable *B* used in the queries is an instance of `assert-o_ws:Behaviour`.
  - What is the main activity of a workflow?
    ```
    Class: mainActivityOf_B
       EquivalentTo:
       activity:Activity
       and inverse math:hasContents value B
    ```
  - Is an activity primitive or structured?
    A structured activity is represented using the concept of `assert-o_ws:Behaviour`
  - Which are the activities composing a structured activity?
    ```
    Class: queryActivitiesOf_B
              EquivalentTo:
    inverse math:hasSequenceMember value B
    ```
  - What is the order of execution of sub-activities within a structured activity?
    The order is given by the `math:Sequence` superclass of `assert-o_ws:Behaviour`
  - Which are the communication activities within a workflow?
    The communications activities are defined using the class `activity:communicationActivity`
  - What is the remote service involved in a communication activity?
    ```
    Class: RemoteAgentOf_CA
       EquivalentTo:
       mental-states:CognitiveAgent
       and mental-states:aim some
             (mental-states:Goal
              and inverse activity:hasGoal some
                     (activity:Activity
                      and inverse math:hasContents some
                         (assert-o_ws:Behaviour
                          and inverse math:hasCodomain some
                          (Function
                           and activity:isPerformedIn value CA))))
    ```
  - What is the type of input/output messages exchanged by means of a communication activity?
    ```
    Class: RequestOf_CA
       EquiavalentTo:
       Communication:Request
       and inverse math:hasDomain some
              (math:Function
               and activity:isPerformedIn value CA)
    ```

- **MO6.FUN – Representation of constraints over resources**
  The variable `cntx` used in the queries is an instance of `activity:Activity`.
  - What is the resource used in a context?
    ```
    Class: ResourceOf_cntx
    ```

```
   EquiavalentTo:
   agents:PersistentItem
   and inverse activity:useTools value cntx
```

- What is the business objective pursued in a context?

```
Class: ObjectiveOf_cntx
   EquiavalentTo:
   mental-states:Goal
   and inverse activity:hasGoal value cntx
```

- What is the action performed by means of a resource in a context?

```
Class: ActionOf_cntx
   EquiavalentTo:
   math:Function
   and activity:isPerformedIn value cntx
```

- What is the subject of an action in a context?

```
Class: SubjectOf_cntx
   EquivalentTo:
   mental-states:CognitiveAgent
   and mental-states:aim some
         (mental-states:Goal
          and inverse activity:hasGoal value cntx)
```

3.1.3Comments and feedback

In this experiment the ontology has been assessed by the evaluators as rather complete from the point of view of ASSERT4SOA domain.
The ontology addresses quite well the requirements reported in deliverable D3.1, even if the execution of the experiment E1, revealed that some concepts (e.g. Security Patterns) are not yet defined in the version of the ontology used for the evaluation experiment. The lack of the synonyms Context and Business Objective, for the concepts Activity and Goal respectively, made difficult to answer to some of the competency questions. The answer was possible only once the equivalences were communicated to the experimenter.
Some object properties are missing or unclear as well (e.g. the property connecting a Service to its business objective).

3.2 Experiment E2 – Common Criteria

The objectives of this experiment are:

- Assess whether and to what extent the ASSERT4SOA Ontology can be used to model concepts pertaining to Common Criteria;
- Understand if the entities included in the ontology are clearly defined and communicate effectively the intended meaning of the defined terms.

3.2.1Description

This experiment is based on a mixed approach (see Section 1.1) because it has been performed by humans and executed considering as a term of comparison the Common Criteria specifications.

The Common Criteria for Information Technology Security Evaluation (CC) certification ISO standard (ISO/IEC 15408) has been defined to fulfil the needs of an international standard for affordable software security certification [27]. Common Criteria provides an unified process and a flexible framework to specify, design, and evaluate the security properties of IT products.
The purpose of this experiment was to model part of the Common Criteria specification (version 3.1) using the concepts defined in the ASSERT4SOA Top Ontology. The part of CC specifications to be modelled during the experiment has been chosen by the evaluators. To run the experiment, the first version of the ontology, released in March 2012 and described in [22], has been provided together with sufficient documentation.
In the following section, OWL entities resulted from this modelling have been described. Problems encountered during the modelling process are also discussed, along with possible solutions that have been proposed.

3.2.2Results

This section presents the results of experiment E2 by describing in details the OWL entities introduced during the experiment to model part of the CC specification. The next subsections explain how these entities have been used to model the Common Criteria key concepts:

- Target of Evaluation
- Protection Profile
- Security Target
- Security Functional Requirements
- Security Assurance Requirements
- Evaluation Assurance Level

The results of this experiment provide indications on the adaptability, clarity and domain completeness of the ontology.

Target of Evaluation Modelling

The product under certification, called Target of Evaluation (TOE) has been modelled as subclass of the `Object` class, as shown below. A TOE can be a software, a hardware, a firmware or all of them. Software, hardware and firmware are not defined in the Top Ontology, therefore there was the need to introduce them as subclasses of `Object`.

```
Class: TargetOfEvaluation
  SubClassOf: agents:Object
  EquivalentTo:
     (mereology:hasPart some cc:Firmware)and
     (mereology:hasPart some cc:Hardware)and
     (mereology:hasPart some cc:Software)
```

Figure 2 The `TargetOfEvaluation` OWL class

Protection Profile Modelling

The Protection Profile (PP) is a document, typically created by a user or user community, identifying security requirements for a class of security devices.

The focus of this experiment, was not on the modelling of this concept, that was introduced as a subclass of `Document`, without further defining it.

Security Target Modelling

The Security Target is the document identifying the security properties of the product under certification, i.e. the Target Of the Evaluation. In this experiment the Security Target has been modelled as depicted in Figure 3.
The class `CommonCriteriaSpecification` is simply defined as subclass of `Document` and represents the Common Criteria Specification followed to issue the certificate. The classes `EvaluationAssuranceLevel` and `SecurityRequirement` are further described below.
It has been noticed that it was not possible to express that the Security Target is the document representing the TOE security specification.

```
Class: SecurityTarget
  SubClassOf: communication:Document
  EquivalentTo:
     cc:conformsTo some cc:CommonCriteriaSpecification and
     cc:conformsTo some cc:ProtectionProfile and
     cc:conformsTo some cc:EvaluationAssuranceLevel and
     communication:isCarrierFor some cc:SecurityRequirementStatement

Class: SecurityRequirementStatement
  SubClassOf: communication:Qualificatory_Expression
  EquivalentTo:
     (communication:hasQualification some  cc:CCSecurityRequirement)
      and (statement:hasSubject only cc:TargetOfEvaluation)
```

Figure 3 The `SecurityTarget` OWL class and its related classes

Security Functional Requirements (SFRs) Modelling

Common Criteria Security Functional Requirements (SFRs) specify individual security functions which may be provided by a product (TOE). The Common Criteria presents a standard catalogue of such functions. In this experiment the class `SecurityRequirement` has been introduced as subclass of the class `Requirement` defined in the requirements module of the Top Ontology. The hierarchy of `SecurityRequirement`, including also OWL Classes modelling Security Assurance Requirements, is shown in Figure 4. A `SFRComponent` is a specific Security Functional Requirement described in the standard catalogue of SFRs reported in the Common Criteria Specification.

Security Assurance Requirement (SARs) Modelling

Security Assurance Requirements (SARs) describe the measures taken during development and evaluation of the product to assure compliance with the claimed security functionality (SFRs). Similarly to SFRs, The Common Criteria provides a catalogue of SARs. To model these requirements, the OWL classes `SecurityAssuranceRequirement` and `SARComponent`, representing a specific SAR described in the standard catalogue provided in Common Criteria, have been introduced.

Figure 4 The `SecurityRequirement` OWL class and its descendants, modelling Security Functional Requirements (SFRs) and Security Assurance Requirements (SARs).

Evaluation Assurance Level (EAL) Modelling

Evaluation Assurance Level (EAL) is the rating describing the degree of assurance followed during the evaluation. Each EAL corresponds to a package of SARs. Common Criteria lists seven levels, from EAL 1 (the most basic level) to EAL 7 (the most stringent level). In case the predefined requirements do not match the level of assurance required, the EAL might be augmented by adding additional assurance requirements.

The definition of the `EvaluationAssuranceLevel` OWL class and its related classes are reported in Figure 5.

```
Class: EvaluationAssuranceLevel
  SubClassOf: cc:AssurancePackage and measurement:Measure
  EquivalentTo:
   (measurement:Measure and math:hasDomain some cc:Evidence and
math:hasCodomain some measurement:NominalScale) and
mereology:hasComponent some cc:SARComponent

Class: Evidence
  SubClassOf: agents:PersistentItem
  EquivalentTo:
     (mereology:hasComponent some cc:ProductDocumentation)
 and (mereology:hasComponent some cc:SecurityTarget)
 and (mereology:hasComponent some cc:TargetOfEvaluation)
 and (mereology:hasComponent some cc:Test)
```

Figure 5 The `EvaluationAssuranceLevel` OWL class

The EAL has been modelled as subclass of both `AssurancePackage` and `Measure`. The latter has been considered because the evaluation assurance level can be seen as the result of a measure taking as input the evidence provided by the TOE developer/vendor and producing a nominal value (i.e. EAL1, EAL 2 and so on) as a result of the measure. The `Evidence` class considers what can be provided to the Evaluation Body. It has been defined as subclass of `PersistentItem`, because another suitable class has not been found.

The Evaluation Assurance Level is also an assurance package because it is a collection of SARs. Even if only the EAL is actually used in the ontology, all types of packages, i.e. collections of security requirements, foreseen in the Common Criteria have been introduced, as depicted in Figure 6.

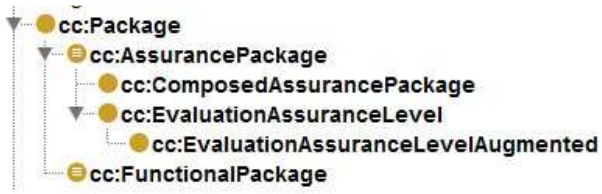


Figure 6 The Package class and its descendants

3.2.3Comments and feedback

After this experiment it has been pointed out that OWL entities belonging to the Top Ontology are clearly defined, mainly because most OWL classes are defined and there are only few primitive classes.
Nevertheless, it has been noticed that some concepts are missing and therefore the certification domain has been considered as not completely represented in the ontology. In particular, it was emphasized the lack of concepts related to certification and security aspects, such as *Certification Authority*, *Evaluation Body* and *Security Requirement*. These concepts are neither so general like the ones defined in the Top Ontology, nor so specific such as the ones pertaining to specific Certification schemes.
Assuming an in-depth knowledge of the already defined concepts, the extension of the ontology by adding required entities has been regarded as a rather simple task by the evaluators.

3.3 Experiment E3 – ASSERT4SOA Data Model

The objectives of this experiment are:

- Assess whether and to what extent the ASSERT4SOA Ontology can be used to model concepts pertaining to the ASSERT4SOA Data Model;
- Understand if the entities included in the ontology are clearly defined and communicate effectively the intended meaning of the defined terms.

3.3.1Description

This experiment is based on a mixed approach (see Section 1.1) because it has been performed by humans and executed considering as a term of comparison the ASSER4SOA Data Model.
The ASSERT4SOA Data Model describes the static structure of the information handled in the ASSERT4SOA Framework, including all the data types involved as the expected inputs and outputs of the operations provided by the ASSERT4SOA components. The first version of the ASSERT4SOA Data Model has been described in the deliverable D6.1 [28], but during the development activity it has been refined and the final version, used within the first ASSERT4SOA prototype v1.0, has been described in D6.2[29].
The purpose of this experiment was to model the ASSERT4SOA Data Model entities using the concepts defined in the first version of the ontology, released in March 2012 and described in [22]. The focus of this experiment was to model,

as a set of OWL 2 Class Expressions, the concepts expressed by the Data Model entities rather than their structures. The version of the Data Model considered in the experiment is the one described in deliverable D6.2. Both the ontology and the ASSERT4SOA Data Model have been provided to the evaluators with sufficient documentation.

3.3.2Results

This section presents the results of experiment E3 by describing how the entities of the ASSERT4SOA Data Model, depicted in Figure 7, can be formalized using the ASSERT4SOA Ontology.

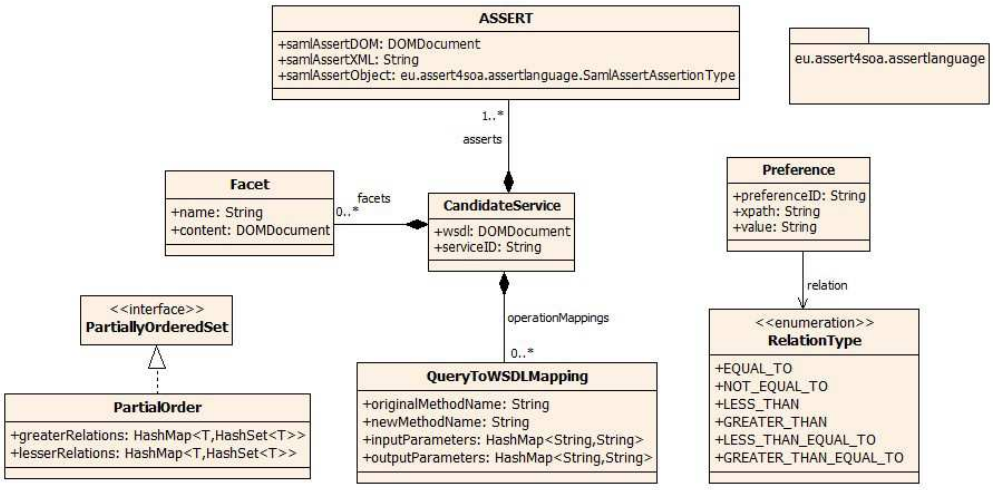


Figure 7 ASSERT4SOA Data Model [29]

In the following subsections, for each entity of the data model the concepts and the resulting OWL Class Expressions are reported.
The results of this experiment provide indications on the adaptability, clarity and domain completeness of the ontology.

ASSERT
In order to cover the various needs of the different components of the framework, this entity contains three different representations of an ASSERT SAML assertion [30]. As depicted in Figure 8, the ASSERT SAML assertion is represented as a DOM Object [31], as an XML string and as a Java Object (i.e. JAXB classes [32]) according to the current version of the ASSERT Language[33].
This entity has been modelled as the carrier of an ASSERT SAML assertion (see Figure 8). Indeed, the focus of this exercise was to use the ontology to model the concept expressed by the `samlAssertObject` element of the ASSERT entity. This element refers to the "SamlAssertAssertionType" entity of the ASSERT Language, that, as depicted in Figure 9, can contain different SAML statements. One of these statements can be an ASSERT SAML assertion statement (i.e. "Assert4SoaASSERTSAMLAssertionStatementType") expressing

that a security property holds for a given service. The resulting OWL Class Expressions are shown in Figure 8:

```
Class: ASSERT
EquivalentTo:
 communication:isCarrierFor some
   a4s_datamodel:Assert4SoaASSERTSAMLAssertionStatementType

Class: Assert4SoaASSERTSAMLAssertionStatementType
EquivalentTo:
 communication:Legal_Expression
  and (communication:hasSubject some wsdl:Service)
  and (communication:hasQualification some communication:QualityAttribute)
  and (communication:stated_by some assert-o:CertificationAuthority)
  and (communication:hasOutput some a4s_datamodel:CertificationProcess)
```

Figure 8 The `ASSERT` and `Assert4SoaASSERTSAMLAssertionStatementType` OWL classes

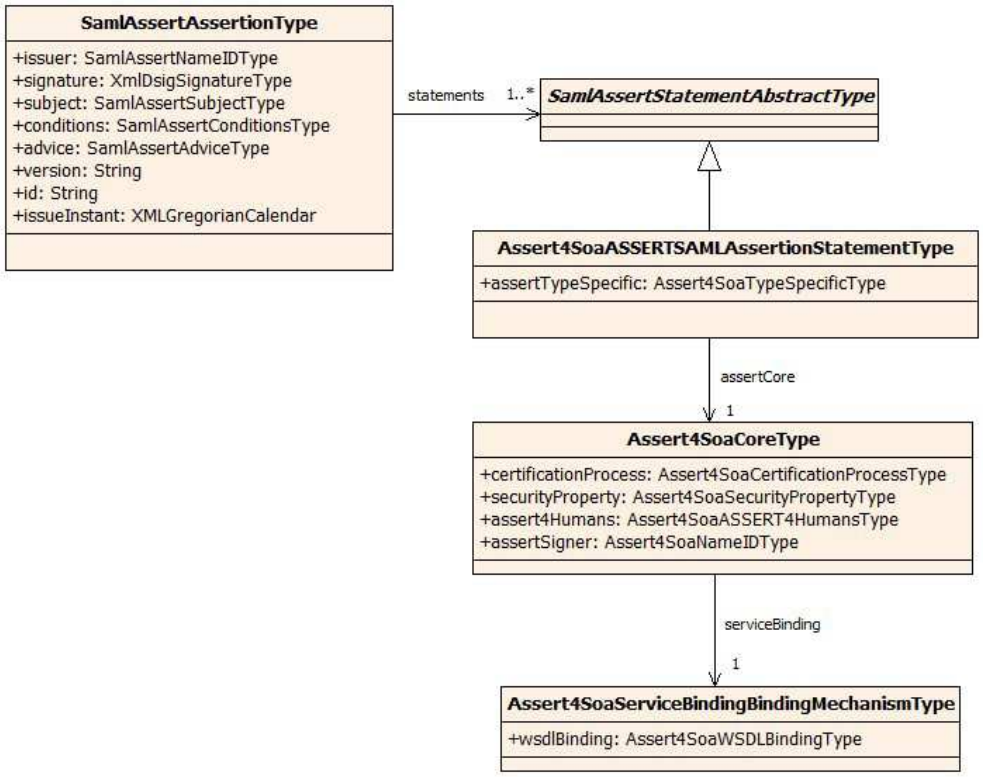


Figure 9 Relationships between some of the entities of the ASSERT4SOA Language

Since the ontology does not provide a class defining the concept of Security Property, it was not possible to specify which is the kind of Quality Attribute expressed in the ASSERT SAML assertion statement. It has been remarked

that also the concept of Certification Process is missing, therefore it was necessary to explicitly define it.
However, it has been emphasized that the reasoner correctly inferred that the ASSERT class is a subclass of communication:Medium.

CandidateService
It represents a service (providing some functionalities) having associated to it at least one ASSERT and, optionally, some Facet and a QueryToWSDLMapping objects. This entity has been modelled as a kind of Service (see Figure 10) that is characterised by other information associated to it (i.e. ASSERT, Facet, QueryToWSDLMapping). The resulting OWL Class Expressions were:

```
Class: CandidateService
EquivalentTo:
 wsdl:Service
 and (mereology:hasComponent some a4s_datamodel:ASSERT)
 and (mereology:hasComponent some a4s_datamodel:Facet)
 and    (mereology:hasComponent    some    a4s_datamodel:QueryToWSDLMapping)
SubClassOf:
 mereology:hasComponent max 1 a4s_datamodel:QueryToWSDLMapping
 mereology:hasComponent min 1 a4s_datamodel:ASSERT
```

Figure 10 The CandidateService OWL class

Facet
It represents a single defining aspect of an object (e.g. a description of different characteristics of a service: service behaviour description, service endpoint description etc.). This entity has been modelled as a Qualificatory_Expression (see Figure 11). The resulting OWL Class Expressions were:

```
Class: Facet
EquivalentTo:
 communication:hasSubject some wsdl:Service
 and communication:hasQualification some owl:Thing
SubClassOf
 communication:Qualificatory_Expression
```

Figure 11 The Facet OWL class

In a first step of this experiment, the Facet entity was modelled without explicitly defining it as a subclass of a Qualificatory_Expression. Since in the definition of a Facet the object properties communication:hasSubject and communication:hasQualification have been used, it was expected that the reasoner would have inferred that a Facet is a Qualificatory_Expression, but this inference was not made. Therefore, it is recommended to check (and eventually modify) the current definition of the Qualificatory_Expression class.

PartialOrder

It represents a partially ordered set formalizing the concept of ordering among the elements of a set. This entity has been modelled in two ways.

In a first modelling attempt, a partial order has been defined as an algebra (i.e. a `Set` characterised by an ordering operation) as depicted in Figure 12. The resulting OWL Class Expressions were:

```
Class: PartialOrder
EquivalentTo:
 math:Set
 and (math:hasOperation some math:OrderingFunction)
```

Figure 12 The `PartialOrder` OWL class

In a second attempt an example of partial order has been considered. In particular, a set S_1 characterised by an ordering relation R_1 has been chosen. The elements of the set S_1 and the relations between them are reported below (see also):

$PartialOrder\_1 = (S1, R1)$
$S1 = \{A, B, C, D, E, F, G\}$
$R1 = \{A > B, A > C, C > D, C > E, F > E\}$

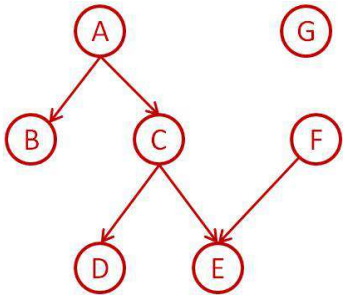


Figure 13 Relationships between the elements of the `PartialOrder_1`

The chosen partial order has been modelled defining its members and their relationships (see Figure 14). In particular, the elements of the set (i.e. A,B,C,E,E,F,G) have been modelled as classes instead as instances and the relationships among these elements have been modelled using the `math:precedes` object property. The resulting OWL Class Expressions were:

```
Class: S1
EquivalentTo:
 (mereology:hasMember some a4s_datamodel:A)
 and (mereology:hasMember some a4s_datamodel:B)
 and (mereology:hasMember some a4s_datamodel:C)
 and (mereology:hasMember some a4s_datamodel:D)
 and (mereology:hasMember some a4s_datamodel:E)
 and (mereology:hasMember some a4s_datamodel:F)
 and (mereology:hasMember some a4s_datamodel:G)
```

```
Class: A
EquivalentTo:
(math:precedes some a4s_datamodel:B)
 and (math:precedes some a4s_datamodel:C)

Class: B

Class: C
EquivalentTo:
(math:precedes some a4s_datamodel:D)
 and (math:precedes some a4s_datamodel:E)

Class: D

Class: E

Class: F
EquivalentTo:
 math:precedes some a4s_datamodel:E

Class: G
```

Figure 14 OWL classes modelling the chosen partial order example

It has been realized that the ontology permits to model the partial order concept in a more generic way by considering it as a Set with an ordering function. In this case, the reasoner inferred that a partial order is a subclass of `math:Algebra`. Moreover, it has been realized that the ontology permits also to model each partial order using the `precedes` (or `follows`) object properties on its members. In this case, the reasoner inferred that a partial order is a subclass of `math:Set`. The modelling choice will depend on the application context.

Preference

It represents the client's preferences on the security requirements but also with respect to other information (such as models, evidences etc...). This entity has been modelled as a `Request` (see Figure 15) made by a client regarding some requirements (functional and not functional). The resulting OWL Class Expressions were:

```
Class: Preference
EquivalentTo:
 communication:hasSubject some communication:Requirement
SubClassOf: communication:Request
```

Figure 15 The `Preference` OWL class

It has been remarked that the ontology does not provide an entity modelling the Client concept.

QueryToWSDLMapping

It represents the mapping of the operations and parameters present in the consumer query with the ones contained in the WSDL of the candidate service.

This entity has been modelled only adding a new class in the ontology. The resulting OWL Class Expressions were:

```
Class: QueryToWSDLMapping
```

However, during the experiment it has been realized that this entity can be defined using the standard constructs provided by OWL (e.g. equivalent classes).

RelationType

It represents the enumeration of relation types to evaluate the preferences. This entity has been modelled as a value in a defined set of `Evaluative_Expression` (see Figure 16). The resulting OWL Class Expressions are shown in Figure 16:

```
Class: RelationType
EquivalentTo:
 a4s_datamodel:EQUAL_TO
 or a4s_datamodel:GREATER_THAN
 or a4s_datamodel:GREATER_THAN_EQUAL_TO
 or a4s_datamodel:LESS_THAN
 or a4s_datamodel:LESS_THAN_EQUAL_TO
 or a4s_datamodel:NOT_EQUAL_TO

Class: EQUAL_TO
SubClassOf: communication:Evaluative_Expression

Class: GREATER_THAN
SubClassOf: communication:Evaluative_Expression

Class: GREATER_THAN_EQUAL_TO
SubClassOf: communication:Evaluative_Expression

Class: LESS_THAN
SubClassOf: communication:Evaluative_Expression

Class: LESS_THAN_EQUAL_TO
SubClassOf: communication:Evaluative_Expression

Class: NOT_EQUAL_TO
SubClassOf: communication:Evaluative_Expression
```

Figure 16 The `RelationType` OWL class

3.3.3Comments and feedback

From this experiment it has been deduced that concepts included in the ontology are clearly defined, even if some definition, e.g. the `Qualificatory_Expression`'s one, should be carefully checked because some expected inferences are not made by the reasoner.

It has been pointed out that it is usually possible to follow more than one modelling choice, often resulting in different inferences performed by the

reasoner. It is not clear if this is due to a non in-depth knowledge of the ontology by the evaluators or to inconsistent definitions.
Concepts associated to the ASSERT4SOA domain are well represented in the ontology, even if some important concept, such as Security Property, are not explicitly modelled. However, it was relatively simple to add missing concept, e.g. Certification Process, as extensions of entities already included in the ontology.

3.4 Experiment E4 – A-SerDiQueL

The objectives of this experiment are:

- Assess whether and to what extent the ASSERT4SOA Ontology can be used to model concepts pertaining to A-SerDiQueL;
- Understand if the entities included in the ontology are clearly defined and communicate effectively the intended meaning of the defined terms.

3.4.1 Description

This experiment was based on a mixed approach (see Section 1.1) because it has been performed by humans but executed considering as a term of comparison A-SerDiQueL specifications.
A-SerDiQueL is an adaptation of the Service Discovery Query Language – SerDiQueL – that was adopted in ASSERT4SOA for the specification of service discovery queries [34].
SerDiQueL is an XML-based language that allows expressing combinations of various characteristics of the services to be identified. More specifically this language supports the specification of structural, behavioural, quality, and contextual characteristics of services to be discovered. The amendments introduced to the SerDiQueL language allow the specification of service discovery criteria regarding ASSERTs.
The purpose of this experiment was to model concepts pertaining to A-SerDiQueL using the concepts defined in the first version of the ontology, released in March 2012 and described in [22]. The A-SerDiQueL reference language version used for the experiment is the one described in [34].

3.4.2 Results

This section presents some samples of the OWL classes resulted from experiment E4, along with some considerations on the expressiveness of the ASSERT4SOA Ontology to represent concepts pertaining to A-SerDiQueL. Some of the difficulties that were experienced in the modelling process are also discussed.
When modelling a language such as A-SerDiQueL in OWL, the first choice that has to be made is on what is the target of the model, in other words, to decide if the OWL classes will represent the structure of the language or the content of the possible queries that can be expressed in that language. As a first experiment, the focus has been put on the structure.

A SerDiQueL query is modelled as a container of all of the specific subqueries (structural, behavioural, etc), using the `mereology:hasComponent` relation of the mereology module of the A4S Ontology (Figure 17 shows some of the modelled classes).
For what concerns the StructuralQuery, it contains the WSDL specification of the requested service. As for now, the more representative class is the `wsdl:Service` concept, but it is recommended to add to the ontology a concept referring to the whole WSDL structure, not only its parts.
When the ordering of the parts of a query are relevant, the *math:Sequence* concept has been used. It showed to be fairly expressive, and allows to represent also complex structures.

```
Class: Query
    EquivalentTo:
        ((mereology:hasComponent some BehaviourQuery)
         and (mereology:hasComponent some ConstraintQuery)
         and (mereology:hasComponent some Parameter)
         and (mereology:hasComponent some StructuralQuery))
         and (agents:isIdentifiedBy some agents:Object_Identifier)

Class: Parameter
    SubClassOf:
        math:Pair

Class: StructuralQuery
    SubClassOf:
        wsdl:Service

Class: BehaviourQuery
EquivalentTo:
        BehaviourSubExpr
         or (math:Sequence
         and (math:hasContents some math:NegationSymbol)
         and (math:hasNext some BehaviourSubExpr))

Class: BehaviourSubExpr
    EquivalentTo:
        math:Sequence
         and ((math:hasContents some BehaviourCondition)
         and (math:hasNext some
            (BehaviourExpression
             and (math:hasNext only
                (math:EmptyList
                 or ((math:first only math:BinaryConnective)
                 and (math:second only BehaviourExpression)))))))

Class: BehaviourSubCondition
    EquivalentTo:
        math:Sequence
         and (math:hasContents only SequenceOfActions)

…
```

Figure 17 SerDiQueL query structure and some of its parts

Following the structural approach, as can be seen in the *BehaviourQuery* class, it emerged a difficulty in modelling the possible negation of the query. The

behaviour query is modelled as a *math:Sequence*, and the modelling of this type of construct highlighted the difficulty to express possible prefixes to a given Sequence, which has to be done explicitly. In the next version of the ontology, it is recommended to explore modifications to ease the modelling of this type of concepts.
An example of modelling A-SerDiQuel query structure is instead shown in Figure 18.

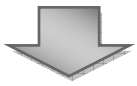
```
Class: AssertQuery
    EquivalentTo:
        ConstraintQuery
         and (agents:isIdentifiedBy some primitive-types:Text)
         and (mereology:hasComponent some ExternalExec)
         and (mereology:hasComponent some Matchmaker)
         and (mereology:hasComponent some Type)
         and (mereology:hasComponent some Weight)

Class: ConstraintQuery
    EquivalentTo:
        LogicalExpression
         or (LogicalExpression
         and (math:hasNext only
            ((math:first only math:BinaryConnective)
             and (math:second only LogicalExpression))))

Class: ExternalExec
    SubClassOf:
        primitive-types:Boolean

Class: Weight
    SubClassOf:
        primitive-types:PositiveNumber

Class: Matchmaker
    SubClassOf:
        primitive-types:Text

Class: Type
    SubClassOf:
        primitive-types:Text

…
```

Figure 18 A-SerDiQueL representation

Being SerDiQueL (as well as A-SerDiQuel) a query language that filters services based on a list of criteria, the modelling approach based on the structure of the language is only one option. One other option would be to use OWL Expression to list the conditions in the query, and thus to let the OWL reasoner identify the correct services. There would be a loss of generality in the model, but the expressiveness of the Ontology could be exploited more efficiently.
Generally speaking, A-SerDiQueL uses concepts that can be recognized easily in the ASSERT4SOA Ontology, such as parameters, WSDL, Services, actions and concepts specific to the domain of ASSERTS.
In Figure 19 there is an example of this type of approach: in the upper part of the figure there is the sample query in A-SerDiQueL (only a section), while in

the lower part there is a possible OWL Class Expression that represents it. Note that the *SOFT* type of the query has been translated with the *SubclassOf* construct, while a *HARD* type would be translated with the *EquivalentTo* construct.

```
<AssertQuery name="A1" type="SOFT">
   <LogicalExpression>
        <Condition relation="EQUAL-TO">
             <Operand1>
                   <AssertOperand facetName="Assert" facetType="Assert">
                        //AbstractASSERTStructure/ASSERTIssuer/Name
                   </AssertOperand>
             </Operand1>
             <Operand2>
                   <Constant type="STRING">Fraunhofer_SIT</Constant>
             </Operand2>
        </Condition>
   </LogicalExpression>
</AssertQuery>
```



```
Class: A1
SubclassOf:
       assert:ASSERT and assert:hasIssuer only Fraunhofer_SIT
```

Figure 19 Modelling A-SerDiQueL using OWL Expression queries

3.4.3Comments and feedback

The ontology has been considered rather complete to represent concepts pertaining to A-SerDiQueL even if not all of the classes used to write the sample are yet present in the ontology. In any case, it has been pointed out that no difficulty was encountered when adding missing concepts, therefore the ontology provides a good degree of extensibility.
This first version of the Ontology models top concepts, and then adopts a bottom up approach to define concepts related to ASSERT. One recommendation is to add those concepts generic for ASSERTs and not specific to one type in particular (e.g. ASSERT-M). Concepts such as security property categories, validity period and issuer of an ASSERT, are already present in the assert-specific modules of the ontology, but need to be identified and moved to the upper level of the ontology, so that they can be referred to by all the specific modules.
It was noted that the `math` module includes classes useful to model languages based on XML.

3.5 Experiment E5 – ASLan++

The objectives of this experiment were:

- assess whether and to what extent the ASSERT4SOA Ontology can be used to model concepts pertaining to ASLan++;

- understand if the entities included in the ontology are clearly defined and communicate effectively the intended meaning of the defined terms.

3.5.1Description

This experiment is based on a mixed approach (see Section 1.1) because it has been performed by humans but executed considering as a term of comparison the ASLan++ language [35]. ASLan++ was considered because it gave us the opportunity to have, at least, a partial assessment of the capability of the ASSERT4SOA Ontology to ground the semantics of the security related concepts delivered in the context of another EU FP7 research project i.e. the AVANTSSAR project (http://www.avantsaar.eu).
ASLan++ [35] is a formal language for specifying security-sensitive service-oriented architectures, the associated security policies, as well as their trust and security properties. The semantics of ASLan++ is defined by translation to ASLan, the low-level input language for the back-ends of the AVANTSSAR Platform.
The experiment is composed by a set of exercises performed by different participants. Each participant received a specification given in the ASLan++ language and was requested to try to formalise an equivalent specification using the terms provided by the ASSERT4SOA Ontology. Each participant returned the set of OWL class expressions proposed as equivalent specification. Moreover also returned feedbacks in the form of both description of the difficulties encountered using the ontology and suggestion for improvements/extension of the ontology.
The results of this experiment provide indications on the adaptability, clarity and domain completeness of the ontology.

3.5.2Results

This sections presents the results of the exercises performed in the context of experiments E5. The results are grouped by the targeted ASLan++ language component addressed. The first group deals with the introductory example. The introductory example was chosen because of the variety of ASLan++ constructs it presents. The second group deals with the ASLan++ Standard Prelude. This component was chosen because of it is imported by all definitions given in ASLan++. The third group deals with ASLan++ definition of security properties. This was selected to assess the suitability of the ASSERT4SOA Ontology to represent security properties.

Introductory example
This section presents how the introductory example presented in [36] (**Figure 20** ASLan++ introductory example from Figure 20) can be formalized using the ASSERT4SOA Ontology.

```
01) specification NSPK channel_model CCM

02) entity Environment {
03)      entity Session (A, B: agent) {
```

```
04)          entity Alice (Actor, B: agent) {
05)             symbols Na, Nb: message;
06)                body {
07)                   Na := fresh();

08)                   secrecy_goal secret_Na: Actor,B: Na;
09)                   Actor -> B: {Na.Actor}_pk(B);

10)                   B -> Actor: {Na.?Nb}_pk(Actor);

11)                   channel_goal Alice_authenticates_Bob_on_Na:
12)                   B *-> Actor: Na;
13)                   secrecy_goal secret_Nb: B,Actor: Nb;

14)                   Actor -> B: {Nb}_pk(B);
15)                   channel_goal Bob_authenticates_Alice_on_Nb:
16)                   Actor *-> B: Nb;
17)                }
18)      }

19)   entity Bob (Actor: agent) {
20)      symbols
21)         A: agent;
22)         Na, Nb: message;
23)      body {
24)         ?A -> Actor: {?Na.?A}_pk(Actor); % Bob learns A here!
25)         %secrecy_goal secret_Na:A,Actor: Na;
26)         Nb := fresh();
27)         secrecy_goal secret_Nb: A,Actor: Nb;
28)         Actor -> A: {Na.Nb}_pk(A);

29)         channel_goal Alice_authenticates_Bob_on_Na:
30)         Actor *-> A: Na;
31)         A -> Actor: {Nb}_pk(Actor);

32)         channel_goal Bob_authenticates_Alice_on_Nb:
33)         A *-> Actor: Nb;
34)      }
35)   }

36)   body {
37)      new Alice(A,B);
38)      new Bob(B);
39)   }
40)}

41)   body { % need two sessions for Lowe's attack
42)     any A B. Session(A,B);
43)     any A B. Session(A,B);
44)   }
45)}
```

Figure 20 ASLan++ introductory example from [36]

The goal of this exercise was to express row 07) as a set of OWL 2 Class Expressions. The answer was approached by trying to represent row 07) as an Activity. The resulting OWL Class Expressions is presented in Figure 21.

```
Class: create_Na
EquivalentTo:
   (hasGoal some Alice_Goal)
   and (performs some (fresh
```

```
                        and (math:hasCodomain some Na)))
SubClassOf: activity:Activity

Class: Alice_Goal
EquivalentTo:
mental-states:aimed_by some Alice

Class: Alice
SubClassOf: mental-states:CognitiveAgent

Class: fresh
EquivalentTo:
math:Function and (math:hasCodomain min 1 message)

Class: message
EquivalentTo:
   wsdl:messageInterfaceReference

Class: Na
SubClassOf: message
```

Figure 21 OWL 2 Class Expressions representing creation of a fresh value (row 07)

Comments and Feedbacks:

- it is required to explicitly introduce a goal to link an function (fresh) to the Actor performing it;
- it is not possible to describe activities using mental objects (e.g. Beliefs) as tools.

Therefore, if we wish to represent step 7 as an activity we must explicitly assert it as such.

The goal of the next exercise was to express row 09) as a set of OWL 2 Class Expressions. The answer was approached by trying to represent row 09) as a Communication Activity. The resulting OWL Class Expressions is presented in Figure 22.

```
Class: Send_Na.Actor_to-B
EquivalentTo:
(hasGoal some Goal_of_Actor_Alice)
 and (performs some
    (math:Function
     and (math:hasCodomain some Behaviour)
     and (math:hasDomain some Request_set_Na.Actor)))
 and (useTools some Channel_A-to-B)

Class: Goal_of_Actor_Alice
EquivalentTo:
mental-states:aimed_by some
    (agent
     and (roles:count-as some Actor_Alice_Role))
Class: Request_set_Na.Actor
EquivalentTo:
communication:request some apply_set_Na.Actor

Class: apply_set_Na.Actor
EquivalentTo:
```

```
(communication:hasSubject some set_Na.Actor)
 and (statement:hasObject some Encrypted_Na.Actor)

Class: set_Na.Actor
SubClassOf: wsdl:InterfaceOperation

Class: Encrypted_Na.Actor
EquivalentTo:
(agents:allows some
    (math:Function
     and (math:hasCodomain some Na.Actor)
     and (math:hasDomain some PrivateKeyOfB)))
 and (mereology:hasComponent some PublicKeyOfB)

Class: PublicKeyOfB
SubClassOf: communication:Medium
DisjointWith: PrivateKeyOfB

Class: PrivateKeyOfB
SubClassOf: communication:Medium
DisjointWith: PublicKeyOfB
```

Figure 22 OWL Class Expressions representing transmission of a value (row 09)

Comments and Feedbacks:

- The ontology does not provide a class to represent encrypted data (i.e. `{Na.Actor}_pk(B)`). Therefore it would be beneficial to introduce concepts dealing with encryption and its related concepts (e.g. borrowing concepts from XML-Encryption and/or XML-Signature);
- The ontology does not provide a class representing one-way communication channels: it is required to define it in the example;
- The ontology does not provide means to describe the correspondence between a public key and its private key therefore it was required to introduce a specific object property to complete the exercise. Recommendation: it would be beneficial to introduce concepts to describe key pairs used in public key cryptographic schemes.

ASLan++ Standard Prelude

This section presents how the standard definitions contained in the ASLan++ Standard Prelude [36] can be formalized using the ASSERT4SOA Ontology.
The ASLan++ standard prelude contains standard definitions that are imported by all ASLan++ specification.
The goal of this exercise was to express the meaning of the terms defined in the ASLan++ Standard Prelude [36] as a set of OWL 2 Class Expressions.
Since the ASLan++ semantics is defined in terms of ASLan [36] the exercise focused on the ASLan transalation of the ASLan++ prelude. The following table shows the semantics of the ASLan++ Standards prelude in terms of ASLan that was considered to perform the exercise. For the sake of the explanation the prelude is presented divided into five sections.

#	ASLan++	ASLan
1	`types`	`section typeSymbols`

	fact; protocol_id; message; nat < message;	fact; protocol_id; message; nat < message;

2	symbols
	i: agent;
	dishonest(agent): fact;
	iknows(message): fact;
	true, false: fact;
	0,1,2,3, ...: nat;
	succ(nat) : nat;
	% non-associative concatenation
	"." (message, message): message;
		section types
		i: agent
		isAgent(i)
		true, false: fact
		0,1,2,3,…:nat
		section signature
		dishonest(agent): fact;
		iknows(message): fact;
		succ(nat) : nat;
		"." (message, message): message;

3	types agent < message; symmetric_key < message; public_key < message; private_key < message;	section typeSymbols agent < message; symmetric_key < message; public_key < message; private_key < message;

4	symbols
	noninvertible hash (message): message;
	
	noninvertible scrypt (symmetric_key, message): message;
	noninvertible crypt (public_key, message): message;
	
	noninvertible sign (private_key, message): message;
	
	private inv(public_key): private_key;
	
	noninvertible pk(agent): public_key;
	
	noninvertible defaultPseudonym(agent,nat): agent;
		section signature
		hash(message):message;
		M:message;
		hc hash_public(M):
		iknows(hash(M)):-iknows(M);
		
		scrypt(symmetric_key, message): message;
		crypt(public_key, message):message;
		
		sign(private_key, message):message;
		
		inv(public_key): private_key;
		PK:public_key;
		hc inv_invertible_1(PK):
		iknows(PK):-iknows(inv(PK))
		
		pk(agent): public_key;
		
		defaultPseudonym(agent,nat): agent;

5	clauses
	true_holds: true;
	dishonest_intruder: dishonest(i);
	analysis_crypt(K,M):
	iknows(M) :- iknows(crypt(K,M)) & iknows(inv(K));
	analysis_scrypt(K,M):
	iknows(M) :- iknows(scrypt(K,M)) & iknows(K);
	analysis_sign(K,M):
	iknows(M) :- iknows(sign(K,M));
		section hornClauses:
		
		hc dishonest(i);
		
		hc iknows(M) :- iknows(crypt(K,M)) & iknows(inv(K));
		
		hc iknows(M) :- iknows(scrypt(K,M)) & iknows(K);
		
		hc iknows(M) :- iknows(sign(K,M));

Table 2 Semantics of the ASLan++ Standards prelude

The objective was to find an OWL encoding for the ASL++ standard prelude allowing to use DL inference mechanisms to get derivations compatible with the transition system defining the semantics of ASLan.

The following paragraphs discuss the proposed solution section by section.

Section 1: The key choice made in the solution was to use OWL individuals to encode ASLan facts. Therefore in the solution no OWL expression was introduced to represent the `fact` type symbol.
For the `protocol_id` type it was introduced a (primitive) subclass of the `agents:Object_Identifier`. The `message` type was set equivalent to the `agents:Signal` class provided by the ontology. The `nat` type was recognised as equivalent to the `primitive-types:Natural` in the ontology. The resulting OWL Class expressions is presented in Figure 23:

```
Class: Protocol_ID
    SubClassOf: agents:Object_Identifier

Class: message
    SubClassOf: agents:Signal

Class: Nat
    EquivalentTo:
        primitive-types:Natural
    SubClassOf: message
```

Figure 23 Section 1 of ASLan++ statndard prelude as OWL Class Expressions

Section 2: The ASLan constant `i` was encoded by means of the `intruder` OWL individual. Similarly the `true` and `false` ASLan facts were encoded by means of two corresponding OWL individuals belonging to `primitive-types:True` and `primitive-types:False` respectively. The infinite number of constants of type `nat` was not translated. Two classes were introduced to map respectively the subset of dishonest agents (`Dishonest`) and the subset of messages known by an intruder (`IntruderKnowledge`). The `succ` function was encoded by means of an OWL Object property having `nat` both as domain and range. The Composite message class encodes the ASLan++ concatenation function ".".
The resulting OWL descriptions is presented in Figure 24:

```
Individual: intruder
    Types: Agent

ObjectProperty: is_Agent

Individual: intruder
    Types: Agent
    Facts:
     is_Agent intruder

Individual: true
    Types: primitive-types:True
    DifferentFrom: false

Individual: false
    Types: primitive-types:False
    DifferentFrom: true

Class: Dishonest
```

```
    SubClassOf: Agent

Class: IntruderKnowledge
    SubClassOf: message

ObjectProperty: succ
    Domain: Nat
    Range: Nat

Class: CompositeMessage
    EquivalentTo:
        (math:first only message)
         and (math:second only message)
    SubClassOf: message
```

Figure 24 Section 2 of ASLan++ statndard prelude as OWL Class Expressions

Section 3: The translation of this section was pretty straightforward. The only consideration was that, since ASLan states that agent is a subclass of message, the ASLan `agent` class was recognized as matching the `agents:Agent_Appelation` in the ontology rather than the `agents:Agent` class. The resulting OWL descriptions is presented in Figure 25.

```
Class: Agent
    EquivalentTo: agents:Agent_Appelation
    SubClassOf: message

Class: Symmetric_Key
    SubClassOf: message

Class: Private_Key
    SubClassOf: message

Class: Public_Key
    SubClassOf: message
```

Figure 25 Section 3 of ASLan++ statndard prelude as OWL Class Expressions

Section 4: The general approach was to map single argument ASLan++ functions (i.e. `hash`, `inv`, `pk`) as OWL object properties. The remaining (multiple arguments) functions (i.e. `script`, `crypt`, `sign`, `defaultPseudonym`) were encoded as OWL class expressions (Figure 26).

```
ObjectProperty: hash
    Domain: message
    Range: Digest

ObjectProperty: plaintext
    InverseOf: cipherText

ObjectProperty: cipherText
    InverseOf: plaintext

Class: Digest
    SubClassOf: message

Class: hash_public
```

```
   EquivalentTo:
       Digest
       and (plainText some IntruderKnowledge)
    SubClassOf: IntruderKnowledge

ObjectProperty: key

Class: SCryptedMessage
    EquivalentTo:
        (key some Symmetric_Key)
         and (plainText some message)
    SubClassOf: message

Class: scrypt_public
    EquivalentTo:
        SCryptedMessage
        and (key some (IntruderKnowledge and Symmetric_Key))
        and (plainText some (IntruderKnowledge and message))
    SubClassOf: IntruderKnowledge

Class: CryptedMessage
    EquivalentTo:
        (key some Public_Key)
         and (plainText some message)
    SubClassOf: message

Class: crypt_public
    EquivalentTo:
        CryptedMessage
        and (key some (IntruderKnowledge and Public_Key))
        and (plainText some (IntruderKnowledge and message))
    SubClassOf: IntruderKnowledge

Class: SignedMessage
    EquivalentTo:
        (key some Private_Key)
         and (plainText some message)
    SubClassOf: message

ObjectProperty: inv
    Domain: Public_Key
    Range: Private_Key

ObjectProperty: pk
    Domain: Agent
    Range: Public_Key
```

Figure 26 Section 4 of ASLan++ standard prelude as OWL Class Expressions

Section 5: The exercise was dealt by mapping the head (consequent) of an Horn clause as superclass of the classes mapping the body (antecedent) of the clause (Figure 27).

```
Individual: intruder
    Types: Dishonest, Agent
    Facts:
     is_Agent intruder

Class: PrivateKey_is_Known_by_Intruder
    EquivalentTo:
```

```
      message
      and cipherText some (CryptedMessage
                           and IntruderKnowledge
                           and (key some (Public_Key
                                          and inv some IntruderKnowledge)))
    SubClassOf: IntruderKnowledge

Class: SymmetricKey_is_Known_by_Intruder
    EquivalentTo:
      message
      and cipherText some (SCryptedMessage
                           and IntruderKnowledge
                           and (key some (Symmetric_Key
                                          and IntruderKnowledge)))
    SubClassOf: IntruderKnowledge

Class: SignedMessage_is_Known_by_Intruder
    EquivalentTo:
      message
      and cipherText some SignedMessage
    SubClassOf: IntruderKnowledge
```

Figure 27 Section 5 of ASLan++ statndard prelude as OWL Class Expressions

ASLan++ Security Properties

This section presents how security properties of channels presented in [36] [37] can be formalized using the ASSERT4SOA Ontology.

ASLan++ considers the following channel security properties [36] [37]:

- *Authentic channels*: a receiver can rely that the sender has sent the message;
- *Confidential channels*: the sender can rely that only the intended receiver can receive a message;
- *Secure channels*: a channel both authentic and confidential;
- *Resilient channels*: a message inserted into a resilient channel will eventually be delivered;
- *Fresh channels*: each sent message can be can be received only once.

ASLan++ allows to formalize the semantics of the above security properties by means of three alternative channel models [37]:

- *Cryptographic Channel Model (CCM)*: focus on individual message transfer and are characterised by certain ways of encrypting/signing messages and transmitting them (e.g. Confidential to mean that a sender can be sure that only the intended receiver can read the content of a message sent over the channel);
- *Ideal Channel Model (ICM)*: focus on individual message transfer and are characterised by transition rules that model the intruder's limited ability to send and receive on those channels;
- *Abstract Channel Model (ACM)*: focus "session" between two parties modelled using explicit send and receive events that are constrained by an LTL formula over the traces.

The goal of this exercise was to express the Abstract Channel Model (ACM) [37] semantics of security properties as a set of OWL 2 Class Expressions. This

channel model was chosen because it was the basis for the discovery of an attack to the SAML-based Single Sign-On for Google Applications.
The ACM describes security properties as Linear Temporal Logic (LTL) formulas constraining the exchange of messages on communication channels.
The general approach used in the solution was to represent ACM facts as OWL Individuals. With this choice the state of the labeled transition system representing the system is naturally mapped by the the ABox component [38] of the knowledge base of an OWL Reasoner.

Confidential Channel A channel *ch* is said confidential if its output is exclusively accessible to a given receiver *p*. This condition is formalized by the following LTL formula [37]:

$$confidential(ch, p) := \boldsymbol{G}\forall(\text{rcvd}(a, b, m, ch) \Longrightarrow a = p)$$

The resulting OWL Class Expressions is presented in Figure 28.

```
Class: ConfidentialChannelOf_p
   SubClassOf:
     Channel,
     mereology:isMemberOf only
        (math:Set and (math:isDomainOf only TransformationBy_p))

Class: Channel
EquivalentTo:
   communications:Medium
   and (mereology:isMemberOf some Set)
   and (communications:carries only Message)

Class: Message
EquivalentTo:
 communication:Qualificatory_Expression
 and hasSender some agents:Agent_Appellation
 and hasReceiver some agents:Agent_Appellation
 and hasPayload some agents:Signal

Class: TransformationBy_p
EquivalentTo:
   math:Function
   and (activity:performedIn only ActivityBy_p)

Class: ActivityBy_p
EquivalentTo:
 activity:Activity
 and (activity:hasSubject value p)

Individual: p
 Types: agents:Agent
```

Figure 28 Confidential Channel as OWL Class Expressions

Authentic Channel A channel *ch* is said authentic if its input is exclusively accessible to a given sender *p*. This condition is formalized by the following LTL formula [37]:

$$authentic(ch, p) := \mathbf{G}\forall\big(\text{sent}(rs, a, b, m, ch) \Longrightarrow (a = p \wedge rs = p)\big)$$

The resulting OWL Class Expressions is presented in Figure 29.

```
Class: AuthenticChannelOf_p
 SubClassOf:
    Channel,
    Communication:carries only ExpressedAs_p,
    mereology:isMemberOf only
       (math:Set and (math:isRangeOf only TransformationBy_p))

Class: ExpressesAs_p
 EquivalentTo:
    Message and hasSender value p
```

Figure 29 Authentic Channel as OWL Class Expressions

Weak properties of channels

The exercise did not produced any OWL Class Expressions. This was not surprisingly given the limitation of Description Logics to describe so called "role-value maps" [39] resulting in the fact that with OWL DL it is not possible to declare an object property (or an object property chain) to be a sub-property of an object property chain. Indeed role-value maps are implied (not shown in this document) by the definition of weak properties of channels like *weakly confidential channel* defined as:

$$\mathbf{G}\forall\big((\mathrm{rcvd}(a,b,m,ch) \wedge \mathbf{F}\ \mathrm{rcvd}(a',b',m',ch)) \Rightarrow a = b'\big)$$

3.6 Experiment E6 – ASSERT4SOA Framework

This experiment has performed an application-based evaluation, i.e. it evaluated the ontology use within the ASSERT4SOA Framework in order to assess the impact of the ontology in the framework itself.

3.6.1 Description

The role of the ontology within the ASSERT4SOA Framework is:

1. to support the description of the different types of ASSERT4SOA certificates (i.e. ASSERT-E, ASSERT-M and ASSERT-O) and the security properties fulfilled by the software services;
2. to support the interoperability and comparison of the these certificates.

Therefore, the ontology provides an unambiguous catalogue of concepts related to security requirements and verification techniques to support the communication between Service Consumers, Service Providers, Certification Authorities and Evaluation Bodies.

The Ontology Manager is the component of the ASSERT4SOA Framework responsible for interacting with the ontology. In particular, the Ontology Manager exposes a set of functionalities for browsing and reasoning with it. The components that need to interact with the ontology can define the OWL Class expressions for their queries and give them in input to the appropriate Ontology Manager API or they can use the APIs with pre-defined queries. A

more detailed description of the functionalities provided by the Ontology Manager is reported in the deliverables D6.1[28] and D6.2[29].
The components of the ASSERT4SOA Framework that are expected to use the ontology via the Ontology Manager are:

- The Consumer Agent: this is the client-side component through which a client can formulate a consumer query, expressing the functional and security requirements;
- The Discovery Engine: the Matchmaker subsystem of the Discovery Engine will use the ontology in order to filter and rank the list of candidate services, retrieved during the discovery process according to their degree of fit with the security requirement requested by the consumer agent;
- The ASSERT-Verifier: this component will use the ontology in order to check the validity of the ASSERTs (using also verifiers specific to the kind of ASSERT to check).

The first version of the ASSERT4SOA Framework has been released at the end of September 2012 (deliverable D6.2).
The results of this experiment provide indications on the completeness and computational efficiency of the ontology within the framework.

3.6.2Results

This section presents the results of experiment E6 by describing the use of the ontology by the components of the framework interacting with it, as reported in the previous section. Performance aspects are also investigated in a dedicated subsection.

Use of the ontology by the Consumer Agent

In this exercise it has been assumed that the Consumer Agent needs to interact with the ontology in order to retrieve the categories of security properties defined on it. The OWL Class Expression query defined to achieve this objective was:

```
Query_1 = SUBCLASSES(
              assert-e: AbstractSecurityProperty
          )
```

The resulting OWL Classes were:

```
assert-e: Authenticity
assert-e:ConcreteSecurityProperty
assert-e: Confidentiality
assert-e: Integrity
assert-e: NonRepudiation
assert-e: Robustness
```

It has been noticed that the concept of “AbstractSecurityProperty” has been defined within the ASSERT-E module of the ontology, but the equivalent concept “SecurityPropertyCategory” is missing and should be added in the ontology. Furthermore some security property categories are missing (e.g. secrecy, accountability, availability etc...) and should be defined.

The list of security property categories also contains the OWL class `assert-e:ConcreteSecurityProperty`. This OWL Class should not appear in the list since it is a concept strictly related to the ASSERT-E certificate..

Use of the ontology by the Matchmaker Subsystem

The goal of this exercise was to evaluate if the ontology supports the specific matchmakers to rank the services according to the requested security requirements. For simplicity, the ASSERT-E Matchmaker was used.

ASSERT-E Matchmaker

In a first execution it has been assumed that the ASSERT-E Matchmaker needs to interact with the ontology in order to retrieve the ASSERT-E security properties stronger then the authenticity "password-based" security property. To this scope it was required to use concepts defined both in the ASSERT-E module and in the `assert-e_intentional` module. The latter was an OWL file added to the ontology in order to define examples of concepts strictly related to the E-based certificate (e.g. some kind of assert-e concrete security property, some kind of assert-e attribute etc.). The OWL Class Expression defined to query the ontology was:

```
Query_2 = DESCENDENTCLASSES(
              assert-e:ConcreteSecurityProperty
              and assert-e:propertyStrongerThan some
                     assert-e_intentional:Authenticity_PWD-based
          )
```

The resulting OWL Class was:

```
assert-e_intentional:Authenticity_Token+PWD
```

In a second execution it has been assumed that the ASSERT-E Matchmaker needs to interact with the ontology in order to retrieve the ASSERT-E security properties with algorithm RSA and key 1024.
The OWL Class Expression defined to query the ontology was:

```
Query_3 = DESCENDENTCLASSES(
              assert-e:ConcreteSecurityProperty
              and assert-e: hasClassAttribute some
                     assert-e_intentional: seq_RSA_1024
          )
```

The resulting OWL Class was:

```
assert-e_intentional: Integrity_RSA_2048_SHA-256
```

In a third execution it has been assumed that the ASSERT-E Matchmaker needs to interact with the ontology in order to retrieve the set of defined Test Types.

In this case it was not possible to define the appropriate OWL Class Expression for the query since the Test Type concept was not defined in the assessed version of the ontology: it must be added in the next releases.

Use of the ontology by the ASSERT Verifier

The goal of this exercise was to evaluate if the ontology supports the verification of the ASSERTs. In particular, in this experiment it has been considered the ASSERT-O Verifier component, invoked at run-time by the ASSERT Verifier, in order to verify the ASSERT-O.

To this end, in the ontology it has been added an OWL file providing an example of ASSERT-O. In this exercise it has been assumed that the ASSERT-O Verifier is invoked to check the ASSERT-O characterised by the following data:

- ASSERT-O IRI: http://www.ocsi.isticom.it/ID_82528990-c288-4a25-b1dc-f9b8f9787602;
- Security Property Reference: AuthenticatingSystem;
- Security Property Solution: secure-storage-authenticating-solution.owl.

Using the "`isCertificatePresent`" API, provided by the Ontology Manager, the ASSERT-O Verifier is invoked the ontology to verify if the ASSERT-O, with the IRI specified above, is present on it.

Then, it has been assumed that the ASSERT-Verifier interacts with the ontology in order to verify the consistency of the ASSERT-O. In particular, has been assumed that it invokes the ontology to check if the service model is a subclass of the ontology-based security property expressed in the certificate. The OWL Class Expression defined to query the ontology was:

```
Query_4 = DESCENDENTCLASSES(
              authentication-enforcer:AuthenticatingSystem
         )
```

The resulting OWL Class was:

```
secure-storage: SecureStorage
```

Performance of the ontology within the ASSERT4SOA Framework

The performance of the ASSERT4SOA Ontology can impact on the ASSERT4SOA Framework when one of the following activities is performed:

- The ontology is classified;
- The ontology is queried by some ASSERT4SOA component.

The ontology classification provides the computation of the subsumption hierarchies for classes, object properties and data properties. The newly computed subsumption relationships, inferred by the reasoner, will be visible on the inferred hierarchies.

Within the ASSERT4SOA Framework has been chosen to classify the ontology at start-up of the framework, so that the time spent for the classification activity will impact on it only once. In this way, the subsequent queries submitted to the ontology by the ASSERT4SOA components will impact on the framework only for the time necessary to execute the query. The classification

will be repeated only if some change will be applied on the ASSERT4SOA Ontology (e.g. a new OWL class will be added).
Table 3 reports both the classification time of the ontology and the execution time of the queries described in the previous exercises.

Task	Execution time (ms)
Ontology Classification	~ 20000
Query_1 execution	~ 10
Query_2 execution	~ 241
Query_3 execution	~ 80
Query_4 execution	0

Table 3 Performance of the ASSERT4SOA Ontology

3.6.3Comments and feedback

When used in the ASSERT4SOA Framework, the ontology revealed to be rather complete, with the exception of some security property categories (e.g. secrecy, accountability, availability etc...) that are missing and should be defined. It is recommended to move all the concepts related to security aspects (e.g. abstract security property, security property category, security requirement, etc...) in a dedicated module of the ontology.
Some security property categories are missing.
With regard to performance aspects, it has been concluded that the current version of the ontology doesn't present a particular impact on the overall performance of the ASSERT4SOA Framework.
.

Chapter 4

Conclusions

As described in section 2.4, the experiments executed during the evaluation activity allow to:

- validate the ontology, i.e. check whether it implemented correctly the requirements reported in deliverable D3.1;
- evaluate the impact of the ontology when used in the ASSERT4SOA Framework;
- improve the quality of the ontology with regard to the quality criteria described in section 1.1.2;
- assess whether the ontology can be used to model concepts pertaining to particular domains/documents, such as Common Criteria, ASSERT4SOA Data Model (the so called A4S Model), A-SerDiQueL and ASLan++.

The results of the experiments permit to draw several conclusions on the clarity, expressivity ad domain completeness of the ASSERT Ontology. These aspects are extensively analyzed in the following sections.

4.1 Ontology clarity

During the execution of the experiments, the main concepts of the ontology resulted clear to identify and use. In some occasions there has been the need for some support by the authors of the ontology to help identify synonyms for some concepts (e.g. Context or Business Objectives).
In general, it was noted that the ontology modularity helps the identification of concepts. It was recommended to document in a more extensive way some of the concepts, to avoid misleading interpretations on their purpose.

4.2 Ontology expressivity

During this experiment it was realized that it was not possible to express specific queries (e.g. “What is the service model used in a proof?”). Indeed, there

are some object properties that link concepts, but their level of abstraction is too high, resulting in somehow generic connections (*hasDomain* object property is the way to retrieve the *SecuritySolutionReference* considered in a proof, but how this entity is related to the service model and to the security controls is not defined yet).
While the definition of the *Sequence* class is very flexible and allows to express all regular expressions, modelling some XML constructs (see experiment E4), has been somehow cumbersome, especially with optional XML elements.

4.3 Domain completeness

The maturity of concepts from the Top Ontology is more than acceptable, as well as domain specific concepts from the three types of ASSERTs. In the execution of several experiments though, it was highlighted ad difficulty in specifying concepts generic for ASSERTs and not specific to one type in particular (e.g. ASSERT-M). Concepts such as security property categories, validity period and issuer of an ASSERT, are already present in the assert-specific modules of the ontology, but need to be identified and moved to the upper level of the ontology, so that they can be referred to by all the specific modules. The same applies for the certification domain, concepts are present in the specific ASSERT (E-M-O) modules, but it is recommended to add a module defining concepts from the certification domain such as *Certification Authority*, *Evaluation Body* and *Security Requirement*.

4.4 Computational efficiency

The computational efficiency, or the performance of the ontology, is related to how easily and successfully can reasoners process the ontology and how fast can reasoning services are.
This aspect has been assesses in experiment E6 where it was noted that current version of the ontology doesn't present a particular impact on the overall performance of the ASSERT4SOA Framework. After the start-up phase, when the classification of the whole ontology takes place, the response time for the queries to the ontology is within acceptable limits.

Bibliography

[1] Rudi Studer, Richard V. Benjamins, and Dieter Fensel, "Knowledge engineering: Principles and methods," *Data & Knowledge Engineering*, vol. 25, no. 1-2, pp. 161-197, 1998.

[2] Asunción Gómez-Pérez, "Ontology Evaluation," in *Handbook on Ontologies in Information Systems, First Edition*, Steffen Staab and Rudi Studer, Eds. Berlin: Springer, 2004, ch. 13, pp. 251-274.

[3] Denny Vrandečić, "Ontology Evaluation," in *Handbook on Ontologies, Second Edition*, Steffen Staab and Rudi Studer, Eds.: Springer Berlin Heidelberg, 2009, pp. 21-43.

[4] Leo Obrst, Werner Ceusters, Inderjeet Mani, Steve Ray, and Barry Smith, "The evaluation of ontologies," in *Revolutionizing Knowledge Discovery in the Life Sciences*, Christopher J.O. Baker and Kei-Hoi Cheung, Eds. Berlin: Springer, 2007, ch. 7, pp. 139-158.

[5] Janez Brank, Marko Grobelnik, and Dunjia Mladenić, "A survey of ontology evaluation techniques," in *Proceedings of the Conference on Data Mining and Data Warehouses (SiKDD 2005)*, 2005, pp. 166-170.

[6] Janez Brank, Marko Grobelnik, and Dunjia Mladenić, "Ontology Evaluation," SEKT Deliverable D1.6.1 2005.

[7] Jens Hartmann et al., "Methods for ontology evaluation," KnowledgeWeb Deliverable D1.2.3 2005.

[8] Marta Sabou, Vanessa Lopez, Enrico Motta, and Victoria Uren, "Ontology Selection: Ontology Evaluation on the Real Semantic Web," in *Proceedings of The 4th International EON Workshop, Evaluation of Ontologies for the Web, colocated with WWW2006*, 2006.

[9] Alexander Maedche and Steffen Staab, "Measuring Similarity between Ontologies," in *Proceedings of the European Conference on Knowledge Acquisition and Management(EKAW '02)*, 2002, pp. 251-263.

[10] Christopher Brewster, Harith Alani, Srinandan Dasmahapatra, and Yorick Wilcs, "Data Driven Ontology Evaluation," in *International Conference on Language Resources and Evaluation, Lisbon, Portugal, 24 - 30 May 2004*, p. 2004.

[11] Robert Porzel and Rainer Malaka, "A task-based approach for ontology evaluation," in *Proceedings of ECAI 2004 Workshop on Ontology Learning and Population*, 2004.

[12] Adolfo Lozano-Tello and Asunción Gómez-Pérez, "ONTOMETRIC: A Method to Choose the Appropriate Ontology," *Journal of Database Management*, vol. 15, no. 2, 2004.

[13] Nicola Guarino and Chris Welty, "An Overview of OntoClean," in *Handbook on Ontologies, First Edition*, Steffen Staab and Rudi Studer,

Eds.: Springer, 2004, pp. 151-159.

[14] Peter Spyns, "EvaLexon: Assessing triples mined from texts," STAR Lab, Brussels, Belgium, Technical Report 09 2005.

[15] Thomas R. Gruber, "Towards principles for the design of ontologies used for knowledge sharing," *International Journal of Human-Computer Studies*, vol. 43, no. 5/6, pp. 907-928, 1995.

[16] Aldo Gangemi, Carola Catenacci, Massimiliano Ciaramita, and Jens Lehmann, "Ontology evaluation and validation. An integrated formal model for the quality disgnostic task," Laboratory for Applied Ontologies - CNR, Technical Report 2005.

[17] Michael Grüninger and Mark S. Fox, "Methodology for the design and evaluation of ontologies," in *IJCAI95 Workshop on Basic Ontological Issues in Knowledge Sharing*, Montreal, 1995.

[18] Nicola Guarino, "Towards a Formal Evaluation of Ontology Quality," *IEEE Intelligent Systems*, pp. 1541-1672, 2004.

[19] Harith Alani and Christopher Brewster, "Metrics for Ranking Ontologies," in *4th Int. EON Workshop, 15th Int. World Wide Web Conf.*, Edimburgh, 2006.

[20] Claudia Pandolfo et al., "Requirements for an ontology supporting certificates interoperability," Deliverable D3.1 2011.

[21] Mari Carmen Suárez-Figueroa, Asunción Gómez-Pérez, and Boris Villazón-Terrazas, "How to Write and Use the Ontology Requirements Specification Document," in *OTM '09 Proceedings of the Confederated International Conferences, CoopIS, DOA, IS, and ODBASE 2009 on On the Move to Meaningful Internet Systems: Part II*, 2009.

[22] Valentina Di Giacomo, Stefania D'Agostini, Claudia Pandolfo, and Domenico Presenza, "First version of the ASSERT Ontology," Deliverable D3.2 2012.

[23] Valentina Di Giacomo et al., "ASSERT Software Infrastructure for SOA v1.0," Deliverable D6.2 2012.

[24] Marco Anisetti, Claudio Agostino Ardagna, and Ernesto Damiani, "Design and description of evidence-based certificates artifacts for services," Deliverable D4.1 2011.

[25] Andreas Fuchs and Sigi Guergens, "Formal models and model composition," Deliverable D5.1 2011.

[26] Valentina Di Giacomo, Claudia Pandolfo, Antonino Sabetta, Rajesh Harjani, Hristo Koshutanski Stefania D'Agostini, "Assert Software Infrastructure for SOA v1.0," Deliverable D6.2.

[27] E. Damiani, C. A. Ardagna, and N. El Ioini, *Open Source Systems Security Certification*.: Springer, 2008.

[28] Antonino Sabetta et al., "Architecture and High-level Design," Deliverable D6.1 2011.

[29] Valentina Di Giacomo, Claudia Pandolfo, Antonino Sabetta, Rajesh

Harjani, Hristo Koshutanski, Stefania D'Agostini, "Assert Software Infrastructure for SOA v1.0," Deliverable D6.2.

[30] Saml - oasis security assertion markup language. [Online]. https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security

[31] DOM - Document Object Model. [Online]. http://www.w3.org/DOM/

[32] JAXB - Java Architecture for XML Binding. [Online]. http://jaxb.java.net/

[33] ASSERT4SOA Consortium, Assert Language v.2, 2012, Project deliverable.

[34] George Spanoudakis et al., "ASSERTS Aware Service Query Language and Discovery Engine," Deliverable D2.1 2011.

[35] AVANTSAAR FP7-ICT-2007-1, Project No. 216471. (2011, March) www.avantsaar.eu. [Online]. www.avantsaar.eu/pdf/deliverables/avanstaar-d2.3_update.pdf

[36] AVANSTAAR-D2.3, "ASLan final version with dynamic service and policy composition," AVANTSAAR FP7-ICT-2007-1 Project , Deliverable D2.3, 2010.

[37] AVANTSSAR Project, "Attacker Model," Deliverable D3.3, 2008.

[38] W.Nutt F. Baader, "Basic Description Logics," in *The Description Logic Handbook*, D.Calvanese, D.L.McGuinness, D.Nardi, P.F. Patel-Schneider F.Baader, Ed.: Cambridge University Press, 2007, ch. 2, pp. 47-104.

[39] G. De Giacomo D. Calvanese,.: Cambride University Press, 2007, ch. 5, pp. 193-236.

[40] Stefania D'Agostini, Francesco Di Cerbo, and Antonino Sabetta, "ASSERT software infrastructure for SOA - Mockup," Working Document WD6.1 2012.

[41] Boris Motik, Peter F. Patel-Schneider, and Bernardo Cuenca Grau, "OWL 2 Web Ontology Language Direct Semantics," W3C Recommendation 27 October 2009 2009.

[42] Andreas Eckelhart, Stefan Fenz, Gernot Goluch, and Edgar R. Weippl, "Ontological mapping of common criteria's security assurance requirements," in *SEC, volume 232 of IFIP*, Hein S. Venter et al., Eds.: Springer, 2007, pp. 85-95.

[43] ASSERT4SOA Consortium, "Requirements for an ontology supporting certificates interoperability," 2011.