Requirements for an ontology supporting certificates interoperability

Deliverable D3.1
Rev. 1.0

Legal Notice
All information included in this document is subject to change without notice. The Members of the Assert4Soa Consortium make no warranty of any kind with regard to this document, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Members of the Assert4Soa Consortium shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.
<table>
<thead>
<tr>
<th><strong>Project Title</strong></th>
<th>ASSERT4SoA - Advanced Security Service Certificate for SOA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Grant Number</strong></td>
<td>257351</td>
</tr>
<tr>
<td><strong>Project Type</strong></td>
<td>CP - STREP</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Title of Deliverable</strong></th>
<th>Requirements for an ontology supporting certificates interoperability</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Subtitle of Deliverable</strong></td>
<td>D3.1</td>
</tr>
<tr>
<td><strong>Deliverable No.</strong></td>
<td>D3.1</td>
</tr>
<tr>
<td><strong>Dissemination Level</strong></td>
<td>Public</td>
</tr>
<tr>
<td><strong>Internal Rev. No.</strong></td>
<td>1.0</td>
</tr>
<tr>
<td><strong>Contractual Delivery Date</strong></td>
<td>30 September 2011</td>
</tr>
<tr>
<td><strong>Actual Delivery Date</strong></td>
<td>30 September 2011</td>
</tr>
<tr>
<td><strong>Contributing WPs</strong></td>
<td>WP3</td>
</tr>
<tr>
<td><strong>Editor(s)</strong></td>
<td>Claudia Pandolfo (ENG), Domenico Presenza (ENG)</td>
</tr>
<tr>
<td><strong>Author(s)</strong></td>
<td>Marco Anisetti (UNIMI), Claudio A. Ardagna (UNIMI), Stefania D’Agostini (ENG), Valentina Di Giacomo (ENG), Claudia Pandolfo (ENG), Domenico Presenza (ENG), Antonino Sabetta (SAP), Gimena Pujol (UMA)</td>
</tr>
<tr>
<td><strong>Reviewer(s)</strong></td>
<td>Renato Menicocci (FUB), Zaharina Stoynova (SIT)</td>
</tr>
</tbody>
</table>
The objective of this document is to present the requirements for an ontology suitable to support some of the functionalities envisaged for the ASSERT4SOA software infrastructure.

Broadly speaking an ontology can be understood as a description, given in a formal language, of part of the world (a domain) with the purpose to give a firm basis to the language that a community of agents (either human or computational) shares to predicate about that part of the world.

The ASSERT4SOA ontology is intended to support the unambiguous communication between Service Consumers, Service Providers, Certification Authorities and Evaluation Bodies.

Security requirements and Software services represent the core of the domain of the discourse described by the ASSERT4SOA ontology. In fact, from an abstract point of view, an ASSERT4SOA security certificate (i.e. an ASSERT in the project’s terminology) is a machine-readable document stating that a security requirement (e.g. integrity of exchanged messages) is addressed by a software service. Moreover, all the three types of ASSERT (ASSERT-E, ASSERT-O, ASSERT-M) entail some partial model of the Software service which is the subject of the certificate.

A challenge for the ASSERT4SOA ontology is to be enable the automatic answering to the decision problems entailed by the main use cases of the ASSERT4SOA infrastructure (e.g. mapping between the different types of ASSERT, discovery relation between the definition security properties, checking consistency between a model of a service and the definition of a security property). This deliverable present a set of possible formal languages that could fit the purpose.

This document is organised in two parts. The first part reports on the state of the art on enabling technologies (ontology-based reasoning, Semantic Web and Semantic Web services) and related work (Security ontologies). The second part presents the functional and non-functional requirements for the ASSERT4SOA ontology.
Contents

1 Introduction 5
  1.1 Goals and scope ................................................. 5
  1.2 Main use cases .................................................. 6
    1.2.1 ASSERT Certified Service Lifecycle ......................... 6
    1.2.2 Authoring of a Security Property ......................... 7
    1.2.3 Development of a software implementing internal security controls 8
    1.2.4 Check consistency between a security requirement and a service model ............................................. 9
    1.2.5 Issue of a Security Certificate ................................ 9
    1.2.6 Implementation of external controls ....................... 10
    1.2.7 Compare requested with provided ASSERT ................... 11

I State of the art 13

2 Ontology-based Reasoning 14
  2.1 Knowledge Representation and Reasoning ........................ 14
  2.2 Ontologies ..................................................... 15
  2.3 Formal languages for ontologies .................................. 16
    2.3.1 Frame Logic (F-logic) ..................................... 17
    2.3.2 CyCL ....................................................... 18
    2.3.3 Description Logics (DLs) ................................... 18
    2.3.4 RDF and RDF Schema ...................................... 19
    2.3.5 Web Ontology Language (OWL) .............................. 20
    2.3.6 Web Service Modeling Language (WSML) .................... 22

3 Semantic Web and Semantic Web services technologies 24
  3.1 The Semantic Web ............................................... 24
  3.2 Semantic Web Services ......................................... 24
    3.2.1 OWL-S ................................................... 25
    3.2.2 Web service Modeling Ontology (WSMO) .................... 26
    3.2.3 Unified Service Description Language (USDL) ............... 27
    3.2.4 Linked Open Services (LOS) ................................ 28
    3.2.5 Semantic Annotations for WSDL and XML Schema (SAWSDL) 28
    3.2.6 DIANE .................................................. 29
    3.2.7 SOAF .................................................. 30
4 Security ontologies

4.1 Existing ontologies

4.1.1 OWL-S Security Ontologies
4.1.2 NRL (Naval Research Laboratory) Security Ontology
4.1.3 Common Criteria ontology
4.1.4 Problem Domain Ontology (PDO)
4.1.5 The Security Ontology
4.1.6 Linkpings Universitet Security Ontology (SecOnt)
4.1.7 SEKT Ontology
4.1.8 security Characterization Language (SCL) ontology

4.2 Comparison of security ontologies

II Requirements

5 Functional requirements

5.1 Overview
5.2 Purpose
5.3 Scope
5.4 Level of Formality
5.5 Intended Users
5.6 Intended Uses
5.7 Representational Requirements (Groups of Competency Questions)

5.7.1 Requirements on security properties
5.7.2 Requirements on certificates
5.7.3 Requirements on service model ontology
5.7.4 Requirements on Upper Ontology

5.8 Pre-Glossary of Terms

5.8.1 Terms

6 Non functional requirements

6.1 Introduction
6.2 Requirements on ontology protection

6.2.1 Requirements for the Ontology Manager

Bibliography
Chapter 1

Introduction

1.1 Goals and scope

The objective of this document is to present the requirements for an ontology suitable to support some of the functionalities envisaged for the ASSERT4SOA software infrastructure [4]. More specifically the expected role of an ontology within the ASSERT4SOA infrastructure is twofold:

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

Broadly speaking an ontology can be understood as a description, given in a formal language, of part of the world (a domain) with the purpose to give a firm basis to the language that a community of agents (either human or computational) shares to predicate about that part of the world.

The ASSERT4SOA ontology is intended to support the unambiguous communication between Service Consumers, Service Providers, Certification Authorities and Evaluation Bodies. Such an ontology is needed not least because there is the need:

• for Service Consumers and Service Providers to unambiguously communicate about the meaning of the terms (e.g. authenticity, confidentiality) denoting the security requirements fulfilled by a consumed/provided software service. This need is in common with Service Providers/Developers, Certification Authorities and Evaluation Bodies that have to unambiguously communicate about the security claim that is requested to be certified.

• for Service Consumers and Certification Authorities to unambiguously communicate about the assumptions and evidences (e.g. testing results, formal proofs) supporting the claim that a software service fulfils a security requirement. Also this need is shared by Service Providers, Certification Authorities and Evaluation bodies in the context of a certification process.

Security requirements and Software services represent the core of the domain of the discourse described by the ASSERT4SOA ontology. In fact, from an abstract point of view, an ASSERT4SOA security certificate (i.e. an ASSERT in the
project’s terminology) is a machine-readable document stating that a security requirement (e.g. integrity of exchanged messages) is addressed by a software service. An ASSERT may also carry the description of the assumptions and evidences (e.g. results of tests performed) supporting the statement. All the three types of ASSERT (ASSERT-E, ASSERT-O, ASSERT-M) entail, within the representation of the assumptions, some partial model of the Software service which is the subject of the certificate.

The primary aim in describing a domain by means of a formal language is to allow automatic machine processing. Literature in the field offers a vast variety of languages for this purpose. Amongst the others, in the last decade, Description Logics have gained interest thanks to the adoption of one them by the W3C in the context of its Semantic Web initiative [8]. The interesting feature of Description Logics, and hence of the W3C OWL 2 DL language, is that they are a decidable fragment of the First Order Logic (FOL): inference problems like satisfiability, subsumption, instance checking and boolean conjunctive query answering expressed in the Description Logic underlying OWL 2 DL can be answered in polynomial time. These inference problems appear promising to back on the decision problems entailed by the main use cases of the ASSERT4SOA infrastructure (e.g. mapping between the different types of ASSERT, discovery relation between the definition security properties, checking consistency between a model of a service and the definition of a security property).

This document is organised in two parts. The first part reports on the state of the art on knowledge representation based on ontology (Chapter 2), Semantic Web and Semantic Web services technologies (Chapter 3), and Security ontologies (Chapter 4). The second part presents the functional (Chapter 5) and non-functional (Chapter 6) requirements for the ASSERT4SOA ontology.

1.2 Main use cases

This section summarises the main use cases envisaged for the ASSERT4SOA Ontology. They represent the basis used for the elicitation of the functional requirements that are presented in the Chapter 5 of this document.

1.2.1 ASSERT Certified Service Lifecycle

The ASSERT4SOA Ontology aims to support the unambiguous communication between the various actors contributing to the lifecycle of an ASSERT Certified Service by offering them a shared and formalised vocabulary (figure 1.1):

1. Security Engineers defines general security requirements that should be fulfilled by an ASSERT Certified Service;

2. Software Engineers address some of the security requirements by adding specific controls (application level controls) within the software implementing an ASSERT Certified Service.

3. Evaluation Bodies carry out the security evaluations aimed to verify (using various possible approaches e.g. model-based, test-based) that the developed software fulfils its security requirements. A model of the implemented software..
Figure 1.1: Secure Service Lifecycle

is one of the inputs of this phase (e.g. for model-based verification or for model-based testing).

4. Certification Authorities create and issue ASSERTs for the software implementing a service. Certification Authorities uses the evaluation results obtained by an Evaluation Body as the basis for the issue of an ASSERT.

5. Service Providers address some of the security requirements by deploying specific controls (container level controls) within the infrastructure hosting an ASSERT Certified Service. Service Providers deploy and operate the ASSERT Certified Service along with the corresponding ASSERTs issued by the Certification Authorities.

1.2.2 Authoring of a Security Property

A Security Engineer wants to author, name and share with other actors the definition of a new security property (figure 1.2):
1. The Security Engineer searches a security property having the identifier she intend to use to name the new security property;

2. The Security Engineer authors the definition of the new security property and adds it to the ASSERT4SOA Ontology;

3. The Security Engineer uses the ASSERT4SOA Ontology to check whether the new property definition is coherent with the other definitions already in the ontology (checks the consistency of the ASSERT4SOA Ontology);

4. The Security Engineer browses the ASSERT4SOA Ontology to learn whether equivalent definitions exist;

5. The Security Engineer makes visible (publishes) the new definition to the other actors;

**Extensions:**
- a security property having the intended identifier already exists.
- the ASSERT4SOA Ontology becomes inconsistent.
- equivalent security property definitions exist.

### 1.2.3 Development of a software implementing internal security controls

A Software Engineer wants to learn the definition of security property referenced within by the requirements of the software she has to develop (figure 1.3):

1. The Software Engineer uses the ASSERT4SOA Ontology to search a security property having the name referenced by the requirements;
2. The Software Engineer uses the ASSERT4SOA Ontology to get the definition of the security property retrieved in the previous step;

3. The Software Engineer uses the ASSERT4SOA Ontology to learn whether exist any security patterns providing the requested security property;

1.2.4 Check consistency between a security requirement and a service model

As part of an evaluation activity, an Evaluation Body wants to check whether the model of a software artefact provided by a Software Engineer is consistent with a claimed security property (figure 1.4):

1. The Evaluation Body uses the ASSERT4SOA Ontology to search a security property having the name referenced by the requirements;

2. The Evaluation Body uses the ASSERT4SOA Ontology to retrieve the definition of the security property returned by the previous step;

3. The Evaluation Body uses the ASSERT4SOA Ontology to retrieve the model of the software to evaluate;

4. The Evaluation Body uses the ASSERT4SOA Ontology to check consistency between the service model and the definition of the claimed security property;

**Triggering event:** A Certification Authority submits a request to evaluate whether a software provides the security property claimed by its developers.

1.2.5 Issue of a Security Certificate

A Certification Body wants to publish an ASSERT for a software verified by an Evaluation Body (figure 1.5):
1. By means of the ASSRT4SOA Framework [5], the Certification Body adds to the ASSRT4SOA Ontology a new ASSERT including the service model provided by the Software Engineer and the Security Property verified by the Evaluation Body.

2. The Certification Body makes visible (publishes) the new ASSERT to the other actors.

**Triggering event:** An Evaluation Body makes available the proof that a software fulfils a security requirement.

### 1.2.6 Implementation of external controls

A Service Provider wants to deploy the external security controls (e.g. firewall rules) assumed by a software she wants to deploy and operate as a service (figure 1.6):

1. The Service Provider uses the ASSRT4SOA Ontology to search a security property having the name referenced by the requirements;

2. The Service Provider uses the ASSRT4SOA Ontology to retrieve the definition of the security property returned by the previous step;

3. The Service Provider uses the ASSRT4SOA Ontology to learn whether exists any security pattern providing the requested security property;

4. The Service Provider implements external controls (e.g. defines the firewall rules);

5. The Service Provider deploys the software and operates it as a service.
1.2.7 Compare requested with provided ASSERT

In the context of a service discovery [5], a Service Consumer wants to evaluate whether a service provides a security property according to a certification scheme (figure 1.7): A Service Consumer wants to evaluate whether a service has an ASSERT certifying requested security property (figure 1.7):

1. The Service Consumer uses the ASSERT4SOA Ontology to search the ASSERTs valid for a service;

2. The Service Provider uses the ASSERT4SOA Ontology to discover which of the ASSERTs certifies a property equivalent to the requested security property;

3. The Service Provider uses the ASSERT4SOA Ontology to establish whether an ASSERT retrieved in the previous step is equivalent to or subsumes the requested ASSERT type.
Figure 1.6: Implementation of external controls

Figure 1.7: Compare requested with provided ASSERT
Part I
State of the art
Chapter 2

Ontology-based Reasoning

2.1 Knowledge Representation and Reasoning

Artificial Intelligence (AI) is the study of intelligent behaviour achieved through computational means. Knowledge representation and reasoning is the area of AI concerned with how knowledge can be represented symbolically and manipulated in an automated way by reasoning programs. Intelligence, as exhibited by humans, is a complex and mysterious phenomena. Intelligent behaviour is conditioned by knowledge: for a wide range of activities we make decisions about what to do based on what we know (or believe) about the world, effortlessly and unconsciously. We say that someone has behaved unintelligently when he or she has failed to use what he or she did know. It is thinking that is supposed to bring what is relevant in what we know to bear on what we are trying to do. Knowledge representation and reasoning is the study of thinking as a computational problem, the part of AI concerned with how an agent uses what it knows in deciding what to do.

"What is knowledge?" is a question that has been discussed by philosophers since the ancient Greeks. Informally speaking, knowledge is a relation between a knower and a proposition, i.e. the idea expressed by a simple declarative sentence; for example,” Charles (knower) knows that Abraham Lincoln was assassinated (proposition)”. Propositions are abstract entities that can be true or false. To say that a knower knows something is to say that he or she has formed a judgement of some sort and realized that the world is one way and not another. Representation can be seen as a relationship between two domains, where the first (the representor) is meant to "stand for" or take the place of the second. The representor is more concrete, immediate or accessible than the second; for example, the drawing of a circle with a plus below stand for the more abstract concept of womanhood. The type of representor is usually a formal symbol, i.e. a character or group of characters taken from some predetermined alphabet; e.g. the digit "2" stands for the number 2, as does the group of letters "II" and the words "two" and "due". Knowledge representation is therefore the field of study related to using formal symbols to represent a collection of propositions believed by some agent. Reasoning is the formal manipulation of the symbols representing a collection of believed propositions to produce a representation of new ones. This the reason why

1Strictly speaking, the sentences expressing the propositions are true or false, the propositions themselves are either factual or nonfactual
symbols are more accessible that the propositions they represent: they should be concrete enough to be manipulated in such a way as to construct representations of new propositions.

The concept of logical entailment is central in reasoning. We say that the propositions represented by a set of sentences $S$ entail the proposition represented by a sentence $p$ when the truth of $p$ is implicit in the truth of sentences in $S$. According to this view the job of reasoning is to compute the entailments of a Knowledge Base.

## 2.2 Ontologies

The word "ontology" is used with different meanings in different communities. Ontology (uncountable noun) is the branch of philosophy dealing with the nature and structure of "reality". In Computer Science an ontology (countable noun) is a special kind of information object or computational artifact. Computational ontologies are a means to formally model the structure of a system, i.e. the relevant entities and relations that emerge from its observation and which are useful to our purpose. In 1993 Gruber [27] defined an ontology as an "explicit specification of a conceptualization". In 1997 Borst [11] defined an ontology as "formal specification of a shared conceptualization". This definition additionally required that the conceptualization should express a shared view between several parties. Such a conceptualization should also be expressed in a (formal) machine readable format. In 1998, Studer et al. [52] merged the above two definitions stating that:

An ontology is a formal, explicit specification of a shared conceptualization

A conceptualization is an abstract and simplified model of the world we wish to represent for some purpose. Such a model is obtained by identifying the entities (concepts) that are assumed to exist in the area of interest and the relationships that hold among them. The main objective of an ontology is the formal representation of the concepts imputable to the examined application domain (i.e. our portion of reality), according to our knowledge of the domain itself. In order to specify a conceptualization, we need a non ambiguous definition of the terms representing the knowledge of the domain. The word shared is used to emphasize that the ontology captures a consensual understanding, i.e. accepted by a group. Each ontology does not represent an objective knowledge, but a shared point of view.

In the literature, we generally find three common layers of knowledge, corresponding to three different types of ontologies:

- **Top ontologies**, which capture general, domain independent knowledge (e.g. space and time), for example CyC (see section 2.3.2). These ontologies are shared by large numbers of people across different domains.
- **Domain ontologies**, which capture the knowledge in a specific domain. Domain ontologies are shared by the stakeholder in a domain.
- **Application ontologies**, which capture the knowledge necessary for a specific application. Application ontologies are not really ontologies, because they are not really shared.

---

2A top ontology should be applicable to every application domain by introducing domain specific concepts.
The separation between these three levels of generality is not always strict. An ontology usually consists of:

- **Concepts:**
  - Generic concepts representing the main categories according to the world could be organized: these main categories constitute the top level of a potential ontology of the world (top ontology). Generic concepts should be domain independent;
  - Concepts describing a specific application domain (domain ontology).
- **Concepts definition** (in formal or natural language);
- **Relationships among concepts.** Possible relationships are:
  - Taxonomies (IS_A)
  - Meronymies (PART_OF)
- **Attributes:** aspects, properties, features, characteristics that individuals can have;
- **Assertions,** i.e. any statement that is true in the ontology.

An ontology always contains a vocabulary including the definition of the terms, i.e. a description of their meaning.

An ontology together with a set of individual instances of classes constitutes a knowledge base, even though there is a fine line where the ontology ends and the knowledge base begins.

Ontologies are explicitly specified in a formal language.

### 2.3 Formal languages for ontologies

Several ontology languages have been developed during the last years. These languages can be divided into the following categories based on their structure:

- **Frame-based languages:** F-Logic, KL-ONE, KM programming language
- **First order based languages:** Common Logic, KIF, CyCL
- **Description logic-based languages:** Description Logics, OWL

Alternative classifications are possible; for further details see Wikipedia\(^3\). In this document only the most popular languages will be considered.

According to [3] the main requirements of an ontology language are:

1. a well-defined syntax
2. a well-defined semantics
3. efficient reasoning support
4. sufficient expressive power

\(^3\)http://en.wikipedia.org/wiki/Ontology_language
5. convenience of expression.

A well-defined syntax is a necessary condition for machine-processing of information. Formal semantics describes precisely the meaning of knowledge; the semantics does not refer to subjective intuitions, nor is it open to different interpretations by different persons or machines. Formal semantics allow humans to reason about the knowledge, therefore semantics is a prerequisite for reasoning support. Reasoning support let to:

- check the consistency of the ontology and the knowledge;
- check for unintended relationships among classes;
- automatically classify instances of classes.

Automated reasoning support allows to check more cases than what manually possible. It is valuable when designing large ontologies and integrating and sharing ontologies from various sources. Formal semantics and reasoning support can be achieved by mapping an ontology language to a known logical formalism, and by using automated reasoners already existing for those formalisms.

When designing an ontology language the trade-off between expressive power and efficient reasoning support should be carefully considered: the richer the language is, the more inefficient the reasoning support becomes, often crossing the border of non-computability. We need a language that can be supported by reasonably efficient reasoners, while being sufficiently expressive to express large classes of knowledge.

### 2.3.1 Frame Logic (F-logic)

Frame Logic (abbreviated as F-logic)[36] is a knowledge representation and ontology language that combines the advantages of conceptual modeling coming from object-oriented, frame-based languages with the declarative, compact and simple syntax, and the well-defined semantics of a logic-based language. In contrast to description logic based ontology formalism the semantics of F-logic is that of a closed world assumption (as opposed to DL’s open world assumption); furthermore, F-logic is generally undecidable.

A simplified and extended version of the original syntax proposed in [36] was developed by the F-logic Forum group\(^4\) by incorporating experience gained using F-logic in real life applications.

F-logic is an objected-oriented language supporting typing, meta-reasoning, complex objects, methods, classes, inheritance, rules, queries, modularization and scoped inference. Ontologies are modelled in this language in object-oriented style: one starts with class hierarchies, proceeds with type specification, define the relationships among classes and objects using rules and finally populates the classes with concrete objects.

There are several implementations of F-logic, such as FLORID\(^5\), Ontobroker\(^6\) and FLORA-2\(^7\).

---

\(^4\)http://projects.semwebcentral.org/projects/forum/
\(^5\)http://dbis.informatik.uni-freiburg.de/index.php?project=Florid
\(^6\)http://www.ontoprise.de/en/products/ontobroker/
\(^7\)http://flora.sourceforge.net/
2.3.2 CyCL

CyCLTM is an ontology language used to represent the knowledge stored in the Cyc Knowledge Server.

Cyc is an artificial intelligence project that defines a comprehensive ontology and knowledge base of everyday common sense knowledge. The project was started in 1984 by Douglas Lenat at Microelectronics and Computer Technology Corporation (MCC) and is developed by company Cycorp. The Cyc system includes some distinct components: the Cyc Knowledge Base, the Cyc Inference Engine, the CyCL Representation Language, the Natural Language Processing Subsystem, the Cyc Semantic Integration Bus and the Cyc Developer Toolsets.

The Knowledge Base (KB) consists of terms and assertions relating them. These assertions include both simple ground assertions and rules. The Cyc KB is divided into thousands of ”microtheories”, each of which is a bundle of assertions sharing a common set of assumptions; some microtheories focus on a particular domain of knowledge, a particular level of detail, a particular interval in time, etc. At present, the Cyc KB contains nearly five hundred thousand terms, including about fifteen thousand types of relations, and about five million assertions. New assertions are continuously added to the KB.

CyCL, the CyC representation language, is a formal language whose syntax derives from first-order logic and from Lisp. The vocabulary of CyCL consists of terms. A term is anything that can be an argument to a predicate or function; terms can be divided into constants, non-atomic terms (NATs), variables, and a few other types of objects. Terms are combined into CyCL expressions, which are used to make assertions in the CYC KB.

CyCL includes a quoting mechanism that allows to differentiate between knowledge involving a concept and knowledge about the term that expresses that concept; in this way it is possible to represent facts about the term, such as the author and the creation time. CyCL also has a number of features of higher-order logics, such as quantification over predicates, functions and sentences, and there is the ability to take predicates as values, including themselves in some cases. For a detailed description of CyCL language features, see http://cyc.com/cycdoc/ref/cycl-syntax.html.

OpenCyc9 is a freely available section of some the general knowledge in the Cyc system, in particular the taxonomic and definitional knowledge. OpenCyc has been criticized for being restrictive, particularly with respect to instance-level knowledge, for lacking any reasoning capability, and being formally inconsistent. Concerns about inadequate coverage of some types of knowledge have been partially addressed by the release of ResearchCyc10, that includes also extensive instance-level population; ResearchCyc is available under a free research-purposes license.

2.3.3 Description Logics (DLs)

Description Logics (DLs), a subset of first-order logic, are a family of knowledge representation languages that can be used to represent the knowlege of an application domain in a structured and formally well-understood way. The important notions of the domain are described by concept descriptions, i.e. expressions built from atomic concepts (unary predicate) and atomic roles (binary predicates) using the concept

---

8http://cyc.com/cyc
9http://opencyc.org/
10http://research.cyc.com
and role constructors provided by the particular DL.

Description logic systems provide their users with various inference capabilities that deduce implicit knowledge from the explicitly represented knowledge. The subsumption algorithm determines subconcept-superconcept relationships: \( C \) is subsumed by \( D \) if all instances of \( C \) are necessarily instances of \( D \), i.e., the first description is always interpreted as a subset of the second description. The instance algorithm determines instance relationships: the individual \( i \) is an instance of the concept description \( C \) if \( i \) is always interpreted as an element of \( C \).

In order to ensure a reasonable and predictable behavior of a DL system, these inference problems should at least be decidable for the DL employed by the system, and preferably of low complexity. Consequently, the expressive power of the DL in question must be restricted in an appropriate way. If the imposed restrictions are too severe, however, then the important notions of the application domain can no longer be expressed. Investigating this trade-off between the expressivity of DLs and the complexity of their inference problems has been one of the most important issues in DL research.

Several web ontology languages are founded on DLs such as \( SHIQ \), \( SHIF \), \( SHOIN \) and \( SROIQ \). All of these logics are based on an extension of the DL \( ALC \) (Attributive Language with Complements) including transitive roles, called \( S \). This logic can be extended to include features such as role inclusion axioms \( (H) \), nominals \( (O) \), inverse roles \( (I) \), functional roles \( (F) \) and number restrictions \( (Q \text{ if qualified, } N \text{ otherwise}) \). \cite{30} describes an extension of \( SHOIN \) called \( SROIQ \). \( SROIQ \) takes into account complex role inclusion axioms to express propagation of one property along another one \( (R) \). Furthermore \( SHOIN \) has been extended with reflexive, symmetric, transitive, irreflexive and disjoint roles, a universal role, constructs \( \exists R.\text{Self} \), negated role assertions in Aboxes and qualified number restrictions.

If the logic is extended with datatypes, a \((D)\) is appended to the name of the logic, e.g. \( SHOIN(D) \).

### 2.3.4 RDF and RDF Schema

The Resource Description Framework (RDF) \cite{7}, developed by the W3C, is a XML-based language for representing information about resources in the World Wide Web. The RDF data model is based upon the idea of making statements about Web resources in the form of subject-predicate-object expressions, called triples in RDF terminology. The subject of an RDF statement is a resource, that is any entity that is uniquely identifiable by a Uniform Resource Identifier (URI). Some resources are unnamed and are called blank nodes (or bnodes) or anonymous resources; they are not directly identifiable. The predicate is a resource as well, representing a relationship. The object is a resource or a Unicode string literal. A collection of RDF statements intrinsically represents a labeled, directed multi-graph.

RDF provides a way to express simple statements about resources but RDF user communities also need the ability to define the vocabularies (terms) they intend to use in those statements, specifically, to indicate that they are describing specific kinds or classes of resources, and will use specific properties in describing those resources. RDF itself provides no means for defining such application-specific classes and properties. These are described as an RDF vocabulary, using extensions to RDF...
referred to as RDF Schema.

RDF Schema [5] (abbreviated as RDFS), is a semantic extension of RDF. It provides mechanisms for describing groups of related resources and the relationships between these resources. RDF Schema vocabulary descriptions are written in RDF.

The expressivity of RDF and RDF Schema is limited. They allow the representation of some ontological knowledge but a number of other features are missing, for example:

- **Local scope of properties**: `rdfs:range` define the range of a property but range restrictions applicable to some classes only cannot be defined;

- **Disjointness of classes**;

- **Boolean combination of classes**: RDFS does not allow the definition of new classes by combining other classes with union, intersection and complement;

- **Cardinality restrictions**;

- **Special characteristics of properties**: in RDF Schema it is impossible to say that a property is transitive, unique or inverse of another property.

### 2.3.5 Web Ontology Language (OWL)

A joint initiative by research groups in America and Europe defined a language richer than RDF and RDF Schema, called DAML+OIL [11]. DAML+OIL was taken as a starting point for the W3C Web Ontology (WebOnt) Working Group [12], now closed, that was responsible for defining an ontology language for the web. The WebOnt Working Group developed the Web ontology Language OWL, which became a W3C Recommendation on 10 February 2004 [45]. Although OWL has been initially developed as an ontology language for the Semantic Web, it has been widely adopted as a de-facto standard for ontology-based applications.

The requirements for the ontology language identified by the WebOnt Working Group prompted the WG to define OWL as three different sublanguages (also called species) with increasing expressivity:

- **OWL Full**: the entire language that uses all OWL languages primitives, allowing also to combine them in arbitrary ways with RDF and RDF Schema.

  The advantage of OWL Full is that it is fully upward compatible with RDF, both syntactically and semantically [13], the disadvantage is that it is so powerful as to be undecidable, therefore neither complete nor efficient reasoning support is available.

- **OWL DL** (short for Description Logic): a sublanguage of OWL Full that restricts the way in which the constructors from OWL and RDF can be used. Its advantage is that it allows efficient reasoning support but full compatibility with RDF is lost. It is expressively equivalent to \( \text{SHOIN}(D) \).

---


[13] It means that any legal RDF document is also a legal OWL Full document and any valid RDF/RDFS conclusion is also a valid OWL Full conclusion.
• **OWL Lite**: a further restriction of OWL DL to a subset of the language constructors. For example, OWL Lite prohibits enumerated classes and disjointness statements. OWL Lite is easier to understand and implement but its expressivity is limited. It is expressively equivalent to \( SHIF(D) \).

In October 2007, the W3C OWL working group\(^{14}\) was started to refine and extend OWL. This new version, called OWL 2, has been released as a W3C Recommendation on 27 October 2009 and is compatible with the OWL standard of 2004 (now referred to as OWL 1). An overview of the documents included in the OWL 2 specification can be found in [58].

OWL 2 inherits the language features, design decisions and use cases for OWL 1 and adds new functionality. Some of the new features are "syntactic sugar" (e.g., disjoint union of classes) to express some common statements in an easier way, while others offer new expressivity, including:

- keys;
- property chains;
- richer datatypes, data ranges;
- qualified cardinality restrictions;
- asymmetric, reflexive, and disjoint properties;
- enhanced annotation capabilities

Various syntaxes are available in OWL 2 to serialize and exchange ontologies\([58]\). OWL 2 is specified not only in terms of a grammar but also of structure: in addition to the Functional Syntax, OWL 2 has introduced the structural specification to specify the conceptual structure of OWL 2 ontologies, defined with UML. The structural specification provides a normative abstract model for all the OWL 2 syntaxes. OWL 2 DL and OWL 2 Full extend OWL 1 species, while OWL Lite is not regarded as important because, although it was defined as a fragment of OWL 1 DL with the aim of being simpler, indeed it is as almost as complex as OWL DL. In OWL 2 some of the restrictions previously applicable to OWL DL have been relaxed; as a consequence, the set of RDF graphs that can be handled by DL reasoners is larger. OWL 2 DL is expressively equivalent the DL \( SROIQ(D) \).

In addition to OWL 2 DL and OWL 2 Full, OWL 2 specifies three additional profiles \([46]\), designed to be approachable (i.e. easier to implement) subsets of OWL 2 having advantages in particular application scenarios:

- **OWL 2 EL** is suitable for applications needing large ontologies. Based on the Description Logic \( EL^++ \), it enables polynomial time algorithms for all the standard reasoning tasks.

- **OWL 2 QL** is suitable for applications using lightweight ontologies with a large number of individuals, where query answering is the most important reasoning task. OWL QL, whose acronym reflects the fact that query answering in this profile can implemented by rewriting queries into a standard relational Query Language, enables conjunctive queries to be answered in LogSpace (more precisely, \( AC^0 \)) using standard relational database technology.

\(^{14}\)http://www.w3.org/2007/OWL/wiki/OWL_Worlding_Group
• **OWL 2 RL** enables the implementation of polynomial time reasoning algorithms using rule-extended database technologies operating directly on RDF triples. It is aimed at applications that require scalable reasoning without sacrificing too much expressive power.

Each profile is defined as a syntactic restriction of OWL 2 DL. None of the profiles is a subset of another. Apart from the above ones, there are many other possible OWL 2 profiles; in [46] a template for specifying additional profiles is provided.

### 2.3.6 Web Service Modeling Language (WSML)

The Web Service Modeling Language (WSML)\(^{15}\) is a formal language for the description of ontologies and Semantic Web services based on the conceptual model of WSMO (see section 3.2.2). WSML brings together different logical-language paradigms and unifies them into one syntactical framework.

WSML has five language variants based on five formalisms related to First-Order Logic and differing in logical expressiveness and in the underlying language paradigms:

• **WSML-Core** is based on the intersection of the DL SHIQ and horn Logic, called DLP (description logic programs) [50]. It has the least expressive power of all the WSML variants. The main features of the language are concepts, attributes, binary relations, and instances, as well as concept and relation hierarchies and support for datatypes.

• **WSML-DL** captures the description logic SHIQ(D).

• **WSML-Flight** is an extension of WSML-Core providing a powerful rule language, based on the Datalog subset of F-Logic.

• **WSML-Rule** extends WSML-Flight with further features from logic programming, such as the use of function symbols, unsafe rules and unstratified negation. It is based on the Horn subset of F-Logic.

• **WSML-Full** unifies WSML-DL and WSML-Rule under a First-Order umbrella with extensions to support the nonmonotonic negation of WSML-Rule. It is very expressive and undecidable.

As shown in Fig. 2.1, WSML has two alternative layerings: WSML-Core⇒WSML-DL⇒WSML-Full and WSML-Core⇒WSML-Flight⇒WSML-Rule⇒WSML-Full. In WSML interoperation between the description logic variant and the logic programming variants is only possible through a common core (WSML-Core) or a very expressive superset (WSML-Full).

WSML provides three syntaxes: a surface syntax, as well as XML and RDF syntaxes for exchange over the Web. The WSML syntax is divided into two parts: the conceptual syntax and the logical-expression syntax. The former is independent of the underlying logic and allows easy adoption of the language for those not familiar with logics. The use of the conceptual syntax does not depend on the chosen WSML variant, except for a few restrictions for WSML-Core and WSML-DL. The logical-expression syntax can be used when the full power of the underlying logic is required. Each WSML variant defines some restrictions on the logical syntax. The semantics of WSML ontologies is defined through a mapping between the

---

\(^{15}\)WSML related documents can be found at [http://www.wsmo.org/wsml/wsml-syntax](http://www.wsmo.org/wsml/wsml-syntax).
WSML logical syntax and the formalism which underlies the variant. WSML-Core and WSML-DL logical expressions are mapped to first-order logic; the expressions stay inside the expressiveness of the DL $SHIQ$. The semantics of WSML-Flight and WSML-Rule is defined through a mapping to F-Logic.

Figure 2.1: WSML Layering
Chapter 3

Semantic Web and Semantic Web services technologies

3.1 The Semantic Web

The Semantic Web is an evolving extension of the World Wide Web in which the semantics of information and services on the web is defined, making it possible for the web to understand and satisfy the requests of people and machines to use the web content. It derives from World Wide Web Consortium director Sir Tim Berners-Lee’s vision of the Web:

The Semantic Web is an extension of the current Web in which information is given well-defined meaning, better enabling computers and people to work in cooperation.[8]

The ”well-defined meaning” of information is provided by semantic descriptions, often referred to as metadata. Metadata describes information in terms of a domain vocabulary whose meaning is specified by a formal domain model (ontology). Metadata is expressed in a representation language that can be parsed and interpreted by machines (for further details on representation languages see section 2.3).

3.2 Semantic Web Services

Web Services are as a set of technologies that allow to implement the service-oriented architecture (SOA) on the Web. Services, implemented as web services, communicate by exchanging XML messages. Each service provides an interface that describes its input and output messages. Currently WSDL[15] is used as interface description language encapsulating XML Schema for describing the syntactical structure of messages but does not provide any constructs for describing the semantics of messages. Semantic Web Services combine Semantic Web and Web Services technologies to overcome the above limitations and enable automation of Web Services usage tasks, such as discovery, selection, composition, execution and so on. The use of Semantic Web Technologies, above all ontologies, for the description of Web Services has several advantages. First of all, reasoning support associated to formal ontology languages is a fundamental prerequisite for the automation of Web Services usage tasks. Secondary, ontologies reflect a shared view of a common domain; therefore the
use of ontologies for service descriptions can potentially increase the understanding and reusability of service descriptions.

A web service provider publishes XML schemes (source schemes), which are the structural description of messages that can be exchanged with the web service and binds this structural description with an ontology (target ontology) that provides the semantic description. Linking between the semantic and the syntactic description levels is commonly called grounding.

Assuming that the semantic information is separate from the syntactic description (WSDL), there are three options for placing the information specified in grounding:

1. Putting grounding in the semantic description document (OWL-S and WSMO);
2. Embedding grounding within WSDL (WSDL-S);
3. Externalising grounding in a third document (not supported by any current specification).

An overview of existing semantic web services approaches is given in the following sections.

3.2.1 OWL-S

OWL-S [44] is a OWL-based Web service ontology, which supplies web service providers with a core set of markup language constructs for describing the properties and capabilities of their web services. It is often referred to as a language for describing services, thus reflecting the fact that can be used together with the other aspects of OWL to create service descriptions.

The OWL-S ontology mainly consists of three interrelated sub-ontologies:

- The profile is used to express "what a service does", for purpose of advertising, constructing service requests and matchmaking;
- The process model describes "how the service works", i.e. what happens when it is used, to enable invocation, enactment, composition, monitoring and recovery;
- The grounding maps the constructs of the process model to detailed specifications of message formats, protocols and so on (normally expressed in WSDL).

All these sub-ontologies are linked to the top-level concept Service through the properties presents, describedBy and supports, as show in figure 3.1. The classes ServiceProfile (identifying the profile sub-ontology), ServiceModel (identifying the process model sub-ontology) and ServiceGrounding (identifying the grounding sub-ontology) are the respective ranges of those properties.

Service serves as a point of reference for declaring Web Services; whenever a service is declared an instance of the Service concept is created and present a ServiceProfile description, is describedBy a ServiceModel description and support a ServiceGrounding description.

The ServiceProfile provides means to describe the services offered by the providers and the services needed by the requesters. Using the OWL sub-classing it is possible to create specialized representations of services that can be used as service profiles. The ServiceProfile considers three basic types of information:
Figure 3.1: Top level elements of OWL-S [44]

- **The Organization that provides the service**: the contact information of the entity providing the service;
- **The function computed by the service**: the transformation produced by the service. The functional description includes the inputs required by the service and the outputs generated, the preconditions required by the service and the expected effects resulting from the service execution;
- **Features specifying service characteristics**, such as the category of the service (within the UNSPSC classification system) and the quality rating of the service.

The type of information that will play a key role during the service discovery is the specification of the functionality provided by the service. No schema to describe inputs/outputs/preconditions/effects (IOPE) instances is provided by the Service-Profile, but exists in the process model ontology. It is expected that the IOPEs published by the OWL-S Profile are a subset of those published by the Process. The interaction with the service can be viewed as a process and the ServiceModel class allows to define processes, i.e. to specify the ways a client can interact with a service. A process involves at least two parties: the client, from whose point of view the process is described, and the service the client deals with. The client and the service are referred to as participants.

### 3.2.2 Web service Modeling Ontology (WSMO)

The WSMO initiative\(^1\) has the aim of standardizing a unifying framework which provides support for conceptual modeling and formally representing services, as well as for automatic execution of services.

The WSMO approach considers three elements:

- the **Web service Modeling Ontology (WSMO)**, a conceptual model for Semantic Web Services;

\(^1\)http://www.wsmo.org
the Web Service Modeling Language (WSML), a language which provides a formal syntax and semantics for WSMO (see section 2.3.6);

• the Web Service Modeling Execution Environment (WSMX)\(^2\), an execution environment, which is a reference implementation for WSMO.

WSMO [40, 48] provides ontological specifications for the core elements of semantic web services. The elements of the WSMO ontology are defined in a meta-meta-model language based on MOF (Meta Object Facility).

WSMO identifies four top-level elements as the main concepts which have to be described in order to define semantic web services:

• **Ontologies**: provide the formal semantics for the terminology used within all other WSMO components.

• **Web Services**: semantic description for describing services requests by service requesters, provided by service providers and agreed between the above two. This description comprise the capabilities, interfaces and internal working of the service.

• **Goals**: objectives that a client may have when consulting a web service. This component describes aspects related to user desires with respect to the requested functionality and behaviour.

• **Mediators**: elements that handle interoperability problems between different WSMO elements. A WSMO Mediator connects the WSMO elements in a loosely-coupled manner and provides mediation facilities for resolving eventual mismatches in the process of connecting different elements.

The conceptual model of the WSMO elements is depicted in Figure 3.2.

![Figure 3.2: Upper WSMO elements [40]](image)

\(\text{http://www.wsmx.org}\)

## 3.2.3 Unified Service Description Language (USDL)

The Unified Service Description Language (USDL) is a platform-neutral language for describing services [14], proposed by SAP Research to cope with the shortcom-
ings of existing standards for service description (such as WSDL, WS-Policy, WS-Security, etc). The coverage of USDL is broader than that of, e.g., WS-* languages, as it also targets purely human/professional services (e.g. project management and consultancy), transactional services (e.g. purchase order requisition), informational services (e.g. spatial and demography look-ups), software components (e.g. software widgets for download), digital media (e.g. video & audio clips), platform (e.g. middleware services such as message store-forward) and infrastructure (e.g. CPU and storage services).

Consolidating the experience gained in several research projects on services, USDL aims to provide a normative and balanced unification of all the different dimensions of service information that are not captured by existing languages. These dimensions, corresponding to language modules, cover legal, service level agreements, pricing, functional and technical aspects.

Recent proposals aim to include information on the security characteristics of a service by extending the Service Level module of USDL[43]. More details on USDL are given in D1.1[42].

3.2.4 Linked Open Services (LOS)

Linked Open Services (LOS)\(^3\) are an approach to exposing services on the Web using the same technologies that are associated with Linked Data, in particular HTTP, RDF and SPARQL.

Although Semantic Web Services are currently oriented towards extending the WS-* stack (based on the XML-based languages WSDL and SOAP, and built on to form UDDI and BPEL) with ontology-based descriptions, there is also a strong movement away from this underlying stack towards resourceful (RESTful) web services\(^4\). The REST model adopts the HTTP Web resource model of HTTP and abandons the WS-* stack languages. Among the adopters of this approach is the Linked Open Data (LOD) community who seeks to increase the amount of Semantic Web-style content, in the form of RDF, online. Linked Data exposes many datasets via passive SPARQL endpoints. Linked Open Services (LOS) combines LOD endpoints and RESTful services in general, with services based on the WS-* stack, and makes service descriptions based primarily on RDF and SPARQL\(^4\).

3.2.5 Semantic Annotations for WSDL and XML Schema (SAWSDL)

The W3C Working Group on Semantic Annotations for WSDL and XML Schema (SAWSDL) developed mechanisms with which semantic annotations can be added to WSDL components. The SAWSDL specification [23] became a W3C recommendation on August 28, 2007.

SAWSDL defines a set of extension attributes for the Web Services Description Language (WSDL) and XML Schema definition language that allows description of additional semantics of WSDL components. The defined extension attributes fit within the WSDL 2.0, WSDL 1.1 and XML Schema extensibility frameworks.

Unlike OWL-S or WSMO, SAWSDL does not specify a new language or ontology for semantic service description but simply provides mechanisms by which ontological

---

\(^3\)http://www.linkedopenservices.org

\(^4\)http://www.w3.org/TR/rdf-sparql-query/
concepts that are defined outside WSDL service documents can be referenced to semantically annotate WSDL description elements.
The design principles for SAWSDL are:

- the specification enables semantic annotations of Web Services using the existing extensibility framework of WSDL;
- it is agnostic to semantic (ontology) representation languages;
- it enables semantic annotations for Web Services not only for discovering Web Services but also for invoking them.

Based on these design principles, SAWSDL defines three new extensibility attributes to WSDL 2.0 elements for their semantic annotation: modelReference, that allows to relate a WSDL component and a concept, liftingSchemaMapping and loweringSchemaMapping, both allowing to specify mappings between semantic data in the domain referenced by modelReference and XML.
The SAWSDL specification also provides some example illustrating the usage of the extension attributes.

3.2.6 DIANE

The DIANE project\(^5\), funded by the priority research program (SPP) no. 1140 of the German Research Community (DFG), aimed at developing and evaluating concepts that allow for an integrated, efficient, and effective use of resources in the form of services in ad hoc networks.
The project developed its own description language, the DIANE Service Description (DSD) \([39]\). DSD is a service description language based on its own light-weight ontology language that is specialized for the characteristics of services and can be processed efficiently at the same time.

DSD makes an explicit distinction between service requests and service offers and describes services exclusively by their state change. The basis for DSD is standard object orientation extended by four additional elements:

- **operational** elements. These elements allow to express the capacity of services to change the state of the world (world-altering).

- **aggregating** elements. The effect of a service (request or offer) is a set of states captured by aggregating elements. For offers these are the states the service can potentially create, for requests these are the states the requester is interested in.

- **selecting** elements. These elements allow to choose upon invocation which effect offered by the service should be provided in a specific instance.

- **rating** elements. These elements are used only in service requests and are the main feature used to choose among competing offers.

Within the project, the DIANE middleware has been built around DSD to facilitate the efficient usage of the language.

---

\(^5\)http://hnsp.inf-bb.uni-jena.de/DIANE/en/
3.2.7 SOAF

(SOAF – Service Of A Friend) proposes the integration of services into social networks by linking humans and services in a common network structure [54]. The described approach follow the spirit of FOAF (Friend Of A Friend)⁶, a machine-readable ontology expressed in RDF and OWL describing persons, their activities and their relations to other people and objects. This approach brings the need to represent:

- Links between services and humans;
- Dynamic changes, because there are relations existing only over a certain time;
- Past relations providing useful information for potential future use.

The idea described in [54] is to augment FOAF network structures with service related information and to link services and humans in the same network. In SOAF⁷ the relation mechanism of FOAF has been extended to model relations between services and persons. SOAF defines the classes Person, Service and Organization and concentrate on the relations between them, as shown in figure 3.3 Benefits of a service-enriched social network structure are described with the support of a scenario illustrating these network structures applied to service discovery.

---

⁶http://www.foaf-project.org/
⁷http://www.infosys.tuwien.ac.at/staff/treiber/soaf/index.rdf
Chapter 4

Security ontologies

4.1 Existing ontologies

Much research has been done on ontology-based representation of security concepts and properties. A review of existing security ontologies has been given in [9, 10]. The work presented in the present document concentrated on studies investigating the aspects described in section 1.2, as reported in table 4.1.

4.1.1 OWL-S Security Ontologies

The OWL-S Security Ontologies \footnote[1]{http://www.ai.sri.com/daml/services/owl-s/security.html} is a set of ontologies to describe the security requirements and capabilities of OWL-S semantic web services and agents. They are the port to OWL and OWL-S of a previous work done for DAML and DAML-S web services [20]. The OWL-S Security Ontologies collection is formed by seven sub-ontologies:

- Credential Ontology: to specify access control restriction of web services requesting authentication for authorised access. It defines both simple credentials (e.g. Login, BioMetric) and composed credentials (e.g IDCard or SmartCard). X.509 Certificates are represented as a subclass of more general Certificate class.
- Security Mechanisms Ontology: contains the classes describing the basic security mechanisms (e.g. Signature, Key Protocol) that are the basis to describe

<table>
<thead>
<tr>
<th>Study</th>
<th>Certification</th>
<th>Security properties</th>
<th>Service model</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWL-S Security Ontologies</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NRL Security Ontology</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Common Criteria Ontology</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PDO Ontology</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Security Ontology</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>SecOnt Ontology</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>SEKT Ontology</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>SCL Ontology</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.1: Existing ontologies classified by area of interest
the security requirements and capabilities of web services and agents. Security properties (e.g. Anonymity, Confidentiality) are defined as instances of the class Security Notation. A set of restriction classes (e.g. Authentication-SubClass) are provided as the basis to describe the security requirements and capabilities of web services and to decide whether security requirements and capabilities are in the appropriate inheritance relationship by means of the subsumption reasoning made available by Description Logic reasoners.

- Service Security Extension Ontology: defines the classes and object properties required to describe the security requirements and capabilities of an OWL-S web service.
- Agent Security Extension Ontology: defines the classes and object properties required to describe the security requirements and capabilities of an OWL-S web service consumer.
- Privacy Ontology: defines the classes to enable unambiguous interpretation, between an OWL-S service and a consumer, of the privacy policies exchanged at the beginning of an interaction.
- Cryptographically Annotated Information Object Ontology: to describe encryption or signature requirements of input or output parameters of OWL-S web services.
- Context-Sensitive Trust Ontology: defines the concepts to define trust policies used to derive trustworthiness assertion about the messages exchanged in a communication.

[19] proposes to use the above mentioned ontologies to interpret security and privacy policies providing the specification of who can use a service under which conditions, how information should be provided to the service and how provided information will be used later. It was proposed the extension of OWL-S to include policies and the inclusion of a property (called PolicyEnforced) for describing the different policies that must be enforced for the correct execution of a service. Three kinds of policies are addressed: authorization, privacy and confidentiality. To describe policies it was used Rei², a logic-based language for policy specifications based on RDF and RDFS.

### 4.1.2 NRL (Naval Research Laboratory) Security Ontology

The work in [37], coming from the military area, highlights some limitations, such as lack of intuitiveness and expressiveness, in the DAML Security Ontology by [20]. To improve upon these limitations a set of security-related ontologies, collectively referred to as the NRL Security Ontology, is introduced. The NRL Security Ontology describes security concepts at various levels of detail; written in OWL, it is made up of seven separate ontologies:

1. Main security ontology: an ontology to describe security concepts;
2. Credentials ontology: an ontology to specify authentication credentials;
3. Security algorithms ontology: an ontology to describe various security algorithms;

[http://rei.umbc.edu](http://rei.umbc.edu)
4. Security assurance ontology: an ontology to specify different assurance standards;

5. Service security ontology: an ontology to facilitate security annotations of semantic web services;

6. Agent security ontology: an ontology to enable querying of security information;

7. Information object ontology: an ontology to describe security of input parameters of web services.

The last three ontologies are based on existing DAML security ontologies while the others are new.

The Security Assurance ontology provides a way to describe standardized assurance methods for security protocols, mechanisms and algorithms and considers concepts related to certification aspects. In fact the 'Assurance' class is classified according to different assurance methods: 'Standard', 'Accreditation', 'Evaluation' and 'Certification'. Even though this ontology is the less complete among the seven ones, it includes classes to describe Common Criteria and TCSEC (Trusted Computer System Evaluation Criteria) certifications.

The Service Security ontology was developed to provide a link to OWL-S, in order to use the NRL Security Ontology in the Web services context. Therefore a way to describe Web services is also provided.

All ontologies show quite a level of detail, above all in the areas of encryption algorithms and credentials. However, a core that shows the connections between concepts is missing.

4.1.3 Common Criteria ontology

[22] describes a Common Criteria (CC) ontology formalised in OWL focusing on security assurance requirements relevant for the CC evaluation. The CC Ontology was created primarily to be used by a tool (CC Ontology Tool) aimed to support the evaluator at the Common Criteria certification process. The CC Ontology Tool relies the Security Ontology [25] to let its users to extract keywords denoting security related concepts (e.g. threats and countermeasures) The CC Ontology is not entirely shown in the paper and it is not publicly available, therefore it would be more appropriate to refer to it as a knowledge base. The CC ontology concentrated on modelling the security assurance requirements because they are used by the evaluator as a mandatory statement of evaluation criteria when determining the assurance of the Target Of Evaluation (TOE). Following the notation used in CC, that classify security requirements in classes, families, components and elements, all Security Assurance Requirements (SAR) reported in CC part 3 are represented as instances of the 'Component' class (linked to the concepts 'Family' and 'Class').

4.1.4 Problem Domain Ontology (PDO)

[41] introduces a problem domain ontology built from regulatory documents enforced by the Department of Defense Information Technology Security Certification

3http://www.commoncriteriaportal.org/
The DITSCAP defines certification in the context of information systems as a comprehensive evaluation of the technical and non-technical security features of an information system and other safeguards made in support of the accreditation process, to establish the extent to which a particular design and implementation meets a set of specified security requirements. It is typically required for connection to Department of Defense, and other Federal systems, networks, and applications. The DITSCAP has been replaced by the Department of Defense Information Assurance Certification and Accreditation Process (DIACAP) in 2006.

The early effort on DITSCAP automation described in [41] provides a generic framework supporting the Certification and Accreditation (C&A) process. The DITSCAP requires multitude of directives, security requisites and other regulatory documents, where security requirements are specified in natural language. The specification of a PDO security requirement typically needs to identify: (1) the protected asset, (2) the threats it is driven by, (3) the vulnerabilities it prevents, (4) the suggested countermeasures, (5) the mission criticality that it is subject to, (6) its source, (7) the goal of the security requirements (i.e. the Information Assurance Level: confidentiality, integrity, availability), (8) the related stakeholders, (9) other domain-specific concepts that need to be considered.

To achieve an unambiguous and uniform understanding of DITSCAP security requirements, the authors define a common language developed through a systematic methodology for extracting and organizing problem domain concepts expressed in natural language regulatory documents (usually ranging from 25 to 200 pages with cross-referencing to each other). Such common language establishes a Problem Domain Ontology (PDO), that is a hierarchical organization of ontological concepts capturing well-defined dimensions of the problem domain with related properties and non-taxonomic dependencies among them. To support the representation of the structures required by the PDO the GENeric Object Model (GenOM) toolkit\(^4\) has been used.

### 4.1.5 The Security Ontology

[25] presents an ontology to support information security risk management. The ontology is formalised in OWL. The proposed ontology is divided into three sub-ontologies: Security, Enterprise and Location.

The Security Subontology defines five core concepts Attribute, Control, Threat, Vulnerability and Rating from which all other concepts in the sub-ontology are derived. The Attribute concept is used to represent the attributes of other concepts (control, threat, vulnerability and rating) and their derivations.

An example of concept specialising the Attribute concept is SecurityAttribute which is used to represent the concept of security property like confidentiality, integrity, accountability. Actually, within the Security Ontology, security properties are represented as OWL Individuals of the SecurityAttribute class. Other examples of classes derived from Attribute are ControlType, ThreatOrigin, ThreatSource and the Severity Scale concept used to rate threat impact, control effectiveness or severity of vulnerabilities.

---

\(^4\)GenOM is an integrated development environment developed by the authors that allows to create, browse, access, view and query associated knowledge bases. It is founded on the frame representation and is compatible with OWL.
Within the Security sub-ontology the Control concept represents the measures used to mitigate defined vulnerabilities. A Control is characterised by its control type (e.g. corrective, detective), related security standard, implementation specification (i.e. the assets required to implement the control) and the vulnerabilities that could be mitigated by the control.

The Vulnerability concept is used to represent lack of controls that could be exploited by a threat and its instances are modelled after the German IT Grundschutz Manual which is also the source used to model the instances (e.g. Denial Of Service attack, untrained personnel, alteration of software) of the Threat class.

The Security Ontology assumes that the objective of information security risk analysis is to protect the business missions of an organisation against threats. The Enterprise sub-ontology is intended to support this approach. It defines the concepts of Asset, Person and Organisation. Within the Enterprise sub-ontology the Asset concepts is the basis for the description of the physical infrastructure of an organisation using OWL properties defined on purpose (e.g. contains, isAbove, isUnder). Assets are related to Enterprise by means of the OWL Property ownedBy.

The Location sub-ontology defines the only concept of Location which is used by the threat probability rating concept to relate threats information, threat probability figures and physical locations.

The overall Security Ontology has been evaluated using a methodology based on the approach proposed by [56]. The Security Ontology first undergone a two round evaluation by experts from several security domains (e.g. business continuity management, multi-objective decision support for control selection). This first lead to some changes to initial version of the ontology. In a second stage the experts developed informal competency questions aimed to assess the capabilities of the security ontology w.r.t. information security risk management support (e.g. which asset are required to implement a given control?). The informal competency questions were formalised in terms of OWL class definitions, SPARQL queries or programs using Protege OWL API. In the last phase of the evaluation the resultset of the run of the formalised competency questions were evaluated by the team of experts to identify shortcomings.

4.1.6 Linkpings Universitet Security Ontology (SecOnt)

The security ontology presented in [29], available on-line, is proposed as a general vocabulary of the domain of the information security. It is built upon four basic components of risk analysis: assets, threats, vulnerabilities and countermeasures. Nevertheless, it also defines concepts concerning access control models (RBAC, MAC, DAC, etc.), cryptographic models, security goals (authentication, confidentiality and integrity goals), vulnerabilities, products (databases, Operating Systems, etc.), strategies and so on. The ontology has been developed in OWL by modeling the four basic building blocks and their relations, and refining each block with technical concepts.

Being publicly available, the ontology is eventually extensible with new elements and the authors provide several examples of how further extensions can be achieved.

http://www.ida.liu.se/iislab/projects/secont/
4.1.7 SEKT Ontology

[53] focuses on security aspects of business processes in collaborating organizations (the so-called virtual organizations (VOs)). Security infrastructure and mechanisms should meet VO security requirements across the multiple administrative domains but security setups are usually domain (i.e., organization) specific and not meant to interact across domains. The approach described in the article to overcome this problem is to provide a semantic layer on top of existing security infrastructures, using concepts of ontologies for describing domain-specific security properties. The exemplary model adopted to represent domain-specific security setup is a Public Key Infrastructure (PKI). A PKI is an infrastructure based on a trusted third party certifying for identities of principals (e.g., company employees), by binding public keys to them. The X.509 standard for digital certificates is used for certificates structure. The proposed ontology allows to declaratively describe domain-specific security properties, define security logic supporting business processes in form of rules and infer security properties at run time, e.g., derive whether employee A from Organization A is allowed to perform a task with employee B from Organization B. The ontology, formalised in F-Logic to model rules, considers also concepts related to the X.509 standard such as Certificate and CertificationAuthority.

4.1.8 security Characterization Language (SCL) ontology

[33] proposes to describe systems-level security contracts (SsC) of component-based systems using the security characterization language (SCL). A system-level security contract (SsC) is a security property provided by a component resulting from the composition of lower level components. The SCL is a rule-based language following the logic programming paradigm. Security properties are defined by means of inference rules expressed in SCL.

The formalisation of systems-level security contracts (SsC) by means of SCL aims to enable automatic reasoning about compatibility of security contracts.

A SsC is expressed in terms of a collection of Compositional Security Contracts (CsC). Each CsC in the collection is a binary relation expressing the security properties required and ensured by two components (i.e., a service consumer and a service provider) directly interacting within a composition.

Systems-level security contracts are intended to support both end-users of the composite service and software integrators assembling lower level components into an higher level composite components. End-users exploit SsC to know what security objective is achieved by a component. Software integrators exploit SsCs to formalise the security property (i.e., the SsC) of the composite component.

[57, 34, 35] proposes an extension of SCL (E-SCL) that introduces features to express security requirements of Web Services. Web Services compositions are recognised to have a much more dynamic nature than components. In a component-based architecture low level components are statically bound within a composition. On the contrary, composition based on Web Services have a much more dynamic nature (e.g., a Web Services can be replaced within a composition at run-time because it does not longer meet the expected Quality of Service (QoS)). Therefore E-SCL introduces features such as time reasoning or runtime conditions (i.e., dependency on actual bandwidth) to express the validity period of a security properties. Other
features introduced by E-SCL include probability reasoning and alternative security properties.

This work presents an ontology intended to support security-oriented runtime binding between clients and web services. In particular a Web Service can publish, via the OWL-S `serviceParameter` property, the security properties it provides in terms of E-SCL security functions.

The proposed ontology, partially based on [37] and the Common Criteria (CC), serves both the scope to share the E-SCL vocabulary appearing in OWL-S service profiles and to map between E-SCL security functions and security objectives (e.g. integrity of the exchanged messages), algorithms (e.g. key exchange algorithm), concepts and credentials.

Therefore a Web Service client could use the ontology to assess the compliance between its desired security objective (e.g. integrity of a message) and the security property described within the OWL-S profile of the service in terms of E-SCL functions.

### 4.2 Comparison of security ontologies

The security ontologies presented within this chapter can be contrasted with respect to different aspects:

- **Targeted user community.** The OWL-S, NRL, SEKT and SCL ontologies have been conceived to support matchmaking between Service providers and consumers. CC and PDO are aimed to support Certification Authorities and Evaluation bodies in the context of certification and evaluation. SecOnt and Security Ontology are primarily targeted to support Security Experts and Risk Managers.

- **Representation formalism.** Most of the reviewed ontologies are formalised in OWL. Only two (PDO and SEKT) use alternative formalisms (GenOM and F-Logic respectively).

- **Type of certificate described.** Common Criteria Certificates are described by both NRL and CC ontologies. The NRL ontologies deals with TSEC certificates as well. DITSCAP certificates are described by the PDO ontology. X.509 certificates are described by SEKT security ontology. The SCL ontology propose a non-standard Compositional Security Contract (CsC) specifically designed to support automatic reasoning about service compositions.

- **Security Properties Model.** Most of the ontologies formalised in OWL represent Security Properties as OWL Individual. Only OWL-S Security defines OWL classes with the aim to support automatic subsumption reasoning about Security Properties. SEKT Security and SCL define security properties by means of rules of their respective formalism (i.e. F-Logic and Security Characterisation Language (SCL)). The CC ontology does not cope with the description of Security Properties.

- **Service Model.** The Security and SEKT ontologies provides concepts to describe an Enterprises in term of both its organisational elements (e.g. roles) and infrastructural (e.g. hardware) assets. SecOnt defines only concepts to...
represent the infrastructural assets. OWL-S Security, NRL and SCL are supposed to be used in the context of the OWL-S framework and rely on that for the concepts needed to describe services. The CC and PDO ontologies do not provide means to describe services.

- Public Availability. Only OWL-S Security, NRL, SecOnt and Security ontologies are publicly available.

Given the above comparison the OWL-S Security, NRL and SCL ontologies suggests a closer look during the development of the ASSERT4SOA ontology. Indeed all of them target the same problem of the ASSERT4SOA framework (i.e. determine secure composability of web services). All of them use a standard formalism (OWL). Because of its formalisation of CC and TSEC certificates the NRL ontology is worth to be taken into account when designing the parts of the ASSERT4SOA ontology aimed to support the production of ASSERTS starting from existing certificates (e.g. CC Certificates). OWL-S Security and SCL gives indication about how security properties could be described within the ASSERT4SOA ontology in a way suitable to support automatic reasoning about security properties. Unfortunately for SCL it will be possible to rely only on the available literature since the ontology is not available per se.
Part II

Requirements
Chapter 5
Functional requirements

5.1 Overview

This section describes the requirements for the ASSERT4SOA Ontology.

In this document we use the term "Ontology" as defined in [52]: "An ontology is a formal explicit specification of a shared conceptualization of a domain."

The structure of this chapter is informed by structure of the Ontology Requirements Specification Document (ORSD) proposed by the NeOn EU Project\(^1\). An ORSD relies on the Competency Questions technique to establish the ontology requirements\[^1\]. Competency Questions are natural language questions that the ontology to be built should be able to answer. The approach based on Competency question has been also adopted for the specification of some of the Security Ontologies presented in the previous chapter (e.g. [25]).

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted in a manner similar to that described in [13].

5.2 Purpose

The ASSERT4SOA Ontology MUST provide a consensual knowledge model of the Software Security Certification domain that SHOULD be used by the ASSERT4SOA framework.

The ASSERT4SOA Ontology SHOULD enable the interoperability and comparison of ASSERT-E and ASSERT-M certificates in the ASSERT4SOA framework.

5.3 Scope

The ASSERT4SOA Ontology MUST focus on the description of [4]:

\[^1\]http://www.neon-project.org
• Security Certification Properties for services (Objective 1)
• ASSERT4SOA Security Certificates (ASSERTs): ASSERT-E, ASSERT-O and ASSERT-M (Objective 2)

The ASSERT4SOA Ontology MAY focus on the description of Software Services.

5.4 Level of Formality

The ASSERT4SOA Ontology MUST be implemented in OWL 2 language.

5.5 Intended Users

The ASSERT4SOA Ontology conceptualisation SHOULD be agreed among the following roles:

• Security Engineer: professional concerned with tools, processes and methods needed to build systems remaining dependable in the face of malice, error, or mischance [1].
• Software Engineer: professional concerned with developing software products (i.e., software which can be sold to a customer) [51].
• Certification Authority: an entity trusted by one or more users to create and assign certificates [21].
• Evaluation Body: accredited lab carrying out evaluations. The evaluation outcome is a pre-requisite for issuing a certificate on the part of the certification authority [21].
• Service Provider: an organisation supplying services to one or more Internal Customer or Externals Customers [32].
• Service Consumer: a person who uses IT Service on a day-by-day basis (User [32]) or someone who buys a service (Customer [32]).

5.6 Intended Uses

The ASSERT4SOA Ontology SHOULD support the following use cases:

• Publish the definition of a Security Property (e.g. integrity of messages). Security expert publishes the definition of a Security Property within the ASSERT4SOA Ontology.
• Search the definition of a Security Property. A software developer searches the definition of a Security Property that she is requested to be guaranteed by the software artefact being designed/developed.
• Discovery of relations (e.g. subsumption) between the definition of two Security Properties. A service consumer uses the ASSERT4SOA Ontology to verify that the security properties asserted by an ASSERT4SOA Security Certificate (ASSERT) meets the expected security requirements.
• Publish an ASSERT4SOA Security Certificate (ASSERT) for a Software Service. A Service Provider publish on the ASSERT4SOA framework the ASSERT certifying that a Software Service meets a security requirement.

• Search the Security Properties certified for a given Software Service. A service consumer uses the ASSERT4SOA Ontology to search the security requirements met by a Software Service.

• Search the proofs supporting the Security Property certified by an ASSERT.

• Discovery of relations (e.g. equivalence) between ASSERTs. A service consumer uses ASSERT4SOA Ontology to compare different types ASSERT4SOA certificates.

• Check consistency between the definition of the Security Property certified by an ASSERT and the Service Model attached to the same ASSERT.

5.7 Representational Requirements (Groups of Competency Questions)

5.7.1 Requirements on security properties

Ontology-based properties

Requirement PO1.FUN

Description: The ASSERT4SOA Ontology MUST support the representation of Security Properties (Ontology-based security properties). The ASSERT4SOA Ontology MUST support the answer to the following competency questions about Security Properties:

- What is the name of a security property?
- Which are the parameters of a security property?
- What is the definition of a security property?

Event\use case: Authoring of a security property

Rationale: Security engineers needs to give a name to a security property in order to let other actors to refer it within system documentation (e.g. within a system requirements document). A security property is also identified by the type and order of its arguments (i.e. its signature). As an example the security property "confidantiality of a message" is identified by a name (confidantiality) and number (1) and type of its parameters (first parameter is of type message).

Source: ASSERT-O

Evidence-based properties

Requirement PE1.FUN

Description: The ASSERT4SOA Ontology SHOULD support the representation of Security Properties that can be proven to hold for a given service through a testing process (Evidence-based security properties). The ASSERT4SOA Ontology SHOULD support the answer to the following competency questions about Evidence-based Security Properties:
• What is the name of a concrete (testable) security property?
• Which is the abstract security property related to a concrete security property?
• Which are the class attributes characterising a concrete security property?
• What is the name of a class attribute?
• Which is the value of a class attribute?
• Does security property $a$ imply security property $b$?
• Which type of relationship link security property $a$ to security property $b$: intra-property or inter-property?

**Event use case:** Authoring of a security property

**Rationale:** The ASSERT4SOA Ontology should describe security properties for services. In general, security properties reported in different types of certificates (ASSERT-O, ASSERT-E and ASSERT-M) can be defined differently depending on the certification process followed.

**Source:** ASSERT-E

**Supporting material:** D4.1[2]

**Model-based properties**

**Requirement PM1.FUN**

**Description:** The ASSERT4SOA Ontology SHOULD support the representation of Model-based security properties. The ASSERT4SOA Ontology SHOULD support the answer to the following competency questions about formal definition of security properties:

• What is the name of a security property?
• Which are the names and sorts of the variables of a security property?
• What is the formula defining a security property?

**Event use case:** Authoring of a security property

**Rationale:** ASSERT-M properties are identified by a name and name and number and type of parameters (e.g. The Authenticity formal definition is identified by a name (authenticity) and the parameters tuple ¡set of actions, agent, sequence of actions, initial knowledge, local view of agents¿)

**Source:** ASSERT-M

**Supporting material:** D5.1[26]

**5.7.2 Requirements on certificates**

**Ontology-based certificates (ASSERT-O)**

**Requirement AO1.FUN**

**Description:** The ASSERT4SOA Ontology MUST support the representation of Ontology-based Certificates (ASSERT-O). The ASSERT4SOA Ontology MUST support the answer to the following competency questions about Ontology-based Certificates (ASSERT-O):
• Which are the security statements of an ASSERT-O?
• What is the proof supporting an ASSERT-O security statement?

**Event \use case:** Compare requested with provided ASSERT  
**Rationale:** ASSERT-O provides the proof that the referenced web service provides the asserted security property so that a service consumer can re-check it if needed.  
**Source:** ASSERT-O

**Requirement AO2.FUN**  
**Description:** The ASSERT4SOA Ontology MUST support the representation of ASSERT-O security statements. The ASSERT4SOA Ontology MUST support the answer to the following competency questions about ASSERT-O security statements:

• What is the service certified by a security statement?  
• What is the security property asserted by a security statement?  
• What is the proof supporting a security statement?  

**Event \use case:** Compare requested with provided ASSERT  
**Rationale:** A service consumer has the need to verify that the ASSERT is about the service she intend to consume, the property certified for it, and the proof supporting the assertion.  
**Source:** ASSERT-O

**Requirement AO3.FUN**  
**Description:** The ASSERT4SOA Ontology MUST support the representation of ASSERT-O proofs. The ASSERT4SOA Ontology MUST support the answer to the following competency questions about ASSERT-O proofs:

• What is the service model used in a proof?  
• Which are the security controls considered in a proof?  
• What is the reasoner used to verify a proof?  

**Event \use case:** Compare requested with provided ASSERT  
**Rationale:** A service consumer, in order to re-check a proof, needs the the service model, the security controls providing the security property and the reasoner used by the evaluation body to check that the security controls are consistent with service model.  
**Source:** ASSERT-O

**Requirement AO4.FUN**  
**Description:** The ASSERT4SOA Ontology MUST support the representation of security controls. The ASSERT4SOA Ontology MUST support the answer to the following competency questions about security controls:

• What is the security pattern matched by a security control?  
• How a security control maps a service implementation against a security pattern?
• How a security control realises the control objectives of a security pattern?

• Which are the elements of a service implementation realising to a security control?

Event use case: Compare requested with provided ASSERT

Rationale: ASSER-O assumes that service provides a security property by implementing a security pattern. A security control maps a security pattern on a service model.

Source: ASSERT-O

Requirement AO5.FUN

Description: The ASSERT4SOA Ontology MUST support the representation of security patterns. The ASSERT4SOA Ontology MUST support the answer to the following competency questions about security patterns:

• What is the security property provided by a security pattern?

• Which are the static elements of a security pattern?

• What is the expected behaviour of a security pattern?

Event use case: Issue of a security certificate, compare requested with provided ASSERT

Rationale: Literature suggests that a security pattern must contain, among the other things, a description of the problem addressed by the pattern, the structural elements contributing to the solution, and the expected interactions among those elements.

Source: ASSERT-O Supporting material:[50]

Evidence-based certificates (ASSERT-E)

Requirement AE1.FUN

Description: The ASSERT4SOA Ontology SHOULD support the representation of Evidence-based Certificates (ASSERT-E). The ASSERT4SOA Ontology SHOULD support the answer to the following competency questions about Evidence-based Certificates (ASSERT-E):

• Which is the service certified in the ASSERT-E?

• Which is the (test based) model of the certified service?

• Which are the (concrete) security properties tested by the ASSERT-E?

• Which is the test based evidence supporting the ASSERT-E?

• Which are the metrics used to evaluate the evidence(s) provided by the ASSERT-E?

Event use case: Issue of a security certificate

Rationale: The ASSERT4SOA Ontology should describe all types of ASSERT4SOA Security certificates (ASSERTs).

Source: ASSERT-E

Supporting material: D4.1[2]
Requirement AE2.FUN
Description: The ASSERT4SOA Ontology SHOULD support the representation of ASSERT-E proofs, called test-based evidences. The ASSERT4SOA Ontology SHOULD support the answer to the following competency questions about test based evidences:

- Which is the evaluation lab that executed the tests?
- Which are the testing practices used during the evaluation process?
- Which are the test cases executed during the evaluation process?
- Which are the types of test considered to evaluate the security properties included in the ASSERT-E?
- Which are the classes of tests considered to evaluate the security properties included in the ASSERT-E?
- Which are the results of the test execution?
- Which is the test generation model used during the evaluation process?

Event\use case: Issue of a security certificate, compare requested with provided ASSERT
Rationale: All ASSERTs are related to type-specific proofs proving the statements included in the certificates. In order to describe all types of ASSERTs, the ASSERT4SOA ontology should also represents their proofs.
Source: ASSERT-E
Supporting material: D4.1[2]

Requirement AE3.FUN
Description: The ASSERT4SOA Ontology SHOULD support the representation of concepts related to tests. The ASSERT4SOA Ontology SHOULD support the answer to the following competency questions about tests:

- Which is the input of a test case?
- Which is the expected output of a test case?
- Which are the pre-conditions and post-conditions of a test case?
- Which are the hidden communications included in a test case?
- Which test type does the test case represent?
- Which class of test does the test type belong to?
- Which are the attributes of a class of tests?
- Does test type \( t_i \) imply test type \( t_j \)?
- Which test generation model has been used to produce the test cases?
**Event use case:** Issue of a security certificate, compare requested with provided ASSERT

**Rationale:** ASSERT-Es prove that some test-related assurance activities have been carried out on a service. In order to describe test-based evidences, the ASSERT4SOA Ontology needs to represent test-related concepts.

**Source:** ASSERT-E

**Supporting material:** D4.1[2]

---

**Model-based certificates (ASSERT-M)**

**Requirement AM1.FUN**

**Description:** The ASSERT4SOA Ontology SHOULD support the representation of Model-based certificates (ASSERT-M). The ASSERT4SOA Ontology SHOULD support the answer to the following competency questions about Model-based certificates:

- What is the security property instantiation asserted in an ASSERT-M?
- What is the system model (ASSERT-M Model) exposed in an ASSERT-M?
- What is the system model (verification model) that is the basis to prove that a service provides a security property instantiation?
- Which are the property preserving homomorphisms that allow to prove that an ASSERT-M model is indeed the abstraction of the respective verification model?

**Event use case:** Issue of a security certificate

**Rationale:** Questions elicited from the currently available documentation (see Supporting Material).

**Source:** ASSERT-M

**Supporting material:** D5.1[26]

---

**Requirement AM2.FUN**

**Description:** The ASSERT4SOA Ontology SHOULD support the representation of Security Property Instantiations. The ASSERT4SOA Ontology SHOULD support the answer to the following competency questions about Security Property Instantiations:

- What is the Security Property instantiated by a Security Property Instantiation?
- Which are the values for the variables of a Security Property in a Security Property Instantiation?
- What is the system model (Verification Model) verifying the security property instantiation certified by a Model-based certificate (ASSERT-M)?
- What is the proof that a verification model verifies a security property instantiation?

**Event use case:** Issue of a security certificate, compare requested with provided ASSERT
Rationale: Questions elicited from the currently available documentation (see Supporting Material).
Source: ASSERT-M
Supporting material: D5.1[26]

5.7.3 Requirements on service model ontology

The aim to use ontology based reasoning to verify the compatibility of a service model with a security requirement lead to the need to provide the ASSERT4SOA Ontology with means to describe service implementations. This sections states the requirements that must be fulfilled to enable the ASSERT4SOA Ontology to support the description of service models.

ASSERT-O service model

Requirement MO1.FUN
Description: The ASSERT4SOA Ontology MUST support the representation of service implementations. The ASSERT4SOA Ontology MUST support the answer to the following competency questions about service implementations:

- What is the interface offered by a service implementation?
- Which are the functions implemented by service?
- Which are the roles of a service implementation?
- Which are the relationships existing between the roles of a service implementation?
- What is the workflow of a service implementation?
- Which are the resources needed by a service implementation?
- Which are the constraints over the resources used by a service implementation?
- Which are the contexts of use of resources within a service implementation?
- Which are the security controls implemented within a service implementation?

Event use case: Check consistency between a security requirement and a service model
Rationale: Elicited from the literature about Semantic Web Services and Security Ontologies reviewed in this document.
Source: ASSERT-O
Supporting Material: [44], [24], [25]

Requirement MO2.FUN
Description: The ASSERT4SOA Ontology MUST support the representation of services interfaces. The ASSERT4SOA Ontology MUST support the answer to the following competency questions about services interfaces:

- Which are the operations of an interface?
• What is the name of an operation?
• What is the message exchange pattern of an operation?
• What is the style of an operation?
• Which are the formal input parameters of an operation?
• What is the output of an operation?
• Which are the possible faults of an operation?

**Event\use case:** Check consistency between a security requirement and a service model

**Rationale:** Elicited from literature about WSDL.

**Source:** ASSERT-O

**Supporting Material:** [15]

**Requirement MO3.FUN**

**Description:** The ASSERT4SOA Ontology MUST support the representation of business objectives pursued by services. The ASSERT4SOA Ontology MUST support the answer to the following competency questions about business objectives:

• What is the input needed to achieve a business objective?

• What is the output representing the achievement of a business objective?

• Can a business objective be decomposed in terms of finer grain objectives?

• Does a decomposition represent a full description of a business objective?

• What is the order in which the business objectives in a decomposition must be achieved?

**Event\use case:** Check consistency between a security requirement and a service model

**Source:** ASSERT-O

**Requirement MO4.FUN**

**Description:** The ASSERT4SOA Ontology MUST support the representation of roles. The ASSERT4SOA Ontology MUST support the representation of relationships between roles (organisational relationships). The ASSERT4SOA Ontology MUST support the answer to the following competency questions about organisational relationships:

• What is the depender role?

• What is the dependee role?

• What is the type of relationship existing between two roles?

• What is the dependum creating a dependency between a depender and a dependee?
**Event use case:** Check consistency between a security requirement and a service model

**Rationale:** The description of some security property (e.g. separation od duty) requires knowledge about roles along their relations.

**Source:** ASSERT-O

**Requirement MO5.FUN**

**Description:** The ASSERT4SOA Ontology MUST support the representation of workflows. The ASSERT4SOA Ontology MUST support the answer to the following competency questions about workflows:

- What is the main activity of a workflow?
- Is an activity primitive or structured?
- Which are the activities composing a structured activity?
- What is the order of execution of sub-activities within a structured activity?
- What is the pre-condition for the execution of an activity?
- Which are the communication activities within a workflow?
- What is the remote service involved in a communication activity?
- What is the type of input/output messages exchanged by means of a communication activity?

**Event use case:** Check consistency between a security requirement and a service model

**Source:** ASSERT-O

**Supporting Material:** [44]

**Requirement MO6.FUN**

**Description:** The ASSERT4SOA Ontology MUST support the representation of constraints over resources. The ASSERT4SOA Ontology MUST support the representation of contexts of use of resources. The ASSERT4SOA Ontology MUST support the answer to the following competency questions about contexts:

- What is the resource used in a context?
- What is the business objective pursued in a context?
- What is the action performed by means of a resource in a context?
- What is the subject of an action in a context?

**Event use case:** Check consistency between a security requirement and a service model

**Source:** ASSERT-O

**Supporting Material:** [25]
ASSERT-E service model

**Requirement ME1.FUN**

**Description:** The ASSERT4SOA Ontology SHOULD support the representation of the Web service model used to produce the test cases for evidence-based certificates. The ASSERT4SOA Ontology SHOULD support the answer to the following competency questions about test-based service models:

- Which is the index of the test based service model?
- Which is the level of granularity of the service model? WSDL-based, WSCL-based or implementation-based?
- Which is the test generation model derived from the service model?

**Event use case:** Check consistency between a security requirement and a service model, compare requested with provided ASSERT

**Rationale:** ASSERT-E certification process is based on a model-based testing approach; service modeling allows to produce very detailed models and fine-grained test cases. In order to describe ASSERT-Es, the ASSERT4SOA Ontology should therefore consider aspects related to web service modelling.

**Source:** ASSERT-E

**Supporting material:** D4.1[2]

ASSERT-M system model

**Requirement MM1.FUN**

**Description:** The ASSERT4SOA Ontology SHOULD support the representation of system models included in an ASSERT-M (ASSERT-M service models). The ASSERT4SOA Ontology SHOULD support the answer to the following competency questions about ASSERT-M system models:

- What is the service modelled by an ASSERT-M system model?
- Which are the agents acting in a system?
- What is the set (alphabet) of actions that agents acting in a system can perform?
- Which are the possible behaviours of a system?
- Which are the correct behaviours of a system?

**Event use case:** Check consistency between a security requirement and a service model, compare requested with provided ASSERT

**Source:** ASSERT-M

**Supporting material:** D5.1[26]

**Requirement MM2.FUN**

**Description:** The ASSERT4SOA Ontology SHOULD support the representation of Agents. The ASSERT4SOA Ontology SHOULD support the answer to the following competency questions about Agents:

- Which are the actions performed by an agent?
• Which are the traces that an agent initially considers possible (agent’s knowledge)?

• Which are the parts of the system behaviour an agent can actually see (agent’s local view)?

• Which is the information that an agent has with respect to another agent’s initial knowledge?

• Which is the information that an agent has with respect to another agent’s local view?

**Event use case:** Check consistency between a security requirement and a service model, compare requested with provided ASSERT  
**Source:** ASSERT-M  
**Supporting material:** D5.1[26]

**Requirement MM3.FUN**  
**Description:** The ASSERT4SOA Ontology SHOULD support the representation of Behaviours. The ASSERT4SOA Ontology SHOULD support the answer to the following competency questions about Behaviours:

• What is the sequence of actions (traces) comprised by a behaviour?

**Event use case:** Check consistency between a security requirement and a service model, compare requested with provided ASSERT  
**Source:** ASSERT-M  
**Supporting material:** D5.1[26]

**Requirement MM4.FUN**  
**Description:** The ASSERT4SOA Ontology SHOULD support the representation of Actions. The ASSERT4SOA Ontology SHOULD support the answer to the following competency questions about System Actions:

• What is the name of an action?

• Which are the variables and sorts of an action?

• What is the agent performing an action?

**Event use case:** Check consistency between a security requirement and a service model, compare requested with provided ASSERT  
**Source:** ASSERT-M  
**Supporting material:** D5.1[26]

5.7.4 Requirements on Upper Ontology

**Requirement U1.FUN**  
**Description:** The ASSERT4SOA Ontology MUST support the representation of generic concepts related to certificates and ASSERTs. The ASSERT4SOA Ontology MUST support the answer to the following competency questions about ASSERTs:

• Which is the certification process followed to produce the ASSERT?
• Which is the Certification Authority who issued the ASSERT?

• Which is the validity status of the ASSERT: valid, revoked or expired?

• Which is the certified service?

• Which is the type of ASSERT: ASSERT-O, ASSERT-E or ASSERT-M?

**Event\use case:** ASSERT certified service lifecycle, issue of a security certificate

**Rationale:** In order to describe ASSERTs, the ASSERT4SOA Ontology should take into account general aspects related to certification.

### 5.8 Pre-Glossary of Terms

#### 5.8.1 Terms

This section describes the ASSERT4SOA terminology. Each term is defined in a table, where words highlighted in bold are related to terms specified elsewhere, i.e. in another table.

<table>
<thead>
<tr>
<th>ID</th>
<th>AbstractSecurityProperty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label</td>
<td>Abstract security property</td>
</tr>
<tr>
<td>Synonyms</td>
<td>Property type, category of security properties, security property</td>
</tr>
<tr>
<td>Description</td>
<td>A generic security characteristic for the service under evaluation, e.g. Confidentiality, Integrity, Robustness, and Authentication.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Abstraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label</td>
<td>Abstraction</td>
</tr>
<tr>
<td>Description</td>
<td>A smaller or simpler representation of a (concrete) model that was done in a systematic way to preserve valuable information. We use the term ”property preserving” in order to describe tests regarding the preservation of such properties</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>ASSERT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label</td>
<td>ASSERT</td>
</tr>
<tr>
<td>Description</td>
<td>The Advanced Security Service cERTificate (ASSERT) is a digital certificate containing a list of security statements about the security properties possessed by a service and the proof that the above statements are valid. The term ”ASSERT” implies the term ”certificate”. Therefore the term ”ASSERT certificate” is redundant. The ASSERT is signed by the ASSERT Authority.</td>
</tr>
<tr>
<td>ID</td>
<td>ASSERTCertificationProcess</td>
</tr>
<tr>
<td>------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>Synonyms</td>
<td>The process by which a certification authority issues an ASSERT.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>ASSERTCertifiedService</th>
<th>Label</th>
<th>ASSERT certified service</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>A service with security properties certified by an ASSERT.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>ASSERTLifecycle</th>
<th>Label</th>
<th>ASSERT lifecycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>The specifications of the distinct phases of the life of an ASSERT.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>ASSERTOntology</th>
<th>Label</th>
<th>ASSERT ontology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>The ontology supporting the description of ASSERTs (ASSERT-E, ASSERT-O and ASSERT-M) in the ASSERT4SOA Framework. It also describes a set of terms required to formalize an ASSERT-O.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>ASSERTProfile</th>
<th>Label</th>
<th>ASSERT profile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Mechanism to describe certification schemes (e.g. CC EAL4, coverISO9001, SAS70, Common criteria). The goal is to allow the homogenous representation of specific certification schemes.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>ASSERTType</th>
<th>Label</th>
<th>ASSERT type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>The type of ASSERT among one of the following: ASSERT-E (evidence-based certificate), ASSERT-M (model-based certificate) and ASSERT-O (ontology-based certificate).</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>ASSERTValidityStatus</th>
<th>Label</th>
<th>ASSERT validity status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>One of the following: Valid, Revoked and Expired (referring to the passed expiration date specified in the ASSERT).</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

D3.1 - Requirements for an ontology supporting certificates interoperability 54 / 69
<table>
<thead>
<tr>
<th>ID</th>
<th>ASSERT_E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label</td>
<td>ASSERT-E</td>
</tr>
<tr>
<td>Synonyms</td>
<td>Evidence-based certificate</td>
</tr>
<tr>
<td>Description</td>
<td>It is the result of the execution of the <strong>ASSERT-E Certification process</strong>. Each ASSERT-E is characterized by <strong>security properties</strong> and <strong>test-based evidence</strong>, and is signed by a third party proving that the service supports those properties.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>ASSERT_ECertificationProcess</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label</td>
<td>ASSERT-E certification process</td>
</tr>
<tr>
<td>Description</td>
<td>It is the process producing proofs based on <strong>test artifacts</strong> that a test carried out on a service has given a certain result, which in turn shows (perhaps with a certain level of uncertainty) that a given <strong>security property</strong> holds for that service.</td>
</tr>
</tbody>
</table>

**Note**: For a definition of test artifacts see **test-based evidences**.

<table>
<thead>
<tr>
<th>ID</th>
<th>ASSERT_M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label</td>
<td>ASSERT-M</td>
</tr>
<tr>
<td>Synonyms</td>
<td>Model-based certificate</td>
</tr>
<tr>
<td>Description</td>
<td>An <strong>ASSERT</strong> in which the assessment of the properties is based on the creation and analysis of a formal model.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>ASSERT_O</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label</td>
<td>ASSERT-O</td>
</tr>
<tr>
<td>Synonyms</td>
<td>Ontology-based certificate</td>
</tr>
<tr>
<td>Description</td>
<td>It is a <strong>certificate</strong> relating a service to its <strong>security properties</strong> using concepts defined within the <strong>ASSERT ontology</strong>.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Assumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label</td>
<td>Assumption/Dependency</td>
</tr>
<tr>
<td>Description</td>
<td>Runtime-checkable condition for the <strong>ASSERT</strong> to be considered valid.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Certificate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label</td>
<td>Certificate</td>
</tr>
<tr>
<td>Description</td>
<td>Digitally signed artefact, issued by a <strong>certification authority</strong>, conveying the statement about the properties owned by a software or service.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>CertificationAuthority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label</td>
<td>Certification authority</td>
</tr>
<tr>
<td>Synonyms</td>
<td>ASSERT issuer</td>
</tr>
<tr>
<td>Description</td>
<td>An entity trusted by one or more users to create and assign certificates.</td>
</tr>
</tbody>
</table>
**Certificate model**

The **web service model** included in the certificate. This model is the minimal model to base the specification of the web service’s **security properties** on.

**Attribute of a class of tests**

It refers to a set of threats the service proves to counteract, to a **security function** supporting the property, or to specific characteristics of the security function that is certified.

**Class of tests**

It represents a class of test (e.g., InputCoverage, Penetration) and contains a set of **test units**.

**Model-based testing**

It is the process of testing a service using a model including the formalization as a state transition system of interfaces, conditions on inputs and outputs, **conversation**, implementation details, and **security specifications**.
<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proof</td>
<td>Sufficient evidence or argument for the truth of a proposition. (See <a href="http://en.wikipedia.org/wiki/Proof">http://en.wikipedia.org/wiki/Proof</a>.)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SecurityControl</td>
<td>A management, operational or technical control (i.e. safeguard or countermeasure) prescribed for an information system to protect the confidentiality, integrity, and availability of the system and its information (See <a href="http://csrc.nist.gov/publications/fips/fips199/FIPS-PUB-199-final.pdf">http://csrc.nist.gov/publications/fips/fips199/FIPS-PUB-199-final.pdf</a>.)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SecurityFunction</td>
<td>A mechanism and/or piece of software in the service supporting a security property, satisfying a set of security requirements, and providing a given security functionality.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SecurityPattern</td>
<td>Describes a particular security problem that arises in specific context, and presents a well-proven generic solution for it. The solution consists of a set of interacting roles that can be arranged into multiple concrete design structures, as well as a process to create one particular such structure [50]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SecurityProperty</td>
<td>A security characteristic that a web service has, about which a security statement can be formulated. It is related to an abstract security property.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SecurityPropertyPreservingHomomorphism</td>
<td>A homomorphism that maps a concrete system model onto an abstract system model, the specific security property provided by the abstract system model is preserved, meaning that the respective security property holds in the concrete system as well. If the homomorphism satisfies specific conditions, it preserves the respective security property.</td>
</tr>
<tr>
<td>ID</td>
<td>SecuritySpecification</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>Label</td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td>It the modelling of the Web service security standards (e.g., WS-security) integrated and implemented by the service at container level.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>SecurityStatement</th>
<th>Security statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td>The claim that a given security property is possessed by a given service. A statement is made and signed by a certification authority and refers (is bound) to a specific service.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>ServiceDescription</th>
<th>Service description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td>A description of some facet of a software service that is stored in a service registry. The following types of description are considered: service behaviour description, service endpoint description, service interface description and service quality description.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>ServiceImplementationLevelTesting</th>
<th>Service implementation-level testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td>It is a model-based testing of the real service implementation.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>ServiceInterface</th>
<th>Service interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td>The set of operations supported by the services and a full specification of the signatures of these operations.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>ServiceInterfaceDescription</th>
<th>Service interface description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td>A service description giving information about the service interface.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>ServiceQualityDescription</th>
<th>Service quality description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td>A service description providing information about the quality properties of a service.</td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>SpecificationModel</td>
<td>Label</td>
</tr>
<tr>
<td>--------</td>
<td>--------------------------------------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td></td>
<td>Description</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>StereotypeAgents</th>
<th>Label</th>
<th>Stereotype agents</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Description</td>
<td></td>
<td>Additions within the certificate model that represent certain classes or categories of agents from the concrete system. They are important e.g., to express confidentiality properties such as ”data will not be disclosed to subcontractors”.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>TestAttribute</th>
<th>Label</th>
<th>Test attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Description</td>
<td></td>
<td>It refers to a characteristic of a given class of test.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>TestBasedEvidence</th>
<th>Label</th>
<th>Test based evidence</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Description</td>
<td></td>
<td>It is the complete set of artifacts describing the test-based model of the service, the test units, the test cases, detailed specification of testing practices (environment, runtime context) and outcome.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>TestBasedServiceModel</th>
<th>Label</th>
<th>Test based model of the service</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Description</td>
<td></td>
<td>It is the model of a service defined as a state transition system used in the model-based testing process.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>TestCase</th>
<th>Label</th>
<th>Test case</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Description</td>
<td></td>
<td>It is the real test executed on a service. It is composed by a pair (Input, Expected Output) and related code.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>TestType</th>
<th>Label</th>
<th>Test unit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Description</td>
<td></td>
<td>It is an instance of a class of tests (e.g., RandomInput is an instance of InputCoverage) and represents the type of any single test case.</td>
</tr>
</tbody>
</table>

With regard to the following terms, see their definitions in the "Web Services Glossary"\(^2\):

• Service
• Service provider
• Service requester

With regard to the following terms, see also their definitions in the "Standard glossary of terms used in Software Testing"\textsuperscript{3}:

• Expected output
• Test case
• Test unit

Chapter 6

Non functional requirements

6.1 Introduction

Non-functional ontology requirements refer to the characteristics, qualities, or general aspects not related to the ontology content that the ontology should satisfy. Examples of non-functional requirements are: (a) whether the terminology to be used in the ontology must be taken from standards, (b) whether the ontology must be multilingual, or (c) whether the ontology should be written following a specific naming convention. The non-functional requirements identified for the ASSERT4SOA Ontology are reported in the following.

Requirement MNT01.NF
Description: The ASSERT4SOA Ontology definitions SHOULD be objective. Where possible, a complete definition (a predicate defined by necessary and sufficient conditions) SHOULD be preferred over a partial definition (defined by only necessary or sufficient conditions).
Rationale: An ontology should effectively communicate the intended meaning of defined terms.
Source: [27]

Requirement MNT02.NF
Description: The ASSERT4SOA Ontology definitions SHOULD be documented with natural language.
Rationale: An ontology should effectively communicate the intended meaning of defined terms.
Source: [27]

Requirement MNT03.NF
Description: The ASSERT4SOA Ontology conceptualisation SHOULD be specified at the knowledge level without depending on a particular symbol-level encoding.
Rationale: An encoding bias results when a representation choices are made purely for the convenience of notation or implementation. The encoding bias of an ontology should be minimized, because knowledge-sharing agents may be implemented in different representation systems and styles of representation.
Source: [27]
**Requirement MNT04.NF**  
**Description:** The ASSERT4SOA ontology SHOULD support a versioning solution to keep track of changes in the ontology.  
**Rationale:** The concepts and relations in the ontology evolve over time. A versioning solution allows a relying party to keep track of changes in the ontology. Furthermore uncontrolled changes in the concepts can be used to attack the system.

**Requirement US01.NF**  
**Description:** Inferences derived by reasoning on the ASSERT4SOA Ontology SHOULD be consistent with the definitions given in the ontology. At the least, the defining axioms should be logically consistent.  
**Rationale:** An ontology should be coherent.  
**Source:** [27]

**Requirement US02.NF**  
**Description:** The ASSERT4SOA Ontology SHOULD be designed to anticipate the uses of the shared vocabulary, i.e. one should be able to define new terms for special uses based on the existing vocabulary, in a way that does not require the revision of the existing definitions.  
**Rationale:** An ontology should be extensible.  
**Source:** [27]

**Requirement LEG01.NF**  
**Description:** The ASSERT4SOA Ontology SHOULD specify its publisher. The signature of the ontology SHALL be verified using the public key of the publisher.  
**Rationale:** The trust on the publisher of the ASSERT4SOA Ontology may allow a client to select the most appropriate ontology available on the Net.

**Requirement PRF01.NF**  
**Description:** The ASSERT4SOA Ontology MUST support real-time reasoning.  
**Rationale:** Real-time reasoning is fundamental to support dynamic selection and integration of services in a SOA environment.

### 6.2 Requirements on ontology protection

In this section we summarize the security requirements for the ASSERT4SOA Ontology. We also present additional requirements for the ontology that are not security requirements, but that are related to security requirements for the ASSERT4SOA Framework.

As stated in D6.1[49], the main threat that the ASSERT4SOA framework faces is that malicious users can force the selection of a service that is not the best with regards to the needs of the client. The ASSERT4SOA Ontology is used in the reasoning process for comparing ASSERTs and for selecting services based on their respective ASSERTs, and is therefore a possible target for attacks implementing the above mentioned.

The main security requirements for the ASSERT ontology and the proposed security measures are summarised below. Both, requirements and proposed solutions...
will be revised later in the project, when the advances in the definition of the ASSERT4SOA ontology is detailed enough to ensure that the proposed solutions are (i) appropriate for the requirements; and (ii) implementable in practice, respecting different constraints (efficiency, client acceptance, complexity of management, etc.).

Requirement SEC01.NF
Description: The ASSERT4SOA Ontology SHOULD provide means to know the source of each element in the ontology.
Event\use case: Whole ASSERT4SOA lifecycle.
Rationale: The ASSERT4SOA Ontology is provided by different authoritative sources because concepts and relations in the ontology correspond to different scopes of authority.

Requirement SEC02.NF
Description: The core ASSERT4SOA Ontology SHOULD contain only the minimum elements relevant for the project. Additional components provided by independent contributors SHOULD not be required in order for the reasoning mechanisms to work.
Event\use case: Whole ASSERT4SOA lifecycle.
Rationale: As the ontology grows larger and more complex, the response times (due to accessing the ontology and due to reasoning about the ontology components) will also grow larger therefore the ontology could introduce delays or performance issues in the ASSERT4SOA Framework.

6.2.1 Requirements for the Ontology Manager

In this section the security requirements for the Ontology Manager, i.e. the software component responsible for interacting with the ASSERT4SOA ontology, are reported. The Ontology Manager is described in details in D6.1[49].

Requirement SEC03.NF
Description: The integrity of the ASSERT4SOA Ontology MUST be guaranteed.
Rationale: To guarantee the consistency and the robustness of the ASSERT4SOA lifecycle, unauthorized or malicious changes to the ontology should be forbidden.

Requirement SEC04.NF
Description: Ontology contributors SHOULD establish their identity and the ASSERT4SOA Ontology elements SHOULD be authenticated when read.
Rationale: The ASSERT4SOA Ontology content MUST be trustworthy and clients SHOULD have equal trust in all the contents of the ontology.

Requirement SEC05.NF
Description: The ASSERT4SOA Ontology SHOULD be replicated.
Rationale: The ASSERT4SOA Ontology SHOULD be available.

Requirement SEC06.NF
Description: (Ontology Manager) Responses to queries SHOULD contain either the ASSERT4SOA Ontology elements retrieved and information allowing clients to...
retrieve those elements.

**Rationale:** Clients could not completely trust the ASSERT4SOA Ontology therefore Ontology Manager responses SHOULD be verifiable in order to check whether the Ontology Manager results omit relevant information. An example for this is related to the unavailability of some externally provided ontology components. In such case, the results retrieved from the ontology should show this situation.

**Requirement SEC07.NF**

**Description:** Ontology Manager requests (or part of them) SHOULD be encrypted or sent using a confidential communication channel.

**Rationale:** Privacy of information included in the Ontology Manager requests SHOULD be ensured.

**Requirement SEC08.NF**

**Description:** Ontology Manager requests SHOULD be authenticated, either by being digitally signed by the client, or by being sent using an authenticated communication channel.

**Rationale:** Ontology Manager requests SHOULD be accepted only from authorised clients. this is particularly important when the ontology is used for intra-enterprise applications; or when there is a risk of flooding attacks.
Bibliography


