PoSecCo

Policy and Security Configuration Management

Project Title	Policy and Security Configuration Management
Project Acronym	PoSecCo
Project No	257109
Instrument	Integrated Project
Thematic Priority	Information and Communication Technologies
Start Date of Project	01.10.2010
Duration of Project	36 Months
Project Website	`http://www.posecco.eu`

D3.2 – SECURITY ONTOLOGY DEFINITION

Work Package	WP3, Decision Support for Policy Refinement
Lead Author (Org)	Cataldo Basile (POLITO)
Contributing Author(s) (Org)	Jacopo Silvestro (POLITO), Antonio Lioy (POLITO), Daniele Canavese (POLITO), Mario Arrigoni Neri (UNIBG), Stefano Paraboschi (UNIBG), Mario Verdicchio (UNIBG), Matteo Casalino (SAP), Theodoor Scholte (SAP)
Due Date	M12
Date	28/09/2011
Version	0.61

Dissemination level

X	PU: Public
	PP: Restricted to other programme participants (including the Commission)
	RE: Restricted to a group specified by the consortium (including the Commission)
	CO: Confidential, only for members of the consortium (including the Commission)

PoSecCo project (project no 257109) is partially supported/co-funded by the European Community/ European Union/EU under the Information and Communication Technologies (ICT) theme of the 7th Framework Programme for R&D (FP7). This document does not represent the opinion of the European Community, and the European Community is not responsible for any use that might be made of its content.

PoSecCo

Versioning and contribution history

Version	Date	Author	Notes
0.1	19.07.2011	Cataldo Basile (POLITO)	First draft
0.2	30.08.2011	Cataldo Basile (POLITO)	Second draft
0.3	01.09.2011	Stefano Paraboschi (UNIBG)	Revised paper structure
0.4	02.09.2011	Stefano Paraboschi (UNIBG)	Added Executive summary; several small updates
0.5	07.09.2011	Cataldo Basile (POLITO)	Finishing up; version for internal review
0.6	23.09.2011	Cataldo Basile (POLITO)	Suggestions in the internal review have been applied
0.61	28.09.2011	Cataldo Basile (POLITO)	Suggestions from the approval delegate have been applied

Disclaimer

CONTENTS

EXECUTIVE SUMMARY

Motivations for the use of ontologies

The management of security in modern information systems is characterized by a continuously increasing level of complexity. In most application scenarios, configurations are manually written by administrators. The integration of the different layers and components is limited and the security management is applied over a set of independent compartments. The overall goal of PoSecCo is to support the management of security policies, giving to administrators and auditors some models and tools able to realize more efficiently and effectively the representation and implementation of the great variety of security requirements characterizing modern information systems. An important contribution in achieving this goal is expected to come from the use of ontologies.

Security requirements for information systems have to be specified using a number of different levels of abstraction, each associated with a dedicated model. The importance and criticality of security requirements make necessary to guarantee the correctness of each of the models using the best available tools. If the system is not able to provide a high level of confidence in the consistency of the security policy model, then its usefulness is extremely reduced. In this scenario, the consistency has to be guaranteed both at an internal level, within each of the separate models, and at an integrated level, among the representations associated with the different models. Experience in the design and management of large data collections clearly shows the benefits that derive from functions supporting the verification of integrity constraints, but the PoSecCo scenario goes beyond the capabilities of traditional data models and needs a more advanced support, like the one offered by modern ontologies. Ontology languages and tools offer great expressive power and the possibility to define in a compact and well understood way a large variety of consistency conditions, giving a crucial contribution to the realization of a consistent model of the security requirements and their representation in the system configuration. Some of the performance limitations that slow the adoption of this approach in classical data management scenarios have a limited impact in this domain. Rather, it is well worth to pay the cost of the design and application of sophisticated consistency verification functions, given the improvement of the models expressiveness they are able to offer.

Ontology languages like OWL [1] have interesting features that fit well with the scenario: they are based on a well defined semantics that enables to map concepts from different perspectives (i.e., system architecture, security requirements, and user profiles) and they are enhanced with standard reasoning services, which can be exploited to support the security administrator. The use of ontologies and associated tools offers a concrete opportunity to support the security administrator in bridging the gap between high-level representation of requirements and its mapping to low-level configurations. Help is provided in tackling the following problems:

- a vertical matching between different levels of abstraction, with the aim to support the verification of the consistency of security requirements;
- a horizontal coordination between heterogeneous hardware and software security enforcement elements, with the goal of managing the implementation of security requirements independently from their specific features;
- a mapping of the relationships between components of the system that are typically managed in a separate way, like user profiles, access control policies, and network-level configurations. The achievement of a robust security configuration requires to configure them in an integrated way.

The use of ontologies offers the additional benefit that a detailed description of the obtained solution is produced, which can be the basis to support services for the explanation of the configuration. The evolution of the security policy is also greatly facilitated.

Goal of the document

It is important to note that the work on ontologies is planned to continue for the entire life of the project. This deliverable aims to present the results of the first year on this topic. The initial version of meta-models and corresponding ontologies will be presented in Work Documents and Deliverables that will be produced by M18. Near the end of the project further deliverables are planned to present the final version of the models produced by PoSecCo. It is then clear that this deliverable cannot describe the complete set of ontologies that PoSecCo is going to adopt. Rather, the goal is to describe the role and advantages that ontologies are going to provide, offering also a description of the architecture of the system and showing several implementation guidelines that will be followed in the development of the distinct ontologies. To support the presentation of the advantages of ontologies, the current status of the design of meta-models and ontologies that are planned for a later period is presented here, but this presentation must not be considered final and it is only provided to offer a better and more concrete understanding of the project advancement in this area.

The work illustrated in the deliverable shows that, using an ontology-based integrated security management, it is possible to define sophisticated functions for (semi-)automatic decision support, e.g., to vertically refine requirements between different abstraction levels. These functions can be built by composition of several standard reasoning services, in some cases resorting to task-oriented specialized ontologies (e.g., an ontology in which roles are modeled as classes to exploit the classification capabilities of the language).

The security requirements at a given abstraction level will be subject to a horizontal harmonization, performing the consistency check of the relevant portion of the ontology. Moreover, the enforceability analysis will be supported, that is, the verification that the security requirements at higher levels are actually met by low-level configurations both of access control and network-level security. An ontology-based approach proves to be very extensible. It is possible to define sub-ontologies and related tools, able perform ad hoc checks and reasoning, without losing the overall consistency that is maintained by the general ontology.

Outline

An introductory Section 1 defines ontologies and describe at a high level their main features, illustrating the main motivations that justify their use in PoSecCo.

Section 2 illustrates the architecture that is going to be used, characterized by a high degree of flexibility and the possibility to enrich basic mechanisms with ad hoc ontologies. The role of the Enrichment Modules, Assisted Modules, Landscape Configuration Modules is presented. The reasoning services in most cases will be supported by the use of standard reasoners, able to offer good performance and well known features. For some specific tasks, the architecture also envisions the use of custom reasoning modules.

Section 3 describes the approaches used for the management of ontologies and their integration. The ontology lifecycle is presented, with the consideration of how ontologies can evolve and react to changes in the landscape and in the requirements. The integration with ontologies originating from third parties is also discussed.

In Section 4 the current status of the meta-models and ontologies is presented. The goal is to provide a concrete demonstration of the support that ontologies can offer in the management of the PoSecCo requirements. The business level is not presented in this document because it is described in Deliverable D2.1. Also, the Business level model relies less on the use of advanced ontology features compared to the underlying levels. The description offered gives a good indication of the stucture of the solution that will be produced by the project. The description of the models allows to present a concrete application of the principles expressed in this deliverable as applied to the PoSecCo scenario. The application of ontologies in the models is particularly extensive when considering the Application level, which represent the specialization that extends the models in order to manage the concrete features of the landscape. Several examples are provided that show how ontologies and reasoning services can provide a significant help in the representation and verification of the correctness of the produced configuration.

Section 5 presents a technical analysis of the implementation of ontologies in PoSecCo. Current tools that

originate from the Semantic Web domain are analyzed and several indications are produced that will be followed in the implementation that will be produced in the next two years of the project.

Finally, Section 6 presents a few concluding remarks.

1 INTRODUCTION

1.1 Ontology definition

In philosophy, ontology is the study of the nature of being, existence, or reality in general and of its basic categories and their relationships. Traditionally listed as part of a major branch of philosophy known as metaphysics, ontology gives particular emphasis to questions regarding which entities exist or can be said to exist, and how such entities can be grouped and related within a hierarchy, subdivided according to similarities and differences.

Ever since the mid-1970s researchers in the field of artificial intelligence have recognized that capturing knowledge is the key to building large and powerful systems. In recent years the development of ontologies has been moving from artificial intelligence laboratories to the desktops of domain experts. Due to the Web development of the last years, ontological aspects of information have become strategic: the standardisation of information content is crucial for the development of the Web and essential to make communication processes easier. The ontologies on the Web range from large taxonomies categorizing Web sites (such as on Yahoo!) to categorizations of products for sale and their features (such as e-commerce sites). Ontologies try to remove, or at least reduce, conceptual and terminological confusion in order to have a shared interpretation.

In information technology a common definition of ontology is the following:

> An ontology is an explicit specification of a conceptualization.

A conceptualization is the abstract model of a phenomenon, that is a simplified view of the world that 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 relevant 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.

An ontology defines a set of representational primitives used to model a domain of knowledge or discourse. The representational primitives are typically:

- Instances or individuals. These may include concrete objects such as people and cars, as well as abstract objects such as words and numbers.
- Classes or concepts. These represent the main categories according to which the world is organized as they are used to describe a specific application domain (domain ontology). They are abstract groups, sets, collections of individuals.
- Attributes. An attribute of an individual can relate the individual to aspects or parts. Typically, these aspects or parts are represented by classes or instances.
- Roles or relationships. They describe the way classes and individuals can be related to each other. Relationships are typically of a particular type. Important types of relationships are the subsumption relationship (is-a-superclass-of, the converse of is-a, is-a-subtype-of or is-a-subclass-of) and the mereology relationship, written as part-of, that represents how individuals combine together to form composite individuals. Ontologies have the ability to describe relationships. As a result, ontologies often include additional relationship types such as ‘is defined as a successor of’.

1.2 Ontology Languages

Ontologies are constructed using formal languages. These languages allow the encoding of knowledge of specific domains. Languages can also allow to include reasoning rules that support the processing of that knowledge. Ontology languages are typically declarative languages, meaning that the ontology is specified without providing an algorithm that describes how the ontology should be constructed or how the reasoning should happen. Examples of ontology languages are Common logic [2], Cyc [3], Gellish [4], IDEF5 [5], KIF [6], RIF [7] and OWL [8].

Developed by the W3C Web Ontology Working Group, the Web Ontology Language (OWL) is aimed to be the standardised and broadly accepted ontology language. It has attracted academic, medical and commercial interests. OWL is built on top of the Resource Description Framework (RDF) [9] and RDF Schema [10]. RDF provides a way to express statements about resources in the form of subject-predicate-object expressions (triples). RDF Schema is an extension to RDF: it provides mechanisms to describe RDF resources and the relationships between them in terms of classes, properties and values. RDF schema (RDFS) descriptions are written in RDF. OWL is a successor of RDF Schema; it is a stronger language and has a greater machine interpretability. It allows one to specify:

- logical expressions (and, or, not)
- (in)equalities
- local properties
- required/optional properties
- required values
- enumerated classes
- symmetry, inverse of relationships

OWL has three increasingly-expressive sublanguages: OWL-Lite, OWL-DL and OWL Full.

- OWL-Lite can be used to express taxanomies and simple constraints. An example of simple constraint is the cardinality constraint where only the values 0 and 1 are permitted.
- OWL-DL combines maximum expressiveness with computational completeness and decidability. It is based on Description Logics and therefore it is possible to automatically compute the classification hierarchy and check for inconsistencies.
- OWL-Full supports full expressiveness and syntactic freedom. This comes at the cost of having no computational guarantees. Therefore, reasoning is less predictable and it is not possible to perform a completely automated reasoning on OWL-Full ontologies.

Each of the sublanguages is a more complex version of its predecessor. OWL-Full can be considered as a full extension to RDF, while OWL-Lite and OWL-DL are extensions to subsets of RDF.

OWL 2 is the successor of the Web Ontology Language (OWL 1). Backwards compatibility of OWL 2 with OWL 1 is ensured: all OWL 1 ontologies are valid OWL 2 ontologies. OWL 2 extends OWL 1 with a small but useful set of features:

- keys
- property chains
- richer datatypes, data ranges
- qualified cardinality restrictions

- asymmetric, reflexive, and disjoint properties
- enhanced annotation capabilities

OWL 2 profiles are sublanguages of OWL 2 and provide important advantages in certain application scenarios. The different profiles are defined as: OWL 2 EL, OWL 2 QL, and OWL 2 RL. Each of them is a syntactic restriction of the OWL 2 structural specification. OWL 2 EL supports polynomial time algorithms for standard reasoning tasks, this language is particularly useful in situations with very large ontologies and where standard reasoning is sufficient. OWL 2 QL enables conjunctive queries to be answered in LogSpace using standard relational database technology. OWL 2 RL enables the implementation of polynomial time reasoning algorithms using rule-extended database technologies operating directly on RDF triples. The latter one is useful in environments with large number of individuals and where it is necessary to operate directly on RDF triples.

Considerable progress has been achieved in the development of tool support for OWL 2. The new syntax is currently supported by the new version of the OWL API [11]. The widely used Protégé system [12] has recently been extended with support for the additional constructs provided by OWL 2.

1.3 Tool Support

There is a large number of tools available to work with ontologies. The most important classes are editors, reasoners and management tools. We provide here a concise evaluation of the main representatives of each class. In Section 5 we will provide a specific analysis of the tools that are best suited to manage the PoSecCo requirements and that we expect will be the basis for the development that is going to occur in the next period.

1.3.1 Editors

Ontology Editors are designed to support the creation and manipulation of ontologies. In general, editors are independent from the actual ontology representation language and provide visual navigation capabilities. In order to persistently store ontologies, editors support the import and export of different ontology representation languages. Some editors have built-in support for querying ontologies using SPARQL[1]. Moreover, some editors include support for additional features such as reasoning. Many different editors are nowadays available. Each editor has its own purpose and application. Well-known examples include: Protégé [12], Hozo [13], KAON [14] and the NeOn Toolkit [15].

1.3.2 Reasoners

Semantic reasoners are used to infer logical consequences from a set of facts or axioms. These facts or axioms can be specified by means of an ontology language and a description language (OWL-DL as an example). Many reasoners are available that support reasoning over OWL-Lite or OWL-DL. These include: Pellet [16], Jena [17], KAON2 [18]. Reasoners can be integrated into applications using a Java API (Jena, OWL-API [11]) or DIG [19]. The DIG standard defines an interface which enables the reasoner to be accessed over HTTP using the DIG language.

1.3.3 Management Tools

Tools also exist that support the management of multiple or distributed ontologies. In environments with complex requirements, it is essential to have systems supporting the storage and retrieval of ontologies. A framework such as Jena or SOFA [20] has the ability to serialize ontologies to an RDBMS. However, Jena and SOFA can only operate on one ontology at the same time. Unlike Jena, systems such as Minerva [21] and SOR [22] support storage, reasoning and search over multiple ontologies.

[1] http://www.w3.org/TR/rdf-sparql-query/

Apart from storage, it is sometimes necessary to perform integration operations on multiple ontologies. An example of such operation is the *merge* operation which creates a new ontology by linking up the existing ones. The tool Chimaera[2] supports users with merging multiple ontologies together and diagnosing individual or multiple ontologies.

1.4 Application of ontologies in PoSecCo

As described in the PoSecCo Deliverable 3.1, PoSecCo features a top-down approach for creating a consistent set of configurations. The first step is the representation of business policies, which are created to satisfy application requirements and existing regulations. These business policies are mapped down to IT policies, which consider the elements of a landscape model. Then, the IT policies are further refined into a set of configuration settings that are consistent and conflict-free. The Security Decision Support System (SDSS) coordinates all the refinement and conflict analysis phases of the top-down approach. In this process, there are several important intermediate steps:

- generation of logical associations, which represent in an abstract and flexible way a variety of concrete security configurations (like the need to support the interaction between two network parties, the right for one user to access a resource, or the need to protect a system from outside connections);
- conflict analysis;
- generation of configurations;
- optimization of configurations, including remediation of conflicts and non-enforceability.

Each of the steps in the refinement process requires knowledge about the system landscape, its elements and their relationships. Thus, a mechanism is required to capture and discover the semantics of the system landscape. In order to understand this information, a description of each element of the system landscape must be expressed using the same language and semantics, and thus a PoSecCo ontology must be defined. As the refinement process covers multiple layers of the system landscape, it is essential that the ontology captures the elements and relationships within the different layers as well as the relationships between these layers.

Ontologies can be queried, adapted, and enriched. These abilities are particularly useful in the context of PoSecCo, as each step in the refinement process requires more detailed information about the system landscape; it should be possible to enrich the landscape model with more information. Moreover, the refinement and deployment of configurations results into changes of the landscape. The landschape model should be adapted accordingly.

A major advantage of using ontologies in the context of PoSecCo is their support of automated reasoning. This allows one to infer knowledge from the model. For example, it is possible to infer that if a reverse proxy server is available to “protect/assist” a webserver, then the reverse proxy server should be configured to handle requests for the webserver.

Last but not least, the development of the Semantic Web in recent years resulted in a number of well-established XML-based knowledge representation languages and standards, including RDF, RDF Schema and OWL. This also resulted in the wide availability of a large number of tools and libraries that allows one to create, read and update ontologies and to reason over them. Because of the availability of existing tools and libraries, PoSecCo does not have to implement these functions. Instead, these tools will be reused throughout the PoSecCo project.

[2] `http://www.ksl.stanford.edu/software/chimaera/`

1.4.1 Limitations

One of the limitations of ontologies is that maximum expressivity comes at the cost of decidability and completeness. The use of OWL-Full implies that the reasoning is not always decidable nor complete. Also, the performance of reasoning is not guaranteed.

Another limitation of ontologies is that they have an inherent bias derived from the domains, cultures and environments in which the concepts exist. An ontology is in essence a categorization. Categorizations, like the one used by librarians, are always subjective.

In PoSecCo, we manage the risks discussed above by requiring that ontologies are represented in OWL-Lite or OWL-DL, as we will discuss later. Strict guidelines will be enforced, to reduce biases to a minimum.

1.4.2 Use of multiple ontologies

Ontology merging is the process of bringing two conceptually different ontologies together. Several tools exist to execute this process (semi-)automatically. In PoSecCo, we plan to use different ontologies for the different layers (business layer, IT layer, infrastructure layer) for the sake of ease of management. Each layer and corresponding ontology will be managed separately. Because the different ontologies will share a number of concepts, automated merging of the different ontologies is possible. In this way, an overall PoSecCo ontology can be generated from several different individual ontologies. This aspect is analyzed in Section 3.

1.5 Requirements for the PoSecCo ontology

The role of the ontology is to represent in a rich way the domain of knowledge, which contains the landscape, i.e., the concrete infrastructure used to realize the application. Ontologies help filling the gap between the IT Policies and the Logical Associations. As briefly discussed in Section 1.4, and explained in Deliverable D3.1, the Ontology Refinement System starts from the representation of the IT Policy and produces a set of Logical Associations that will lead to generation of the security configuration of the system.

This means that different concepts must be included in the ontology:

- IT Policy concepts;
- Landscape concepts;
- other concepts needed to connect the IT Policies to the Logical Associations and Landscape concepts.

The connection between IT Policy concepts and Logical Association concepts is needed to perform the refinement. In fact, the Logical Associations require the low level information that is associated with the individuals used at the IT Policy level. Several of the IT Policy concepts will be specializations of landscape concepts, denoting their security features. For instance, while a service is a landscape concept, its classification into public or private services (or at different security levels) may be a typical IT Policy concept.

The relationships between these concepts are presented in Figure 1:

- some concepts can be used for the specification of both IT Policies and Logical Associations;
- IT Policies can be specified using also concepts that are not related to the landscape;
- Logical Association concepts are a subset of the landscape concepts, since all the Logical Associations will be eventually refined to configurations.

The security ontology can contain some information about the outsourced services (e.g., their IP addresses) that must be managed using PoSecCo. For instance, to configure a secure channel between an internal

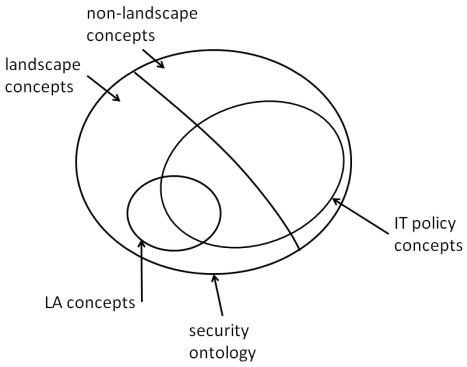

Figure 1: Relationship between the ontology concepts.

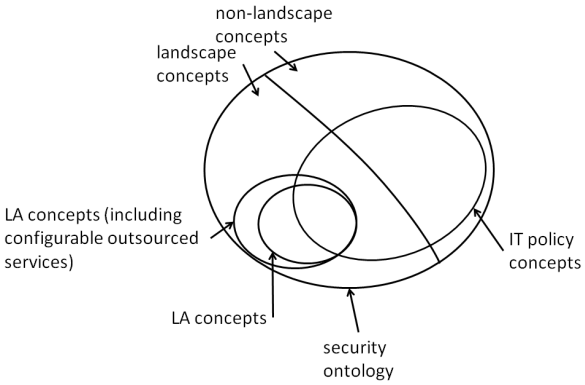

Figure 2: Relationship between the ontology concepts, including the concepts related to outsourced services.

service and an outsourced service, it is necessary to configure the local gateway and all the internal firewalls in the paths between the two services.

In many cases it is useful to explicitly define and name a group of individuals (e.g., if they are subjects or objects of some IT Policy), in order to assign to it some properties. For instance, it could be convenient to define groups of users (e.g., administrators, guests) and assign to them different privileges, or to define groups of network nodes (e.g., subnets, domains) and assign to them different security levels. In general, the flexibility offered by modern ontology languages and tools permits the realization of rich representations in the different domains.

1.6 Overall organization of PoSecCo ontologies

Starting from the considerations presented above, we decided to structure the PoSecCo ontology in a modular way, as an aggregation of several ontologies, each covering a different aspect. The description of all these ontologies appears in Section 4. We discuss here the relationship between these ontologies and the system and policy models, which is depicted in Figure 3. The business layer does not use the advanced features of ontologies and it has been omitted. The ontology based refinement starts from the IT layer. For the IT layer we have the IT ontology, which maps the classes of the System Functional meta-model, and the IT Policy ontology, which maps the classes of the policy meta-model and contains the Access Control Ontology. Since many of the policy related concepts are also system related, as a policy refers to system components, these ontologies have some overlap. At the landscape level the system model classes are mapped to three different ontologies, depending on the type of concept the ontology contains: Virtual, Physical and Application which represents the classes of the “virtualization model”, “network level infrastructure model” and “distributed

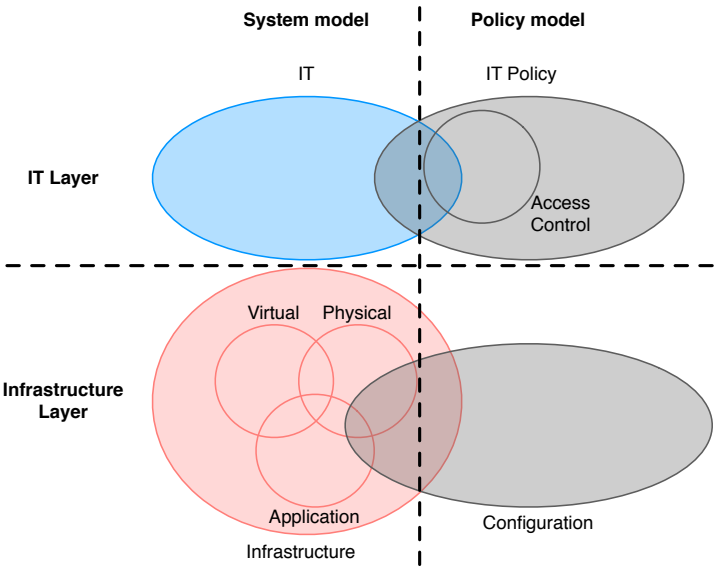

Figure 3: Relationship among the PoSecCo ontologies and the meta-models.

application model", described in the deliverable D4.2. Also in this case the three ontologies share some common concepts, as the three different domains are strictly related to each other (e.g., a virtual machine runs a software and is run on a physical host). Moreover, the classes of the configuration model are mapped to the configuration ontology, which, in turn, will contain some landscape related concepts (e.g., the devices to which the configurations apply).

In Section 4 we succinctly present the current status of the ontologies that have been designed at the IT and Infrastructure layers. The goal is to provide a concrete demonstration of the support ontologies can offer in the management of the PoSecCo requirements.

2 ONTOLOGY-BASED REFINEMENT

2.1 Architecture

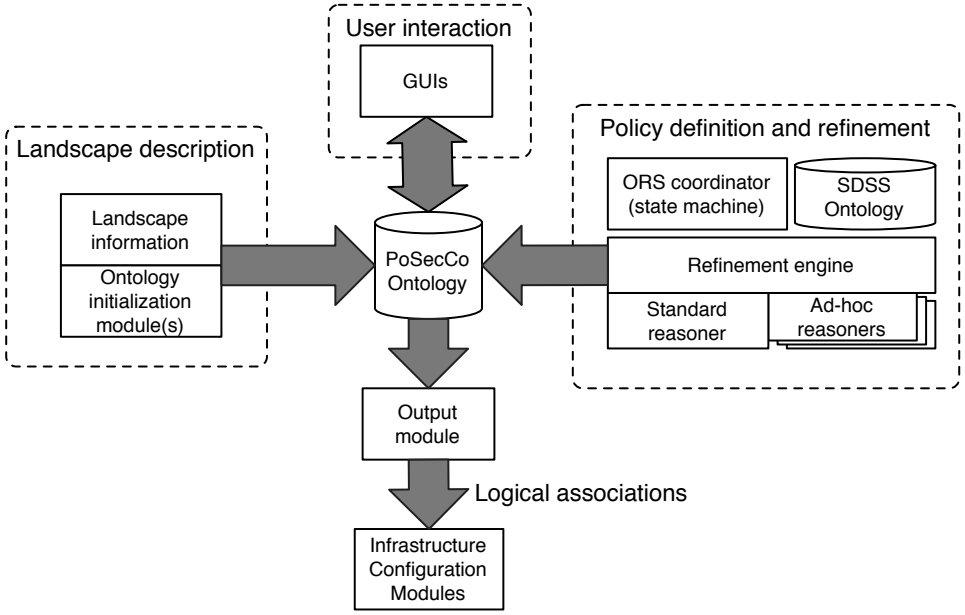

Figure 4: General schema of the ORS architecture.

The Ontology Refinement System (ORS) depicted in Figure 4 is designed around the PoSecCo Ontology (PO), which works as a knowledge base and through which all the modules interact with each other. The modules can be divided into different parts depending on their role. The landscape description modules read the landscape description from the model repository, described in the deliverable D2.1, and store the resulting information in the PoSecCo Ontology as individuals. The user interaction GUIs interact with the user to start a new project or restore an existing one. They are also used to define the IT policy and during the refinement process, when a system administrator is asked to provide information.

The policy definition and refinement modules are the core of the entire system. They include:

- The ORS coordinator, which drives the policy definition and refinement phases;
- The SDSS Ontology, which stores all the information about the available reasoning and refinement modules and their relationships. It includes: the SDSS components' list (GUIs, controllers, etc.), Enrichment Modules (EMs), Assisted Modules (AMs), and Landscape Configuration Modules (LCMs). It also stores the dependencies among the available modules (AMs or EMs), by adding properties to the PoSecCo Ontology individuals that represent them.
- The refinement engine, which runs the standard and ad-hoc reasoners to perform the refinement from IT policy to Logical Associations.

After policy definition and refinement the output module is executed, which extracts the results of the refinement from the PO as a set of Logical Associations. A Landscape Configuration Module (LCM) takes as input a set of Logical Associations and maps them to a set of configurations for the landscape devices. A complete and detailed description of LCM modules will be provided in the deliverable D3.6, that will be available at M24.

In this architecture, the PoSecCo Ontology is used to store all the knowledge about the landscape. When initialized by the *ontology initialization module*, it contains the description of all the classes and properties

(Tbox), but no instances. Then, the *Ontology initialization module(s)* loads the initial description of the landscape. Successively, the PoSecCo Ontology will store all the information generated during the entire refinement process by the standard and ad-hoc reasoners, that is the EMs and AMs.

The section 2.3 will describe in detail the methods used to perform the actual refinement using the ontology-based reasoning.

2.2 Workflow for policy definition and refinement

Before starting the policy definition and refinement, which is depicted in Figure 5, we have the general initialization. A setting GUI is shown and the user, typically a system administrator, can start a new refinement process by defining its name and some preferences. The user also provides the landscape description and/or an already populated ontology together with the list of the available LCM. The user can also resume a previously saved project and complete its execution. In this case, the next step depends on the previously saved state.

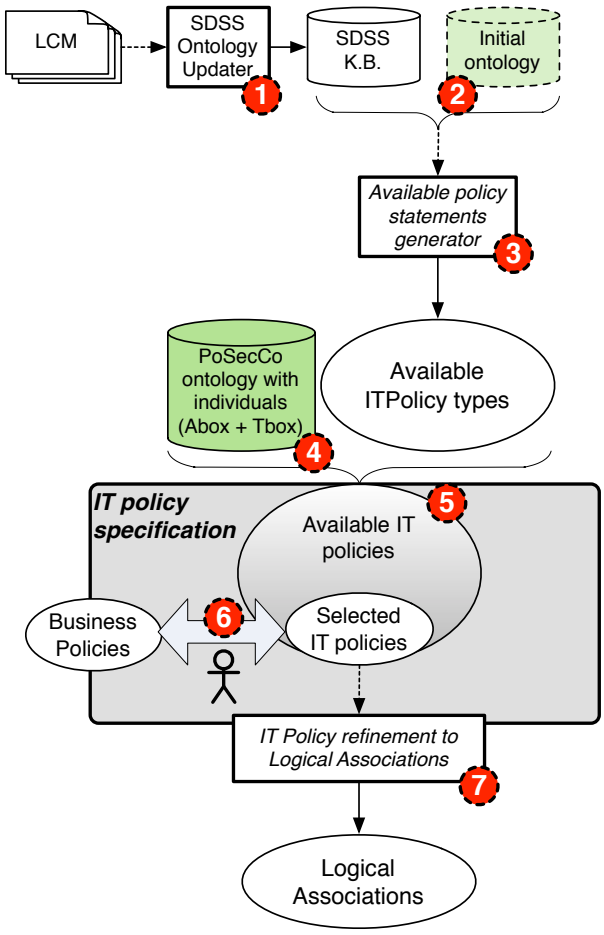

Figure 5: Policy definition and its refinement to Logical Associations.

2.2.1 Policy environment initialization

We want to design the policy definition process in such a way that the refinability of each policy is implied by the process itself, i.e., it is not possible to define a non-refinable policy. We started from the following considerations:

- Each LCM handles a triple (subject,property,object), where subject and object are PoSecCo ontology concepts, while verb is a security property, e.g., (IPAddress, Reach, IPAddress);
- At the beginning, the PoSecCo Ontology contains only the hierarchy of concepts (no instances) and the object property definitions (Tbox), which represent logical relationships among different concepts. Considering both the concept hierarchy and the object properties, it is possible to define the Ontology Graph, having as vertices the PO concepts and as edges ontology relationships.

Starting from the LCM input triples and the class relationships described in the initial ontology, all the possible (S, V, O) triples are computed. Each ontology concept C_i can be used as subject or object for a given verb V of a triple (S, V, O) if it is possible to find a path between C_i and S or O in the Ontology Graph. Given all the subjects and objects, we can generate all the possible (S, V, O) combinations via a cartesian product operation. Each possible (S, V, O) will be a valid (available) IT policy type.

Using such a method, we guarantee that an IT Policy can be understood by the tool and refined to a triple that can be managed by at least one LCM by traversing the Ontology Graph.

In order to design our system, we started from the description of the workflow to delineate the structure of the model and all the components.

1 – SDSS knowledge base initialization. At this point the SDSS knowledge base for the structure of the tool itself, is populated with all the available modules (LCM, EM and AM) and their properties, according to the information acquired from the configuration provided in the previous step.

2 – PoSecCo Ontology initialization. The PoSecCo Ontology is initialized as empty (all the classes are described, but there is no instance). We can also provide an already populated one that can be the result of a previous execution.

3 – Available IT policy type generation. The Ontology Graph is created by querying the PO, then considering the LCM described in the SDSS Ontology and the Ontology Graph, the available type of IT policy, expressed as (S, V, O) triples, is generated.

4 – Landscape description import. If requested, the landscape description is imported and all the information is added to the PoSecCo Ontology, adding class instances and setting their properties.

5 – IT security policy generation. In this stage, the available IT policies are composed, considering the available IT policy type and the PoSecCo Ontology classes and individuals.

6 – BP to ITP mapping. The user is asked to select the IT policies that map the input Business Policies via an ad-hoc GUI.

7a – ITP groups explosion. The IT Policies must be refined to LAs before being processed by the LCM to generate the configurations. We introduce here the concept of “aggregation” that will be used in the following description. An ontology individual is an aggregation if and only if the “hasMember” relationship has been set in the ontology. This relationship connects an aggregation individual to the members it represents, which are other individuals, having in general $1 \rightarrow M$ cardinality. Aggregations need a different treatment when performing the refinement, since each individual represents a collection of individuals.

The IT Policy refines the ontology concepts and the aggregation-individuals (e.g., roles) are exploded to the corresponding individuals (network nodes, users), thereby a set of (S, V, O) triples are generated as the combination of all the subjects and objects of each IT policy (keeping the same verb).

7b – Module execution. At this point the actual refinement process starts and the modules are executed. If any module requires interaction from the administrator, a custom GUI is shown. This module execution step represents an exception, because it is not an atomic execution unit. In fact, since it may require the execution of a number of EMs (see Section 2.3.3) and AMs (see Section 2.3.4), it may be divided into many steps. It also supports logging and storage of user input data to allow its re-usage in other branches or projects.

2.3 Reasoning methods

2.3.1 Standard reasoner

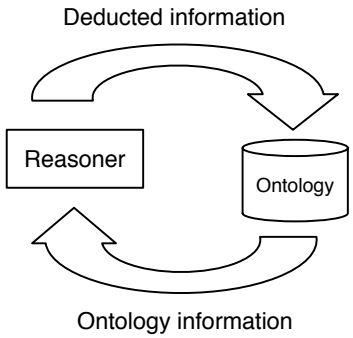

Figure 6: Reasoner execution scheme.

The standard reasoner is one of the core elements of an ontology based system. Figure 6 shows how it works. Starting from the information contained in the ontology, it is able to perform several tasks. In general, it is able to check the consistency and validity of the ontology, classify its information, answer queries, and generate inferences, using a variety of techniques deriving from the work of the Artificial Intelligence community and enriching existing information.

In particular, standard *DL* reasoning performed w.r.t. a formal ontology can check complex consistency constraints in the model. Such constraints are different from the usual ones from database and *UML*-like systems. For example, using *DL*-based languages, like *OWL* (*Web Ontology Language*) and *DAML-OIL*[3], we can express:

- constraints on properties, domains, and ranges;
- definitions of concepts (classes) in terms of relationships with other elements;
- boolean operations on classes; in the simplest case, they allow for the representation of complete, total, and partial refinements.

One of the main differences between *DL* based schema definitions and *UML* or *ER* ones concerns the constraints on property domains and ranges. Properties can be defined in a general way and their behavior in terms of range type can be precisely described while refining the ontology concepts. This promotes the definition and reuse of high level properties, without losing the ability to force precise typing. For example an "*abstraction-of*" property can be used to support the traceability of the refinement process, by connecting one element with another one that represents the same node at a lower level. Generally speaking, such a property will be able to connect almost any element in the model. However, some homogeneity must be guaranteed on the types of elements connected by an *abstraction-of* link. In this case we can describe at a lower level axioms that, for example, IT level machines must be refined only in some types of resource (physical servers and virtual machines). Such a description goes beyond classical property domains and range definitions.

Object refinability is typed by specific axioms and can be verified by classic *DL* reasoning. For example, Figure 7 shows the case of a system *system_1* declared to be refined in an action (*action_1*)[4]. The property *abstraction-of* represents this relationship, but *T-box* level axioms from the same figure state that:

[3] `http://www.daml.org/2001/03/daml+oil-index`

[4] The notation used is a graphical notation for the constructs offered by OWL. In the simple examples provided in this deliverable, the notation should be self explaining.

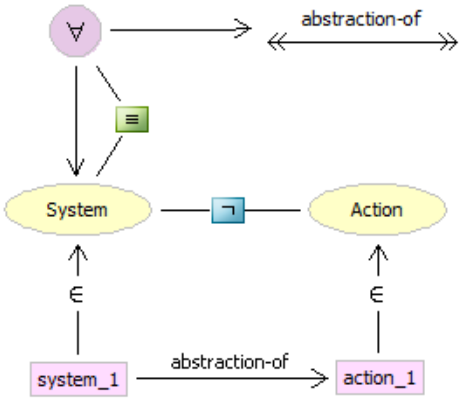

Figure 7: Type constraints on refinement through *DL* axioms.

- *System* only refines in lower level systems;
- *System* and *Action* are disjoint; that is, *action1* cannot be an *Action* and a *System* at the same time.

In this case, *Description Logics* reasoning rapidly finds an inconsistency about the individuals in the model.

2.3.2 Ad-hoc reasoning methods

Standard *Description Logics* reasoners can answer complex questions and can verify structural and non structural constraints. Furthermore, *Description Logics* based language expressiveness often exceeds classical solutions (like UML for design and SQL for data storage models). This allows the description and verification of more complex structural constraints.

However, *DL* systems, as well as *Semantic Web* tools in general, are designed and implemented with a focus on knowledge management services, such as knowledge integration, schema matching and instance retrieval. Such a specialization poses some limitations on the use of pure *Description Logics* reasoning in scenarios like the one of PoSecCo, in which reasoning must be carried out on a well defined and complete description of a closed system.

In particular, some characteristics of a typical knowledge management oriented description that can be critical and must be carefully considered during the adoption in PoSecCo are:

***Closed World Assumption* (*CWA*) reasoning** is a generally accepted requirement in model driven systems. Conversely, *DL* reasoners usually work under the *Open World Assumption* (*OWA*). This means that the facts asserted in the model (e.g., about the layout topology or the authorization policies) are not assumed to be complete. Obviously, this can become a problem if model characteristics are described in terms of the existence of some properties or some relationships between model elements.

Reasoning on complex property paths (e.g., commutativity of non trivial graphs) rapidly brings to the undecidability of the formal logics the language is based on. Checking the closure of complex paths is beyond the expressive power of classical database systems too, but unfortunately it is sometimes necessary to check structural constraints. This is the case, for example, for the consistency loop in Figure 8 stating that

> a *privilege* of executing an *action* must be assigned to a *resource* that runs on a *system* compatible with the *action type*.

***Unique Name Assumption* (*UNA*)** is a commonly accepted assumption in most model driven tools. It consists in assuming that different names will always denote different elements in the model. This

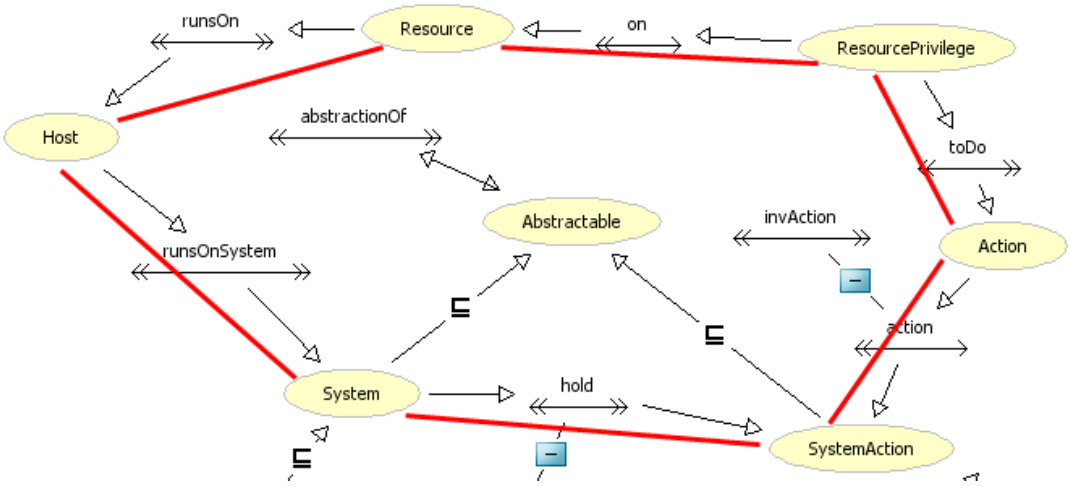

Figure 8: A simple consistency loop.

is usually not true in *Description Logics* reasoners, because of the essential nature of knowledge integration problems. In fact, in the *Semantic Web* scenario, different authors may describe the same entities (both shared conceptualizations and physical objects) assigning a new name, generally in the form of an *URI*, defined independently from other *Semantic Web* users.

In order to solve these issues, classical *DL* reasoning must be integrated and completed by adding a set of specific reasoning modules.

Application-level closure

The first and probably simplest point to address is the necessity to use both *Open World Assumption* and *Closed World Assumption* reasoning in the same system. In general, and in the PoSecCo system in particular, reasoning can be used for at least two tasks:

Schema reasoning operates on the *ontological* level of the semantic model. This level comprises the schema (T-box in *Description Logics* terminology) and optionally a very limited number of fundamental instances. The reasoning at this level boils down mostly to consistency checks and concept classification. In this scenario, the *Open World Assumption* is very well suited to drive consequences (theorems) that are correct for any possible model of the schema.

Instance level reasoning involves both schema axioms (as background knowledge) and the individuals used to express actual models at the extensional level. Constraints checking is a typical reasoning task at this level, as well as semantic model completion through the derivation of implicit properties between specific instances. At this level a more database-styled semantics can be used, which interprets the instance level assertions with a closed world semantics.

In order to integrate the two semantics, some metadata must be added at the ontology level to describe which properties must be interpreted under *CWA* at the extensional level. In other words, a set C of properties (or *roles* in *Description Logics* terminology) must be kept, which will be “closed” before any reasoning can be performed. This means that the compatible models that are extensions of the explicit model represented by the user cannot contain links of type R that are not explicitly included by the user and that cannot be derived (proved to exist) starting from the original model.

For each property R in C, $Links_R$ is the set of all the predicates of type R (that is, all the *RDF* triples having R as predicate). $Links_R$ must be extracted. After that:

1. an axiom is added, stating that the individuals that are not listed as a source in any predicate in $Links_R$ must have no R links to any individual;
2. a new axiom is added for each individual i that appears as a source in $Links_R$. The axiom states that i can be connected through R to an individual j if and only if j is an R-successor of i in $Links_R$.

A similar solution can be adopted to obtain the *Unique Name Assumption* semantics. The *Unique Name Assumption* can be considered a sort of *Closed World Assumption* applied to the "*same-as*" relationship; in fact, no individual can be considered as representing the same object if it is not explicitly stated in the model or it can be proved starting from the model itself. However, if we assume that no individual equivalence can be derived from the model, an even simpler solution can be implemented by stating explicitly a "*different-from*" property between all the possible couples of individuals.

Rule based inferencing

Rule inference reasoning is widely used in knowledge management systems. Recently some combinations of theorem proving systems (like *Description Logics* ones) and rule inference systems have been proposed to address some limitations of decidable theorem proving systems.

Semantic Web Rule Language (SWRL) is the *W3C* standard proposal for integrating rule-based inferencing into systems that represent knowledge as a set of *RDF* triples and introducing some limitations to the use of the rules, in order to preserve joint system decidability.

In the PoSecCo scenario, the main advantage of combining rules and classical theorem proving systems is the support for complex property chains. In fact, even if some *OWL2* profiles introduce the support for property (a.k.a *role* in *DL* terminology) chaining, this can be not sufficient to express some complex topological properties. For example, the simple consistency loop shown in Figure 8 involves six different properties and requires that a loop must exist for each *toDo* link between a *ResourcePrivilege* and an *Action*. The existence of such a loop is not enforceable at the schema level using only *Description Logics* axioms. Furthermore, as a general outcome of the adoption of the *Open World Assumption*, even if we could enforce the existence of the loops, we would not be able to require that such loops are explicitly stated into the assertional part of the semantic model (*A-box*).

At the opposite end, due to decidability issues (*DL* safe rules) the rule based component of the language operates in a sort of *Closed World Assumption (CWA)* limited to the nodes. This means that a forward chaining rule can triggered by any property derived by the reasoning, but involving only nodes that are explicitly named in the *A-box*. Then, we can operate only on nodes and properties explicitly stated in the semantic model.

Furthermore, rules can freely combine as antecedents triple patterns to capture complex topological structures, and this solves the lack of complex property chains of *Description Logics*. This means that we can check for loops, or for the absence of loops, by adding custom rules to the PoSecCo ontology.

However, *SWRL* safe rules can consume only positive knowledge, so they can be used to directly detect errors that consist in the existence of some structure in the ontology, that is the existence of a loop. However, if the property that we want to detect involves the non existence of some links, an application-level closure is required. So we have to distinguish between two kinds of rule:

Error detection rules apply when a misconfiguration or an error in the model is detectable by checking positive knowledge. In this case a *SWRL* rule can directly check such a configuration and mark a node as "misconfigured", both by classifying it as an instance of a technical concept (*Misconfigured*) or, even more directly, by making it and the whole model unsound by adding it to a non satisfiable concept (like *owl:Nothing*). In this case the adoption of rule based inferencing can perform the error detection task without any impact at the *DL* and application level.

Property checking rules apply if the error condition consists in the lack of some links. For example, the link shown in Figure 8 represents a mandatory situation; if the loop cannot be closed, then a

misconfiguration must be revealed. Since inference rules cannot be triggered by the non existence of a link, the rule will detect the correct configuration and will mark the node or the link as "correct" (in the example from Figure 8, the *toDo* link between two individuals will be marked). After role execution terminates, a custom application level reasoner will check that all the suitable links and nodes have been marked and verified, if any element is not checked, then a misconfiguration error will be detected.

As an example, let us consider a simplified version of the loop as shown in Figure 9.

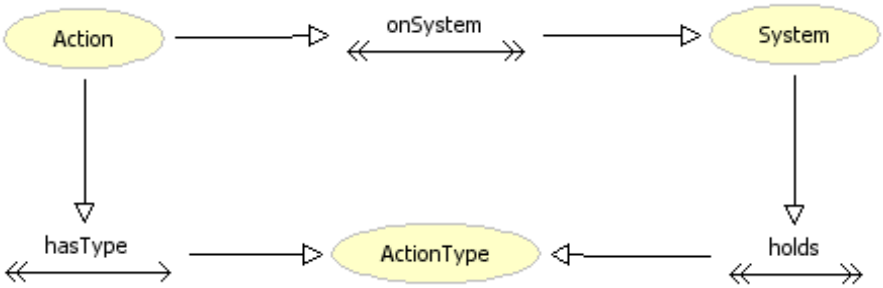

Figure 9: A simplified version of consistency loop.

The following rule can check if the system on which an action is defined holds an *ActionType* incompatible with the one of the action itself:

$$Action(?a), onSystem(?a, ?s), holds(?s, ?t1), hasType(?a, ?t2), incompatibleWith(?t1, ?t2) \rightarrow \perp(?a)$$

In the same scenario, the following property checking rule verifies that the action is supported by the system. For simplicity the action is classified as *"Ok"*, while in practice a more complete representation will be necessary to enable multiple criteria verification.

$$Action(?a), onSystem(?a, ?s), holds(?s, ?t), hasType(?a, ?t) \rightarrow Ok(?a)$$

After that, the custom reasoner can use the verification result to query for all the actions that are not marked as *Ok*.

2.3.3 Ad-hoc reasoners: Enrichment Module

An Enrichment Module (EM) is a software module that inserts into the ontology information that comes from logical inference (derivation) or network analysis (e.g., test the SSL server in order to obtain the list of available cipher-suites). Figure 10 shows how it works. It takes as input a set of ontology individuals and returns some Abox assertions that will be added to a subset of them.

Every EM has the following characteristics:

- *reads information from the ontology*: it takes as input the individuals of given input concepts;
- *outputs information into the ontology*: writing Abox assertions, the standard reasoner derives the new classification;
- *performs an atomic operation*: it writes the updates on the ontology only once, when it finishes all the computations.

The EMs can be described by providing the following information:

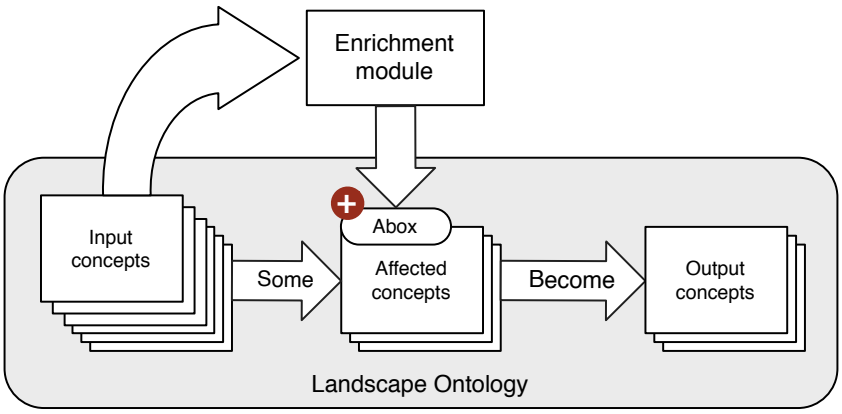

Figure 10: Enrichment Module general workflow.

- *input concepts*: the concepts to which the individuals that are read by the EM belong;
- *effect*: which ontology concepts will be reclassified and what type will they turn into as a consequence of the added Abox assertions;
- *completeness*: a boolean value that defines if the EM is always able to complete the transformation for all the individuals of the input classes, or can sometimes require further input by the administrator.

Therefore, in general, an EM will be composed of a standard *Description Logics* reasoner, an integrated *SWRL* rule engine, and a set of application level reasoners.

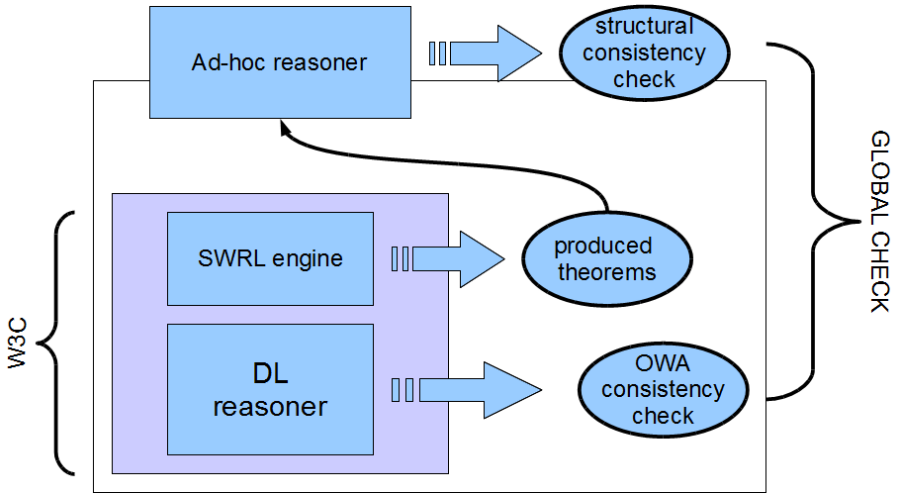

Figure 11: PoSecCo integrated reasoners

Figure 11 details the *policy definition and refinement* module from Figure 4. It shows the general case of an ad-hoc reasoner module, in which standard *DL* reasoning, rule based reasoning and ad-hoc application level reasoning are combined, taking advantages of the different approaches to face all the cases presented in this section.

At the lower level a standard *DL* reasoner derives all the theorems implied by the axioms in the ontology. This step is performed under the *Open World Assumption*. *Closed World Assumption* reasoning can be executed by running some forward-chaining derivation rules against existent individuals and properties. Finally, an application level ad-hoc module can drive some conclusions or perform custom checks by querying and manipulating the resulting model, composed of both logical theorems and facts derived by rules. In simpler cases, the ad-hoc reasoner, the rule inference module, or both may not be necessary.

2.3.4 Ad-hoc reasoners: Assisted Module

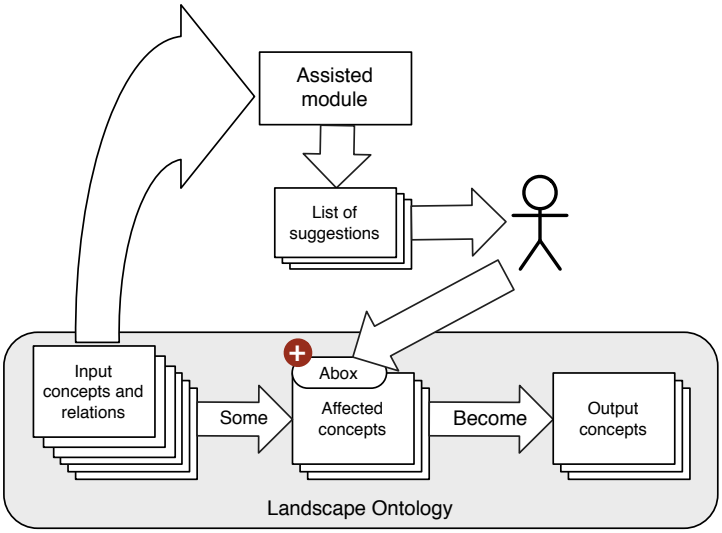

Figure 12: Assisted Module working scheme.

An Assisted Module asks the user for missing information by means of a GUI. Figure 12 shows how it works. It takes as input a set of ontology individuals and relationships and returns some suggestions that will be shown to the user, who will manually confirm or reject those suggestions. As a result, a set of Abox assertions are added to a subset of the input individuals.

It has the same characteristics of the EM and requires the same information to be described, thus it is interchangeable with Enrichment Modules that have the same effect or can be used after an EM that does not provide *completeness* to process the remaining individuals.

3 ONTOLOGY MANAGEMENT GUIDELINES

Besides the support for reasoning, one of the most distinguishing characteristics of formal ontologies used in the *Semantic Web* is the ability to represent both data and schema in the same language. Since the schema itself can be considered a model, and thanks to automatic reasoning services, the ontology life cycle can be assisted by tools and methods during the continuous process of ontology evolution and integration.

Generally speaking, with the term *ontology* we denote a semantic schema that represents a domain of interest. Such a schema contains classes (a.k.a. *concepts*) and relationships (a.k.a. *roles*) describing how objects in the domain relate with each other. Often some instances (a.k.a. *individuals*) are added to the ontology level because of their fundamental contribution to the semantics of the terms of disclosure. Sometimes such individuals belong to the actual domain to be represented, for example an ontology of natural numbers cannot be effective without the introduction of the number *zero*. In other cases, and this is the most common situation in information technology scenarios, ontology level individuals are "*technical*" individuals, in the sense that they are introduced to explicitly represent entities in order to guarantee some desirable properties of the ontology itself. Typically such technical individuals represents types of other instances through a technique called *reification* and are used to keep the ontology in the first order, preserving its decidability.

At a high level of abstraction, we can introduce a hierarchical architecture for ontology design.

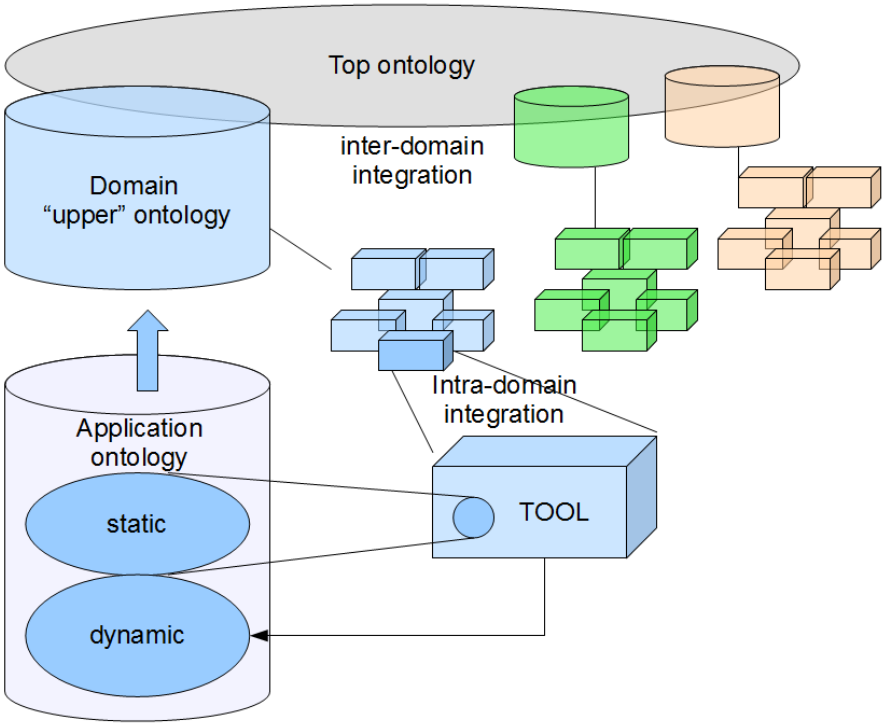

Figure 13: Ontologies hierarchy

Ontologies can be classified at least into three levels:

- The *top ontology* is a high level conceptualization of common and abstract concepts. There are different proposals like *SUMO* (*Suggested Upper Merged Ontology*), *GFO* (*General Formal Ontology*) and *COSMO* (*Common Semantic Model*). An upper ontology aims to work as a common starting point for any possible application domain, by introducing concepts like *temporal entities* for time concepts. The common understanding of these terms facilitates the integration of systems and tools devoted to

solve problems in different application domains, although a large amount of manual mapping is still necessary due to the low degree of interconnection between core elements from different domains.

- For each domain of interest, or at least for some groups of them, specific and deeply detailed ontologies can be defined. Such reference models are also called "*upper*" ontologies to highlight their vocation to be used as a common understanding of upper level concepts by knowledge engineers. Domain upper ontologies are used both to reuse useful models of significant fragments of the domain and in order to guarantee a simple mapping between specific ontologies from the same domain. If the domain ontology is very detailed, it could even be used directly by a specific tool in very standard scenarios. However, in most cases the developer will further refine concepts and individuals from the domain ontology. Even in these cases a well defined domain ontology can reduce the mapping between different implementation models to a fully automated reasoning. In the security and process domain we can refer to the *OWL-S* (*OWL for Web Services*) ontology models to represent high level concepts for both the functional and the security components.
- At the lower level, each application will define its own detailed and task-oriented representation of the domain. This lower level model is the *application ontology* and usually refines the reference *domain ontology* with some specific classification and by adding all the technical instances needed by the specific implementation.

So, the *application ontology* is specifically tailored for the needs of the application layer. The *Application ontology* is the connection point to the semantic model for the application. In fact, a complex axiom system can define the semantics of each term in the schema and the relationship between all of the terms; however, a small portion of such terms must be natively known by the application (in short, they must be hard coded inside the source code of the application itself). Obviously this is a mandatory step if we want the application to take advantage of the semantic model, and the *application ontology* is the natural place for the terms natively used in the tool code. The set of all the application hard coded terms is in general a subset of the *application ontology*, and we will name it the *static application ontology*.

The rest of the *application ontology* can be simply a set of terms and axioms constraining the meanings of the terms in the *static* ontology. In this case the semantic schema is used to derive theorems about the individuals in the model. Such consequences are observable by the application using only the hard coded terminology fragment.

In more complex and flexible scenarios, semantic models can evolve and can be enriched to better represent the application domain or to keep the system up to date with a dynamic domain. This evolution is in general much faster than the evolution of the application code. A modification of a declarative model is much easier than a modification of the code or of an algorithm, and it can take advantage of reasoning and consistency check techniques to guide the system to a correct (sound) new version of the ontology. So, in general, portions of the *application ontology* will change to reflect new domain characteristics (e.g., the introduction of a new system type or a new policy profile in PoSecCo system). These fragments compose the *dynamic* portion of the *application ontology* and, due to the natural introspection capabilities of *Semantic Web* ontologies, can be browsed and examined by the application to take care of the model enrichments without the need for imperative code modifications. For example, this is the case of a wizard that guides the user through different steps to obtain a complete definition of the landscape. If a new type of device is introduced and classified at the schema level as a specialization of the *Device* concept, the tool can navigate through specialization links starting from the *static* concept "*Device*" and can propose the new device type to the user. In this scenario the hard coded elements are simply the "entry points" for the browsing of the ontology and not only the set of possible terms used for querying and populating the semantic model.

3.1 Ontology extension and maintainance

Ontology maintenance is a necessary process to keep the ontology up to date. In most cases PoSecCo Ontology extension and maintenance will involve the lower level concepts and properties from the *dynamic*

application ontology. This will be the main way a user can follow to extend and refine the functional, landscape and security ontology.

If a significant change is performed to the ontology a regression test must be executed to verify the model to be consistent and that the desired behaviors of the involved tools are preserved. Functional regression tests are quite similar to classic tests and can follow the usual software engineering best practices. Ontology inconsistency diagnosis and remediation can be guided with the assistance of the reasoning service. Obviously, inconsistency cannot be explained by a simple counterexample like the lack of some schema property. However, many recent reasoners provide so called *explanation tools*, which are able to provide justification for all the derived knowledge, and for inconsistency in particular. Such a diagnosis is obtained by collecting all the axioms responsible of the inconsistency. Then, if a reasoning service is available, the user can be guided to the relevant parts of the ontology.

In rare cases upper level concepts from the *static* fragment of the application ontology may require maintenance too, for example to reflect some adaptation at the lower levels or simply to fix mistakes. If the *static* fragment of the application ontology is changed or enriched, the involved tool must be analyzed and extended to support the new features.

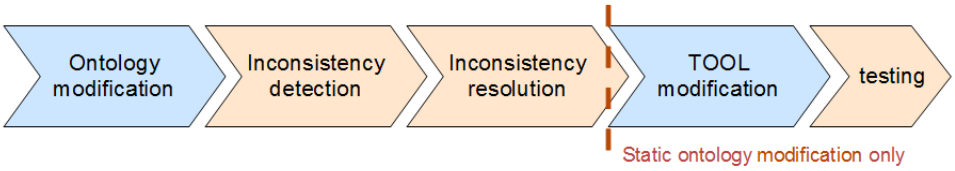

Figure 14: Main taks in ontology update

3.2 Inclusion of third party ontologies

In the context of the *Semantic Web* all the models and the ontologies developed by an engineer and supported by a tool must always be considered as partial representations of the actual world. It is fundamental to be ready to accept and integrate new knowledge from other players, both to refine or correct the current version of our model and to extend the scope of our system to new affine domains.

The integration of third party ontologies can be considered in two different ways:

1. a task to keep a system compatible with third party ones, for example in a supply chain system or whenever a semantic integration is necessary to enable the communication between two independent systems;
2. a process of including some knowledge from an external contributor into our own model. This not only enriches the available terminology, but also includes new axioms that can further restrict the relationships between elements in the ontology.

Integrating third party ontologies means adding new terms and new axioms from external sources to the PoSecCo ontology. It is a special case of ontology extensions, so all the considerations from Section 3.1 apply. However, in this case the knowledge change almost always consists in the addition of new terms and actions, with no removal of any preexistent knowledge. *Description Logics* are based on a monotonic semantics, this means that adding new axioms all the previously proved theorems still hold. This means that, as long as no inconsistency is introduced by integrating the new ontological fragment, all the behaviors of the system are still legal w.r.t. the new version of the ontology and no regression test at the application level should be necessary.

The first phase to be executed is the *ontology alignment* or *ontology matching*. Different techniques are presented in the literature, but the fundamental idea is to measure the similarity between concepts, individuals

and roles from the ontology that we are importing and the ones already in the application ontology. This similarity can be computed on the basis of:

- Terminological similarity, simply computed via stemming and lemmatization processes applied to all the textual descriptions related to ontology nodes (like node name, unique *URI*, labels and comments). Cross domain thesauri, like the well known *WordNet*, can provide the background knowledge necessary to guide this phase.
- Structural similarity. On the basis of a terminological measure, similarity can be refined by analyzing the way similar nodes are connected with each other within the respective ontologies. A very similar topology may indicate a stronger similarity.
- Semantic matching can be finally proved if a sufficient number of nodes are already mapped from one ontology to the other.

On the other side, knowledge integration can be much more complex than general ontology maintenance. In fact, the ontology fragment being added to the model is not manually defined by the knowledge engineer. It is taken from an existing ontology and can be designed on the basis of different approaches or with other use cases in mind. This can pose some issues during the import phase. For example the granularity of the imported ontology may be different from the desired one, or it could describe as individuals some entities that should map to concepts in our ontology. In these cases automatic ontology alignment usually fails or produces partial and very limited results. A manual intervention of the knowledge engineer is required and in a *harmonization* phase he will translate imported resources to the right level and will introduce all the technical individuals that are task oriented for the application ontology, but are usually represented as classes or may even not exist at all in the imported ontology.

4 THE POSECCO META-MODELS AND ONTOLOGIES

In this section we present the main features of the meta-models that are going to be used to manage the representation of the system in PoSecCo. The overall architecture of the models has been introduced in Section 1.6. The first version of the IT and Infrastructure models will be ready respectively at M15 and M18, according to the project plan, and will be presented in deliverables D2.3, D3.3 and D4.5. The current deliverable offers the opportunity of showing the current status of the proposals. In particular, the Access Control model, not covered in other deliverables, will be presented in Section 4.2. Evolutions can occur in the months following the publication of this deliverable. The description offered here can give a good indication of the structure of the solution that will be used in the project. The Configuration model is not discussed, as it is in a preliminary state and will be subject of significant work in the next few months.

4.1 IT level functional view

The IT level functional model has the goal of describing the main functional components that characterize the services offered by the system. The meta-model consists of classes that focus on the Future Internet scenario, with attention mostly paid to the scenario of service providers cooperating with suppliers and other service providers in order to offer to customers the functions they need. There is a clear emphasis on the description of aspects associated with a Service Oriented Architecture, but the model has been designed to be also applicable to the management of generic distributed applications.

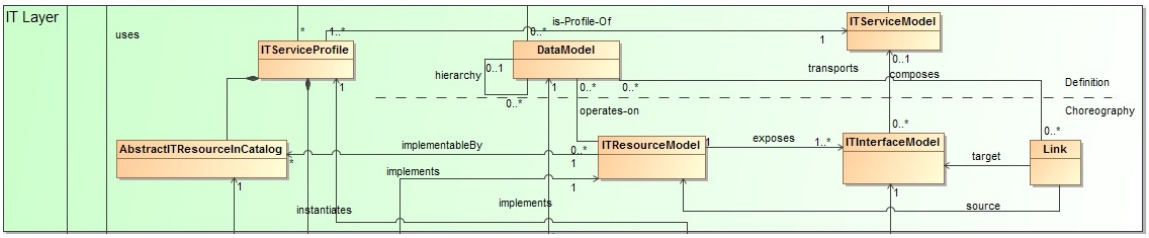

Figure 15: The IT abstraction layer.

The PoSecCo project is not planning to build a complete system design environment, rather it is focused on the management of security requirements. Then, the attention is paid to the identification of the components of the architecture that are of interest for the representation of the security policy. The consequence is that existing proposals for the representation of services in an IT architecture cannot be directly applied, as they require a greater level of detail than the one needed to support the security administrator. The Web Service Choreography Description Language (WS-CDL) represents a relatively high-level description of the structure of services, which also takes into account the need to manage the coordination among services offered by a plurality of parties. WS-CDL has been an inspiration for the design of the IT level functional model, but it also cannot be directly used, as it focuses on the specification of how services offered by independent suppliers can cooperate.

The meta-model consists of classes ITServiceModel, ITServiceProfile, AbstractITResourceInCatalog, Data, ITResourceModel, ITInterfaceModel, and Link. The relationships between the classes are graphically represented in Fig. 15. The classes of the model are described in Deliverable D4.2. An OWL representation of the model has been produced. The advanced features of ontologies are not particularly significant for the management of this ontology in isolation. Significant advantages are instead obtained by the definition of rules that verify the consistency in the reuse of the concepts of this ontology in the security and landscape ontologies. A few examples will be presented when discussing the application layer.

4.2 Access control ontology

The IT layer Security ontology supports the representation of authentication and access control properties. Figure 16 shows a graphical representation of the overall model. The model is structured in parts, where each part contains all the entities describing one aspect of the model.

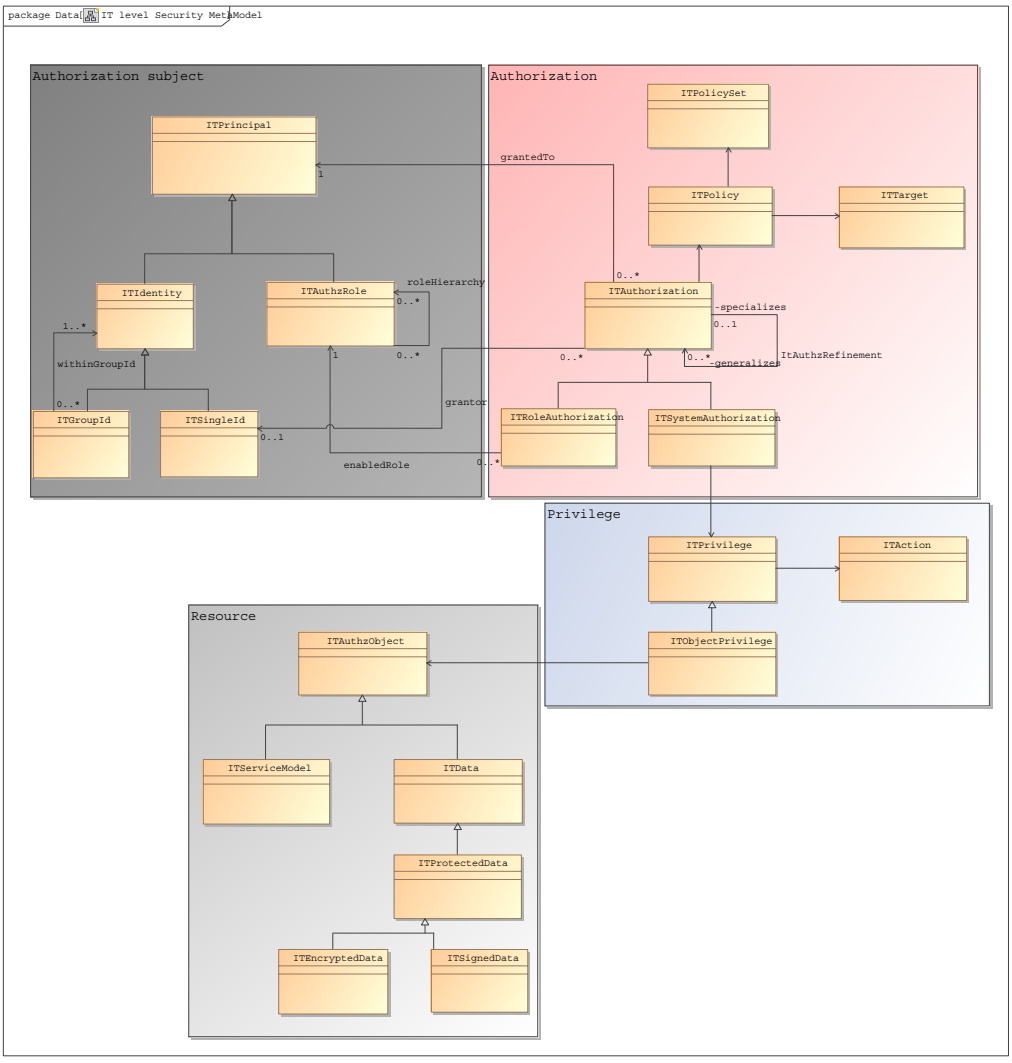

Figure 16: IT-level Security Meta-model.

The IT meta-model consists of four related concept blocks, describing different aspects of the security environment.

- the *Profiling* meta-model is used to describe the organization of identities, how users are structured into groups and allows to assign roles to users;
- the *Authorization* meta-model describes a taxonomy of authorizations and some basic properties of authorizations themselves;
- the *Privilege* meta-model is used to describe the structure of privileges;
- the *Resource* meta-model describes the structure of the static and dynamic resources that are associated with the privileges in the authorizations.

In the remainder of this section we briefly describe each class in each model block, as well as the fundamental relationships between them. The model offers a concrete demonstration of the benefits that can be expected in the project from the use of ontologies. The first crucial advantage offered by ontologies is the possibility of checking the consistency of the model instances, going well beyond what can be offered by classical modeling tools. A second advantage is represented by the opportunity of managing with ontological tools some design problems that would otherwise require the implementation of ad hoc tools. The flexible and expressive consistency checking is applied in many points in the schema. A concise example is provided in the presentation of the authorization meta-model.

4.2.1 Profiling meta-model

The classes in this model are ITPrincipal, ITIdentity, ITGroupId, ITSingleId, ITAuthzRole.

ITPrincipal ITPrincipal represents a generic user profile or group. It is an abstraction of whoever person or whichever entity can be the beneficiary of an authorization (or has to be restricted by a negative authorization). It has two subclasses: ITIdentity and ITAuthzRole;

ITIdentity ITIdentity introduces a static hierarchical user classification. In general, a principal can correspond to any user or to any specific organization unit that corresponds to a group of identities. Properties that describe the configuration of authentication services will mostly be associated with this class.

ITGroupId An instance of ITGroupId is a specialization of the ITIdentity concept and denotes (directly or indirectly) a set of lower level identities. Groups are organized in a taxonomy via the *contains* relationship to instances of ITIdentity. This means that a group can contain one or more groups as well as one or more specific identities.

ITSingleId Each instance corresponds to an Identity that represents a single user in the system. A SingleIdentity can be considered as a singleton group, but we preferred to model it as a separate class because it is not involved as a container in any contains relation. Single identities can be people, but also legal entities (e.g., a trading partner or a customer in the Crossgate EDI application are not necessarily associated with a specific person).

ITAuthzRole ITAuthzRole instances offer a variety of interpretations, adaptable to the specific features of the access control models implemented in the landscape. ITAuthzRole instances may correspond to “organizational roles”, to “collections of privileges”, and to “accounts”. There is no clear line of separation between these different interpretations in the scenarios PoSecCo is going to consider, thus it is considered appropriate to keep a single IT level concept for the representation of this variety of situations. The critical feature is that roles are separated from the representation of the real identities and the rigid organization described under the identity hierarchy. We can shortly analyze the interpretations considered above.

- *Organizational role*: this is the use of the ITAuthzRole entity that better corresponds to the BusinessRole entity appearing at the Business level. The representation provided at the IT level is more concrete and specifies the collection of privileges that have to be available to the members of an organization that have the responsibility to execute a specific function[5]. The assumption here is that typically the roles will have to be “enacted” before they can be used, and the use of the concept at run-time will be strictly connected with the idea of a “session”. Systems may put restrictions on the number of roles that can be used at the same time (e.g., in SQL there is the restriction that only one role at a time can be active in a session, and each SQL statement has to be executed within the context of a session). The concept of session is crucial for the run-time evaluation of the Access Control (AC) restrictions, but this is not required for PoSecCo, which deals only with the static configuration of the policy and does not consider its enforcement.

[5]Note that we do not have a direct connection between the BusinessRole and ITAuthzRole entities; the connection is mediated by the use of a BusinessRequirement that specifies the security requirements associated with a specific role.

- *Collection of privileges*: An ITAuthzRole instance is the container for a set of privileges that have to be assigned to an active party in the system. This is similar to the organizational role, except that no restriction is imposed on the dynamics, and potentially many collections of privileges can be given to a user and exercised at the same time. Historically, the concept of role in RBAC models was associated with the first interpretation, but currently the emphasis has shifted to the "security management" aspects, rather than the run-time aspects, and the interpretation of a role as a collection of privileges is probably today the most common one. At the IT level in PoSecCo there is no need to distinguish between the two interpretations.
- *SystemAccount*: An instance of ITAuthzRole can represent an account available on a system (Application/DBMS/OS). The advantage of putting this concept within ITAuthzRole is the possibility to manage as authorizations the assignment to identities of the privilege of activating a specific account in a system. This is a specific aspect that has to be considered in PoSecCo, because the security policy has to look at a complex infrastructure. Common access control models refer to a single component and do not have this need, as they essentially combine the concept of user with the concept of account on the system.

4.2.2 Authorization meta-model

The authorization meta-model contains classes ITAuthorization, ITRoleAuthorization, ITSystemAuthorization, ITPolicy, ITPolicySet, and ITTarget.

ITPolicySet A policy set represents an aggregation of policies. The motivation for the introduction of the policy set is to offer a level of aggregation of policies that supports the integrated representation of the business requirements. This approach corresponds to the solution in XACML, that offers these two levels of aggregation.

ITPolicy A policy consists of a set of authorizations. It is associated with a target that specifies restrictions valid over all the authorizations in the policy.

ITTarget The target permits to specify for each policy a set of restrictions that have to be applied to all the authorizations in the policy. The target can specify restriction over the principal and the privilege/resource managed by the authorizations in the policy.

This component has a crucial role in PoSecCo, because the model has to define a security policy for a complex, federated, heterogeneous landscape. Opposed to classical access control models, which have an implicit single Policy Enforcement Point, the component responsible for the enforcement of the policy has to be explicitly provided. The target entity can also be the component where powerful features of the language can be introduced, like the use of a declarative language. The specifici language will be investigated in the project. Natural candidates are XPath, like in XACML, or XQuery. Careful attention has to be paid with respect to the management of variables. This approach shows a strong similarity with the structure of XACML.

ITAuthorization ITAuthorization instances apply to principals and are possibly tracked to be granted by a specific person (ITSingleId). Two different types of authorizations are foreseen: ITRoleAuthorization and ITSystemAuthorization. ITAuthorizations have a sign, so we have a total partition in Positive ITAuthorizations and Negative ITAuthorizations. Techniques for the management of conflicts can be referenced. In general, it appears convenient to assume a model where negative authorizations (prohibitions) can only be defined as exceptions to previously defined positive authorizations.

Optionally an authorization can be marked with a "grant option", which means that the assignee can assign the same authorization to another user. (The SQL access control model offers this capability; in an environment describing the security management of a complex infrastructure, there are significant opportunities for the use of this feature)

ITRoleAuthorization ITRoleAuthorizations represent the authorization to enable a role. An ITRoleAuthorization permits to specify which principals are associated with a role. Negative ITRoleAuthorizations should

probably be avoided (they are reasonable only if they are managed as exceptions, and this adds a level of complexity probably not needed). The idea of having explicit role authorizations was presented in the original proposals for RBAC models. The XACML profile for RBAC uses the same approach, with two suggested kinds of XACML rules. Roles are disjoint from identities and they cannot belong to groups.

ITSystemAuthorization ITSystemAuthorizations correspond to the classical authorizations/rules of most access control models. Authorizations are often considered as <subject,resource,action> triples. The model proposed relies on a different approach, where an authorization associates a subject (the principal) with a privilege. The privilege is described in the next section. This choice increases the flexibility, because it can be immediately used to describe resourceless privileges (like the “auditing”, “shutdown”, or “backup” privileges that appear in operating systems and DBMSs access control model, not associated with a specific resource).

The sign is associated with the authorization and no hybrid authorization is possible, that is, a system authorization cannot mix positive and negative privileges. If the Windows access control model (which allows the mixing of positive and negative authorizations) has to be mapped, a concept of ordering among the authorizations has to be introduced.

As an example of the use of the advanced features of ontologies for consistency checks, we notice that the specialization of authorizations has to create a correspondence between ITAuthorizations a_f and a_c where the specialized authorization a_c cannot have as subject a generalization of a_f's subject. Also, the correct management of the links between each entity in the schema and the BusinessRequirement entity, that permits a hierarchical structure, requires to use some reasoning. Ontologies permit to concisely represent general properties that hold for each entity of this meta-model, like the possibility to be connected to an instance of the BusinessRequirement class at the Business level.

4.2.3 Privilege meta-model

The meta-model consists of entities ITPrivilege, ITObjectPrivilege, and ITAction.

ITPrivilege An ITPrivilege instance represents the right of performing some action on a resource. A privilege is always related to an action and can optionally (if it belongs to ITObjectPrivilege) be related to a resource (e.g., in the Windows access control model there is a clear distinction between privileges, which are resourceless, and access rights, which appear in the DACL of a resource). A privilege with no resource is interpreted as a Generic privilege and indicates some system level action. For example "log access" can be granted without the reference to a specific resource

ITObjectPrivilege An ITObjectPrivilege is a specialization of an ITPrivilege and represents a privilege that is associated with a specific resource.

ITAction An ITAction instance represents a generic business or technical action. Internally, the action may present a set of privileges (as in common ACLs). The structure of the action is strictly related with the characteristics of the underlying access control system. At the IT level it is possible to use rather descriptive actions, that will be made concrete in the refinement at the Landscape level. Dependencies can be modeled among the privileges (e.g., a rule may specify that an IT level “update” privilege requires an IT level “read” privilege), with the use of ontologies.

4.2.4 Resource meta-model

The Resource meta-model consists of entities ITAuthzObject, ITServiceModel, Data, ProtectedDataObject, and EncryptedDataObject.

ITAuthzObject The class is an abstraction of all the IT level entities in the functional meta-model that can be the target of an authorization. The entity loosely corresponds to the concept of “resource” in a XACML

authorization. The PoSecCo environment sees a large variety of resources to manage; the advantage of the introduction of the abstract class is that a single regular format can be used for the representation of access control privileges over different kinds of resource. The main distinction we introduce in resources is between static and active resources. This distinction corresponds to the classification of access control models, that are typically distinguished in “classical” access control models (OS, DBMS, etc.) and access control models for services and object-oriented systems that emphasize the description of privileges for software modules. This separation is justified by the separation between services and data that appears in the IT-level system meta-model.

ITServiceModel The entity is imported in the IT-level Security meta-model from the IT-level Functional meta-model. It describes any active resource that can be the target of an authorization. This entity is used in the representation of authorizations specifically designed for a Service Oriented Architecture. If we look at a classical OS, a program is stored in a file, and the user can be authorized to execute it, with two alternative modes in terms of privileges (executing the process in the user environment and exploiting the privileges associated with it, or using a special "setuid" mode that applies in the execution of the service the privileges of the owner of the program). For these scenarios, there is the possibility to describe the authorizations on the static resource. For the services considered in a SOA environment, either with a WS-* or RESTful invocation paradigm, the use of a different kind of resource appears appropriate.

Data This entity is also imported from the IT-level Functional meta-model. It describes a variety of static resources, like a file or a directory in a file system, an attribute or a view or a table in a relational database, or a specific piece of information in a business application.

ITProtectedData This specialization of Data has the goal of managing protection requirements that have been defined at the Business-level.

ITEncryptedData This further specialization of ITProtectedData is introduced to specify the encryption properties of the information that is subject to encryption requirements, when stored or in transit.

ITSignedData This specialization of ITProtectedData permits to specify the signature details for information that has to satisfy integrity requirements. The same instance of ITProtectedData can belong to ITEncryptedData and ITProtectedData (i.e., the hierarchy is not exclusive).

4.3 Application layer ontology

The main goal of the application layer ontology is to fill the gap of knowledge between the service choreography described at IT layer and the model of the low-level, either physical or virtual, network infrastructure. In order to do so, the ontology provides a concrete representation of the software directly or indirectly involved in the delivery of IT services, as well as the related technical interfaces allowing for service consumption.

Within the PoSecCo policy refinement process this ontology plays a primary role, constituting the entry point for the description of the security capabilities offered by the software for which PoSecCo aims at generating security configurations. Hence, applications are detailed mainly from the point of view of the functionalities that they offer being subject to the enforcement of security properties. For example, the SQL query functionality provided by a DBMS being subject to access control, or the invocation of web service operations being subject to message encryption in order to achieve confidentiality.

The basic concepts constituting the schema of the application layer ontology are the same as the ones contained in the PoSecCo application layer meta-model specializations. Each software category within the scope of PoSecCo (web services and web applications, databases, operating systems, etc.) has been modeled and described in PoSecCo deliverable D4.2, hence we refer to this document for the complete explanation of the schema (class hierarchy and relationships). The application layer ontology enhances the meta-model specializations by leveraging the embedded reasoning services, allowing for more expressive constraint checking and completing the missing knowledge in the model by the means of logic inference. The remainder of this section illustrates some examples of such features.

4.3.1 Ensuring deployment consistency

Applications are usually implemented by several pieces of software cooperating with each other in order to implement business functions. Different types of interactions are possible on several abstraction layers (link to static or dynamic libraries, IPC, sockets, RPC, etc.). In PoSecCo, network interactions (ISO/OSI layers 2 and 3) are mainly modeled within the network layer (topology), while the application layer is in charge of covering the relevant parts of the other models. However, modeling the entire internal structure of applications would require a huge effort and the result would be for the most part out of the scope of PoSecCo. Since the focus of PoSecCo is on security configurations, only those entities that are subject to security configurations are explicitly modeled. Figure 17 shows an extract of the application layer meta-model specialization depicting the class hierarchy of software components. As described in D4.2, `ITResources` represent whatever piece of software, either not directly involved in the delivery of a service or actually implementing parts of the IT service choreography. `Application Containers` and `Modules` respectively represent software components containing or providing additional functionalities to other components.

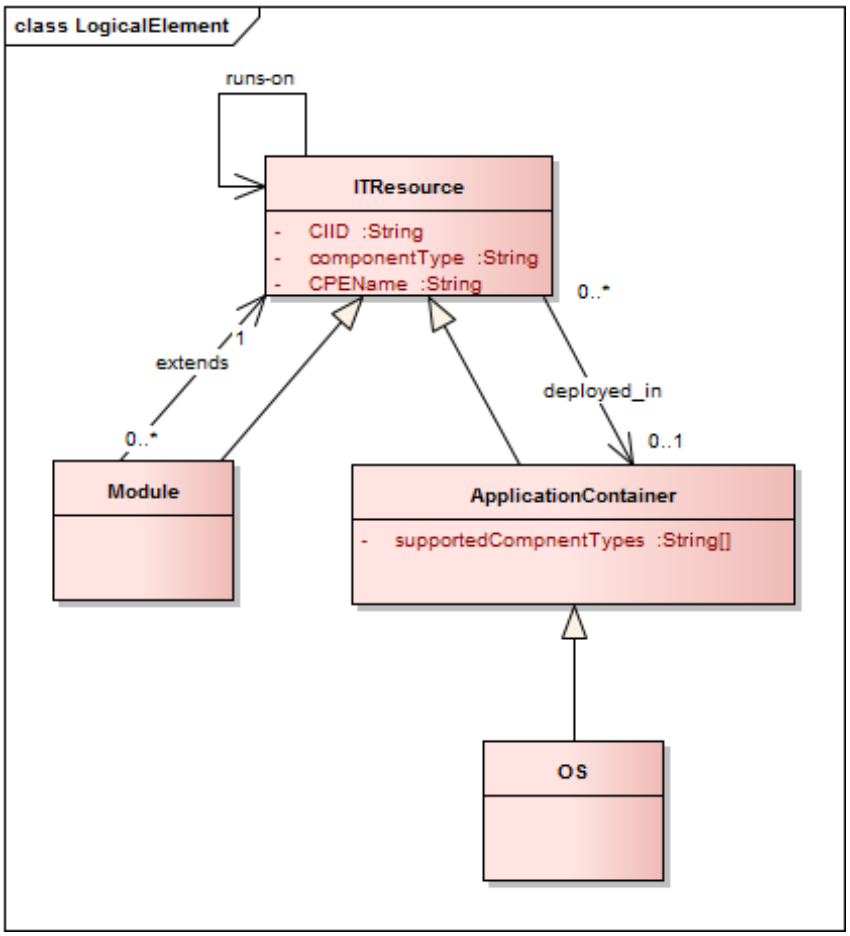

Figure 17: Extract of the specialization of the `ITResource` class of the PoSecCo functional meta-model

The actual meaning of a local interaction among software components is determined by properly sub-classing the `ITResource` class in conjunction with the use of the proper specialization (either `deployed_in` or `extends`) of the `runs-on` relationship. For example, an instance of `ITResource` connected through a `deployed_in` relationships to an `Application Container`, represents a piece of software deployed into some application server (e.g., a J2EE web application) and therefore inherits the security features (and configurations) of the container. In contrast, a `Module` linked via the `extends` relationship to the same `Application Container` would rather provide some additional security capability, therefore its security configurations could be propagated to the container (further details and examples are provided in D4.2).

Clearly the UML model does not prevent to instantiate the `runs-on` relationship instead of its specializations and this could lead to configurations that do not make sense in practice. For instance we do not want to have any `Application Container` *running on* a `Module`, or we may want to ensure that only compatible types of `ITResources` are deployed inside a given `Application Container`, so that a J2EE web application cannot be modeled as *running on* a PHP application server. Such constraints, not being included in the meta-model specialization, can instead be enforced within the ontology, via standard DL reasoning or rule-based reasoning. For instance, the following TBox construct forbids any `ITResource` to be run on any application `Module`, hereby fulfilling one of the requirements mentioned above:

$$ITResource \sqsubseteq \neg(\exists runsOn.Module)$$

In order to ensure the consistency within the application containers and the contained deployed modules, we define a *nominal* (enumerated class) describing the possible types of deployment formats, which — in the meta-model specialization — are represented as simple class attributes (`ComponentType` attribute in Figure 17):

$$DeploymentFormat \equiv \{WAR, EJBJAR, AAR, PHP4, PHP5, ASPX, ASP, ASMX, \ldots\}$$

We then define appropriate roles to represent the links between software components and the supported deployment formats and we use them to add number restrictions to the respective concepts:

$$ApplicationContainer \sqsubseteq ITResource \sqcap (\geq 1\ supportsDeploymentFormat.DeploymentFormat)$$

$$ITResource \sqsubseteq (\leq 1\ hasDeploymentFormat.DeploymentFormat)$$

We can finally classify the `ITResources` that are consistently deployed inside a compatible container via the following rule:

$$ITResource(?r), hasDeploymentFormat(?r, ?df), runsOn(?r, ?c), ApplicationContainer(?c)$$
$$supportsDeploymentFormat(?c, ?df) \rightarrow CorrectlyDeployedResource(?r)$$

4.4 Network ontology

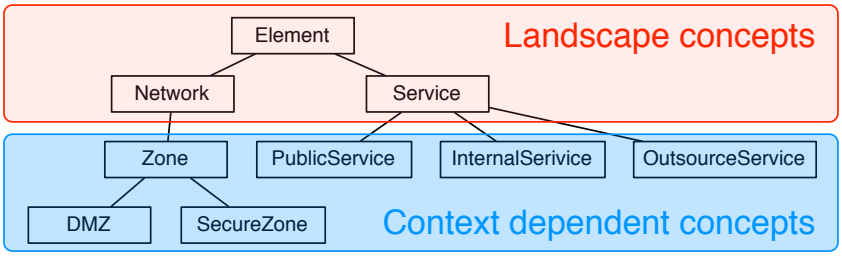

Figure 18: IT policy Context dependent concepts mapping to landscape ontology.

The ontology describing the network layer contains all the classes and properties of the landscape meta-model specialization described in Deliverable D4.2. This ontology adds properties and associations that permit to easily derive connectivity properties (at data link, network and transport layer) between landscape entities. Additionally, it supports several types of consistency checking, like, for example, instance checking and class specialization checking.

The main role of the network ontology is to contextualize the meta-model elements: the network ontology is composed of a core part, the “upper network ontology”, to which it is possible to link some “contextualization ontologies”. Even if some landscape elements do not convey particular information from the security point of view, usually companies and service providers categorize elements according to their position in the network, role and other security properties. For instance, starting from the *Service* landscape concept, we can define the *PublicService* and *InternalService* concepts that can be used inside an ITP to assign different

access control policies. The categorization strongly depends on the institutions owning the landscape. While some companies may distinguish between private and public servers, other companies can divide servers according to a set of security levels (e.g., high, medium, low).

Additionally, there are other categorizations that are induced by the policy. For instance, by resorting to a standard reasoner and an adequate definition of the contextualization ontologies, it is possible to automatically group services according to the type of the enforced policy or the profile of the customers.

All this contextualization consists in the addition of sub-concepts of the upper ontology concepts, as shown in Figure 18. Therefore, the network ontology permits to better fit the needs of different service providers by allowing to choose one of the PoSecCo defined contextualization ontologies or by defining their own.

4.5 Virtualization ontology

Virtualization is arising as a powerful and cost effective solution for a wide range of IT applications and it will be used by the Crossgate team for their testbed. The virtual ontology, i.e., the PoSecCo upper ontology concerning the virtualization, is capable of representing not only the virtual machines themselves, but also their virtualized devices and CPUs together with their characteristics. It can also model the network connections between the virtual machines and the physical computers, and several other relationships that exist between the virtual world and its physical counterpart.

The virtualization ontology has been built on top of the virtualization meta-model presented in PoSecCo deliverable D4.2, which has been inspired by the Libvirt XML configuration format. Libvirt is considered the standard de facto solution and it is used by most cloud management platforms, including OpenStack [6] and OpenNebula [7], therefore it is able to convey many useful information to reason about the virtualization environment and derive security properties.

The Virtual Ontology includes all the 272 classes included in the meta-model, together with 74 object properties and 102 data properties, making it a remarkably complex and powerful tool, which allows the PoSecCo user to realize a consistent description of a virtualized system with the desired level of detail. The level of detail is very fine grained because it has been developed in collaboration with the TClouds project (EU contract IST-257243, `http://www.tclouds-project.eu/`). In fact, it also includes information about CPUs, bus addresses and other configuration parameters whose use is not presently needed for refinement or conflict analysis purposes.

Furthermore, it also allows several types of analysis by resorting to a standard OWL-DL reasoner or to a custom inferential engine. The virtualization ontology enables a range of analyses. The most important one is the discovery of relationships between the physical hardware and virtualized elements (such as, for instance, when looking for the physical machines that provide the disk volumes to a domain). Knowing these relationships is particularly relevant for management purposes, to guarantee a proper allocation of the physical resources, as well as to assess the effect of the physical on the virtual world. On the other hand, this information is essential to enforce some categories of policies that are relevant in the context of PoSecCo, like, for instance, service isolation.

Another interesting aspect of the reasoning allowed by this ontology is the ability to understand when, according to properties derived from the policy, channel protection mechanisms can be avoided since the connecting elements run on the same machine.

In the same way as the network upper ontology, its expressiveness can be enriched by merging it with a contextualization ontology. Since the contextualization concepts are often independent from the physical or virtual domain, the contextualization virtual ontology will contain the same sub-concepts as the network contextualization, that is, together with private and public services available in the network contextualization ontology, we will have the virtual private and public services.

The ontology description in OWL-DL format is available at `http://security.polito.it/posecco/ontology`

[6] `http://www.openstack.org/`
[7] `http://opennebula.org/`

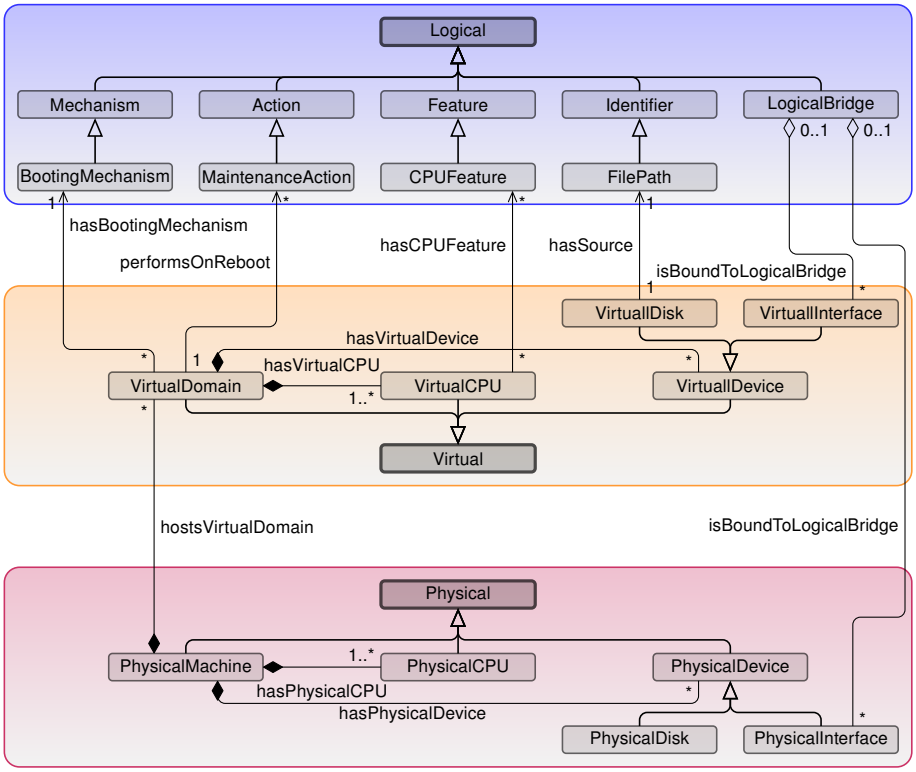

Figure 19: The virtual ontology: high-level UML diagram.

and it is described in the document. All the ontology concepts are organized in three main layers and presented more in depth in the following paragraphs. Each layer has a root class that contains all the remaining layer classes.

Starting from the Libvirt XML format, we have identified several classes which map all the objects that can be virtualized by a hypervisor. This brought us to the creation of the *virtual layer*, which contains elements such as the domains, together with all the virtual devices that are contained into the virtual machines.

One of the major limitations of the Libvirt XML format is the lack of a representation for the physical hardware. To bridge this gap, we have built the *physical layer*, which extends the PhysicalNode class in the meta-model (see D4.2, section 11) using the same base structure as its counterpart, the virtual layer. Their similarity allows us to represent both virtual and physical objects in a simple and modular way. It is important to notice that this layer is only intended to provide the minimum set of classes necessary for reasoning on the virtual realm, and it is not a comprehensive ontology of the physical hardware.

During the ontology creation process we have frequently encountered several “concepts”, such as bus addresses or CPU features, that do not fit exactly into the virtual or physical landscape meta-model. For the sake of modularity, all these concepts were classified into an ad-hoc layer, the *logical layer*. Its content is rather heterogeneous and will be discussed later. Even if these concepts do not have a counterpart in the meta-model, this does not affect the global view since they are local to the virtualization ontology.

4.5.1 Logical Layer

Since both the virtual and the physical entities make an extensive use of logical concepts, their description is provided first. The *logical layer* contains concepts that are used to describe several additional characteristics of physical and virtual objects, together with elements that are used to establish relationships between the virtual and physical layers. Furthermore, it includes all the software related classes.

The logical layer contains several classes. The most significant ones are `Action`, `Mechanism`, `Feature`, `Identifier`, and `LogicalBridge`.

The *actions*, represented by the `Action` class hierarchy, are entities used to define what the system must perform when a particular event occurs. Actions effectively model event-driven reaction concepts. For example, the class `VirtualMaintenanceAction`, depicted in Figure 19, is used to instruct a domain what to do when a system reboot or shutdown is requested. This class contains several individuals, each one representing a single action, such as the `Restart` and the `Destroy` objects, which respectively represent a simple virtual machine reboot or a full domain shutdown.

The *mechanisms*, modeled by the `Mechanism` class hierarchy, specify how a particular goal should be accomplished. Both actions and mechanisms describe how to perform a job, but the first are event-driven, whereas the latter are event-independent. For example, the `BootingMechanism` class defines how to start a guest OS. It contains several individuals that are used to specify how the boot sequence must be accomplished, e.g., launching the domain boot loader or letting the hypervisor perform a direct kernel launch.

The *features*, mapped to the `Feature` class hierarchy, are used to model the characteristics of a physical or virtual object used to declare what a physical component supports and what a virtual entity requires. For example, the `CPUFeature` class describes both the physical and the virtual CPU features, allowing the precise determination of what a physical processor offers and, on the other hand, what a virtual processor needs. This class includes several individuals, such as the `SSEFeature` and the `TSCFeature` objects, which respectively represent the Streaming SIMD Extensions and the TimeStamp Counter features.

The *identifiers*, represented by the `Identifier` class hierarchy, are named tokens that uniquely designate an object. Since the IT world extensively makes use of a wide range of such concepts, this class hierarchy is remarkably broad, counting 36 descendant classes. Figure 19 displays, as an exemplification, the `FilePath` class that models a file pathname, a concept used for several purposes, such as to specify an image file for a virtual disk. Another particularly useful and flexible identifier is the *network host*, represented by the `HostId` class, which is used to specify a generic machine using a DNS name, a MAC address, or more frequently an IP address, modeled by the `IPAddress` class. Identifiers contain additional notions such as directory pathnames and bus addresses (PCI, USB, CCID, ...).

The logical layer also contains the `LogicalBridge` class, which represents the concept of *logical bridge*, a core entity that is used to create virtual networks and to allow a domain to communicate with the outside world. Bridges are used to join together both physical and virtual NICs, connecting the two layers. Several virtual and physical objects, particularly the network interfaces, can use the `isBoundToLogicalBridge` object property to explicitly state their connection with a specific logical bridge. This class is crucial for a great variety of network analyses. A bridge acts in a hub-like manner, that is whenever an interface sends a packet, the latter is delivered to every interface connected to the bridge. Figure 20 depicts a bridge configuration comprising two virtual interfaces and a physical one. If the virtual interface A sends a message M, then the bridge will deliver a copy of this message to the other two interfaces.

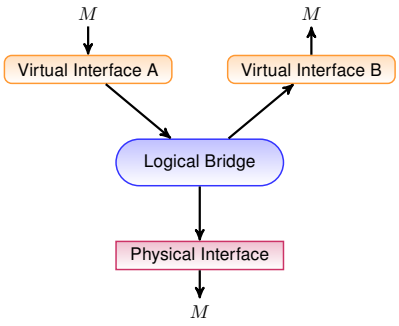

Figure 20: A logical bridge forwarding a packet.

Most of the virtual and physical layer classes are strongly dependent on the logical layer through several object properties, such as the `hasAddress` property which specifies the address owned by an entity, e.g.,

the IP address of a physical or virtual network interface. Furthermore, several virtual devices can have the `hasSource` and the `hasTarget` properties, which respectively define the data-source (e.g., a path to an image file) and the target address, that is the location where a guest OS will find this data (e.g., the `hda` disk).

4.5.2 Virtual Layer

The *virtual layer* includes all the components that are virtualized by a generic hypervisor, i.e., the virtualized hardware and the domain concept. It is mapped to the `Virtual` class hierarchy and a subset of its classes is shown in Figure 19.

One of the fundamental concepts is the notion of *domain*, which is represented by the `VirtualDomain` class. Since this class plays such a key role in our ontology, a great number of data and object properties can be attached to it. These attributes are used to link virtual machines to other virtual notions or to the physical world, relying on the objects provided by the logical layer.

Every domain has a set of *virtual CPUs*, mapped to the `VirtualCPU` class hierarchy. Virtual CPUs can be described in detail, for instance using the property `hasCPUFeature` that takes as value `CPUFeature` individuals, which define the requirements of a particular virtual processor.

Similarly to virtual CPUs, *virtual devices* are modeled by the `VirtualDevice` class hierarchy which represents all the virtualized hardware that can be attached to a domain with the exception of the processors. Virtual devices are basically virtual peripherals such as disks, virtual sound and video cards, and generally everything that can be connected to a virtual bus (PCI, USB, CCID and so on). The virtual device structure is vast and heterogeneous since every piece of hardware is extremely specialized. Figure 19 displays the `VirtualInterface` class, which models all the virtual network interfaces that a domain can use for communicating. The `VirtualInterface` class has several descendants that specialize the network interfaces, e.g., the `VirtualBridgeInterface` class represents a virtual network interface connected to a logical bridge. Figure 19 depicts also the `VirtualDisk` class, which models the virtual disks, i.e., hard disks, CD-ROM and floppy drives.

Another representative entity is the `VirtualPool` class, which describes the storage pools. A *storage pool* is a set of volumes, e.g., disk images, that can be used to implement several advanced storage techniques, such as remote disks or backing stores.

Since `VirtualDomain` is the core class of our ontology, we briefly introduce several object properties that are used to connect it to other concepts. The `hasVirtualDevice` and `hasVirtualCPU` object properties are used to specify that a particular domain contains a set of virtualized CPUs and devices. These properties also have a number of sub-properties like the `hasVirtualInterface` and `hasVirtualDisk` properties, that respectively specify the virtual interfaces and disks owned by a domain. Moreover, virtual machines are related to the logical layer entities, for example, as stated in Section 4.5.1, the `hasBootingMechanism` property indicates how a domain should boot its operating system. Furthermore, for each virtual machine we can specify what to do when a reboot is requested; the `MaintenanceAction` that will be performed is defined by the `performsOnReboot` object property. Additionally, the `hasHypervisorType` property is used to refine the domain type, by defining the compatibility between a virtual machine and a specific hypervisor.

4.5.3 Physical Layer

The *physical layer* is the counterpart of the virtual layer, containing all the tangible objects, i.e., the real hardware. Libvirt itself does not provide a description model of the real world, since this is outside its scope, but we felt that creating an ontology solely describing the virtual realm would not be very useful. In fact the virtualized hardware is strongly related to the real hardware and decoupling these two layers will severely limit the usability of our ontology for performing analyses. Therefore we created a hierarchical structure for the physical world, similar to the one described in Section 4.5.2 for the virtual layer. To aggregate concepts for reasoning purposes, the root class is named `Physical` and its internal architecture closely resembles the virtual layer one. Considering these similarities, we briefly discuss a selection of the most distinctive subclasses.

The `PhysicalNode` class represents a *physical machine* where a virtualization environment can be created, i.e., real computers hosting zero or more domains. Note that this class corresponds to the homonymous class used in the Infrastructure meta-model. This class is the physical counterpart of the `VirtualDomain` class and, in a similar way, several object and data properties are used to describe it in detail, ranging from the owned hardware to the hosted virtual machines.

The `PhysicalCPU` class models the *physical CPUs*, i.e., the real processors of a physical machine. The supported features of the physical processor can be specified using the `hasCPUFeature` object property that points to the desired `CPUFeature` class individuals.

The `PhysicalDevice` class hierarchy maps all the *physical devices* apart from the processors. Figure 19 depicts the `PhysicalDisk` and `PhysicalInterface` classes which respectively represent the physical disks (hard disks, CD-ROM and floppy drives) and physical NICs. The latter possesses an object property named `isPhysicallyLinked` that is used to represent a physical connection, e.g., a cable, between two physical network interfaces.

A virtual machine can contain several virtual CPUs and devices. A physical one can be described in a similar manner but with a set of physical processors and devices. This can be done by specifying the object properties `hasPhysicalCPU` and `hasPhysicalDevice`, which in turn have several sub-properties such as `hasPhysicalInterface` and `hasPhysicalDisk`, that are used to define the network interfaces and disks owned by a physical machine. Furthermore, the physical machines have the `hostsVirtualDomain` object property which represents its hosted domains. This property is particularly important because it links the physical layer to the virtual world. In addition, the `runsHypervisor` object property can be used to specify which hypervisor is running on a host.

4.6 Reasoning about security capabilities

Security capabilities provide the link between the functional and the security aspects of applications and in general of PoSecCo's modeled infrastructure components. They constitute a model of security-related functionalities being subject to security configurations, hereby defining the space of all the possible configurations that PoSecCo can generate. As such, they must provide all the information needed in order to generate configurations, for example a reference to the component of the landscape acting as enforcement point, but also to the elements on which a given security property is meant to apply.

Capabilities are particularly important because they determine the scope and the granularity of the concepts modeled inside the PoSecCo infrastructure model. These aspects are especially critical for the application layer models, which could in theory reach a great degree of complexity if designed to cover the complete internal structure of applications. Being the aim of PoSecCo to generate technology-dependent but vendor-independent security configurations, applications and their security capabilities are to be modeled accordingly, relying on standards or well-known specifications, like SQL or J2EE. Custom vendor-dependent features may nevertheless be covered by properly extending the vendor-independent models. Figure 21 shows a hierarchy of capabilities which includes some examples of security capabilities that applications can provide.

Since the concept of security capability is crucial, it has been included both in the PoSecCo meta-model and in the respective ontology schema. However, while the UML model merely allows capabilities to be attached to any kind of `ITResource`, the ontology provides support for more sophisticated inference of additional knowledge, as illustrated in the following paragraphs.

4.6.1 Capability inference

The PoSecCo landscape model is built for the largest part by querying the CMDB (Configuration Management Database) information model, assumed to contain a representation of the system landscape and to provide a faithful snapshot of the actual configuration of the system (see section 4.1 and 4.2.1 of the deliverable D1.1. However it's very unlikely that detailed information about software's security capabilities can be provided directly by such a system. On the other hand we can expect to dispose of details about the hardware and

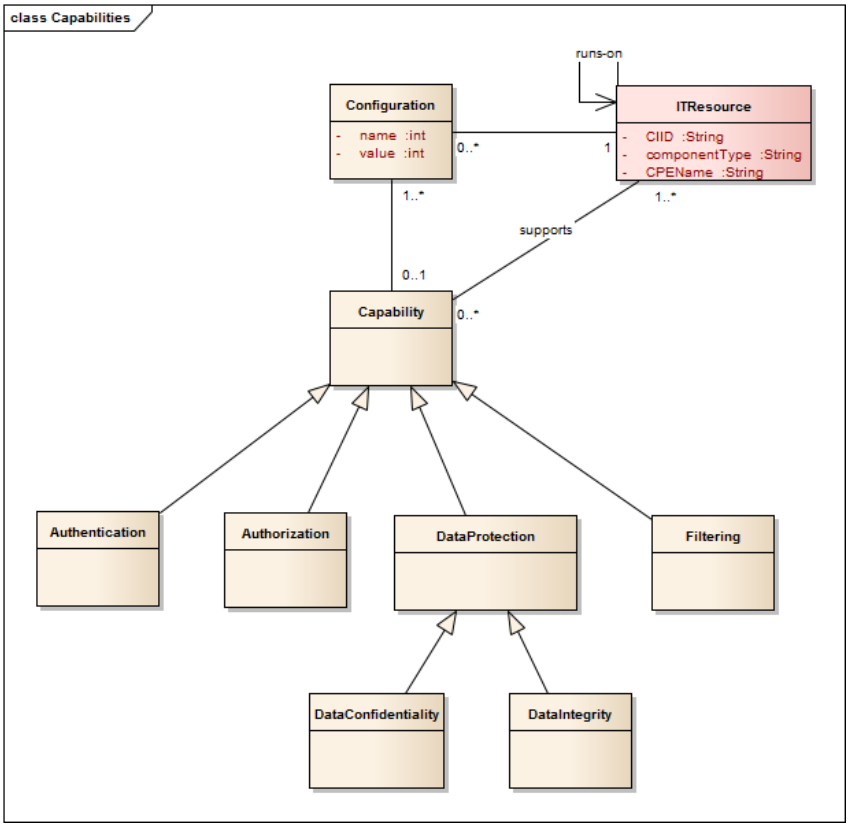

Figure 21: Capability hierarchy in the meta-model.

software components of the landscape (e.g. vendor, version, patch level, etc.), such information is encoded in the model through an attribute carrying a CPE [23] name for every `ITResource` (see Figure 17).

The ontology can include knowledge to automatically infer security capabilities for given patterns of software components. The object diagram in Figure 22 shows an example of a software configuration, where both web applications and web services are deployed inside a Tomcat web application server hosting the web service container Axis2. Solid lines linking instances identify relationships conforming to the meta-model specialization, whereas dashed lines are relationships being present only in the ontology and representing useful knowledge inferred from the model and related to the discovery of capabilities. As such they will be discussed in more detail within the next subsection.

All the components are identified by their respective CPE name. Such information, like the deployment structure given by the `runs-on` relationships, can realistically be retrieved by the CMDB. As previously stated, the CMDB may not contain information about software security capabilities. In this case, dedicated SWRL rules within the ontology will associate each specific piece of software with individuals representing the featured security capabilities, by relying on the CPE name of the `ITResources`:

$$ITResource(?r), hasCPEName(?r, \text{“}\texttt{cpe:/a:apache:rampart:1.3.0}\text{”}) \\ \rightarrow hasCapability(?r, WSSecurity)$$

The inferred *hasCapability* role, carries the same semantics of the `supports` relationships in Figure 21.

4.6.2 Capability discovery

During the policy refinement process, it is essential to build up knowledge in terms of the security capabilities of each component in the system landscape. This knowledge allows the refinement process to generate all

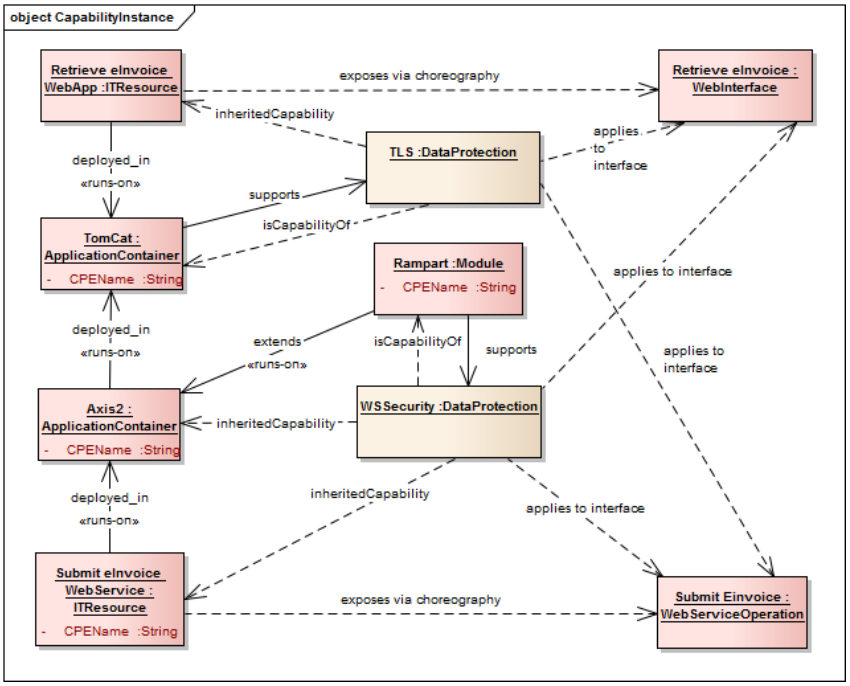

Figure 22: Security capability inference

the possible alternative configurations that satisfy a given IT policy. For instance, communication protection for a web service (IT policy) can be achieved both via TLS or message encryption (WS-Security), moreover secure channels at network level may also be configured through VPN. The PoSecCo ontology provides the essential link among infrastructure layers, which allows for performing several reasoning tasks in order to discover all the security capabilities that given a system landscape can offer in order to enforce security policies.

A prominent example of such reasoning tasks is the discovery of capabilities being propagated from one `IT Resource` (e.g., a web application container providing support for TLS) to another (e.g., a deployed web application inheriting the channel protection capability). A simple algorithm can be executed to discover this kind of inherited security capabilities, which propagates a capability among `IT Resources` related by either `deployed_in` or `extends` relationships (specializations of `runs-on`). It starts by identifying the `IT Resources` being directly involved in the implementation of the service choreography and therefore constituting possible terms (subjects and objects) in IT policies. Then it proceeds by propagating all the inherited security capabilities as follows:

1. for each instance r of `IT Resource` linked to the service choreography:
   (a) add the set of capabilities directly attached to it to the set of discovered capabilities C
   (b) for each instance i of `IT Resource` connected via any specialization of the `runs-on` relationship to r:
      i. Initialize the set of next nodes to be processed $S_i \equiv \emptyset$
      ii. add to S_i all the instances of `Modules` linked by an incoming `extends` relationship
      iii. add to S_i all the instances of `IT Resource` (but not `Modules`) linked by an outgoing `deployed_in` relationship
      iv. run the algorithm recursively from step 1 on all the elements in S_i

This algorithm can be easily implemented within the ontology via role chaining. After having subclassed the *runsOn* role (corresponding to the `runs-on` relationship) in *deployedIn* (and its inverse *deployedIn*$^-$) and

moduleOf, representing respectively the specializations of the `runs-on` semantics for standard containment (`deployed_in` relationship in the meta-model) and installed application modules (`extends` relationship in the meta-model), we define the *isCapabilityOf* role (semantically inverse of the `supports` relationship in Figure 22) and its sub-role *inheritedCapability*, which represents the relationship between a software and its inherited (propagated) capabilities:

$$deployedIn \sqsubseteq runsOn$$
$$moduleOf \sqsubseteq runsOn$$
$$inheritedCapability \sqsubseteq isCapabilityOf$$

We finally infer the *inheritedCapability* role, according to the capability discovery algorithm, through the following role chains:

$$deployedIn^{-} \circ isCapabilityOf \sqsubseteq inheritedCapability$$

$$moduleOf \circ isCapabilityOf \sqsubseteq inheritedCapability$$

The refinement process will leverage this inferred information in order to choose among the available alternative capabilities according to the refinement strategy. In this phase the security implications of choosing a propagated capability rather than a native one should be taken into account. For instance, confidentiality should be enforced at the closest point to the actual processing of the confidential information.

Further information about the applicability of security capabilities on the actual secured resources can also be automatically derived. For instance, security capabilities offered by web application containers, such as the ones depicted in Figure 22, can be linked to the interface on which they actually apply (e.g., the capability of enforcing WS-Security policies is linked with the exposed web services interfaces). Inconsistent links must be forbidden. For instance, WS-Security must not be applied to a web application interface.

5 POSECCO TECHNICAL INFRASTRUCTURE

5.1 The PoSecCo repository

In the PoSecCo scenario different tools must be used together to accomplish the different tasks involved in the workflow from requirements elicitation on business level to actual landscape hardware and software configuration. Different tasks may require different tools, implementing specific algorithm and using specific task-oriented information possibly expressed in specialized languages. However, these tasks are not strictly sequential. A central component of the architecture is the PoSecCo Repository, which is responsible of managing the different models that represent the input and output of the PoSecCo tools. An extensive discussion of the architecture of the PoSecCo Repository appears in Section 4 of Deliverable D1.2. After the Eindhoven meeting the Consortium agreed that the ontologies presented in this document also have to be managed as resources in the Repository, in order to facilitate the integration among the tools.

5.2 Semantic Web infrastructure

From the discussion in previous sections it is evident that PoSecCo suite will contain some tools that will interact with a semantic representation of requirements and specifications at different levels of abstraction.

Ontology enabled tools must be able to

- read and interpret semantic models w.r.t. one or more formal ontologies
- enrich such models and write new models from scratch
- reason on formal models, partially demanding standard reasoning to *Description Logics* tools as well as implementing ad-hoc reasoning

In order to do that, a language that provides good reasoning services and good library support must be selected. *OWL* (*Web Ontology Language*) is the main proposal from the field of the Semantic Web to express complex ontologies. *OWL* is the successor of previous XML based languages such as *DAML* and *OIL*, proposed by *DARPA* and then standardized by the *World Wide Web Consortium (W3C)*.

OWL semantics is based on the well studied family of the *Description Logics* and guarantees the decidability of principal reasoning tasks both at the schema level (T-box reasoning for concept satisfiability and classification) and at the instance level (A-box completion and query answering).

Nowadays *OWL* is available in two version: *OWL 1* and *OWL 2*. From the PoSecCo perspective the main difference between two versions is the *OWL 2* ability to express roles chaining, that is a restricted version of property composition. However, the advantage of roles chaining is surclassed by the use of *DL* and rule-based mixed reasoning proposed in section 2.3.2. On the other hand, *OWL 1* is actually more supported by existent tools and libraries and is deeper integrated with lower level standard tools like *RDF* storage systems.

Besides of that, *OWL* is well supported from both academic and industrial tools and libraries. According to licensing and maturity requirements two possible candidates can be adopted as foundational libraries:

Jena was initially developed in the HP labs at Hewlett Packard as a library to manipulate *RDF* graphs (much more in competition with *Sesame* library than with other Semantic Web initiatives). Since the development group was closed by *HP* a rich community supports it as an Open Source project. Jena is probably the most mature framework for the Semantic Web. It does not only manage formal ontologies expressed in *OWL* and *RDF Schema*, but covers a wide range of functionalities from the repository level (raw data manipulation) to the logical one (like built in reasoning, language compliance check etc.). It also integrates database storage both via a mapping to classic *RDBMS* schemas and native high performance triple store persistence layer.

Jena supports *SWRL* inferencing and a proprietary inference engine as well. Finally it integrates a *SPARQL* query evaluation engine that can be used to express complex queries.

OWL Api is a much more specialized library. It is developed and maintained by the Semantic Web research community and supports both *OWL 1* and *OWL 2* ontologies. As a drawback it only uses in memory models and provides no support for persistence management except the serialization of *OWL* models in one of the *OWL* standard supported formats (XML and text based).

Reasoning support depends on the expressivity required from the language (*RDF* or one of the different *OWL* sublanguages) and can vary from one library to another. For example, while *Jena* comes with a built in rule based reasoning engine especially tailored for *RDF*-like models, *OWL Api* does not provide built int reasoning support.
However, complete *OWL* reasoning generally requires an external reasoner. Beside commercial ones, like *RacerPro*[8] and *Cerebra*[9], the two main alternatives are:

Pellet is a mature *Description Logics* reasoner that supports both *OWL 1* and *OWL 2* correspondent logics (respectively $SHOIN(D+)$ and $SROIQ(D+)$).

HermIT is a newer solution, with a language support comparable to the *Pellet* one. The main difference is that *HermIT* is based on newer algorithms based on *Hypertableaux* reasoning. This often guarantees better performance if compared to classic *tableaux* algorithms (like the ones used by *Pellet*). On the other side *HermIT* is not as mature as its competitor and sometimes bugs are reported in the reasoning.

Since a variety of libraries and reasoners is available, custom connectors have been developed, mainly by library vendors, to guarantee interoperability. Unfortunately reasoners and libraries often make different assumptions on the ontology and the effort to maintain connectors up to date can become an issue.

In order to facilitate connections between model manipulation softwares (that manage model manipulation storage and exchange) and reasoner with their own internal structures some standard language have been proposed. The most prominent one is probably *DIG (Description Logics Interested Group)*. However, the maintenance of *DIG* specification was not able to follow the evolution of *OWL* and related tools. Recently a common agreed language and protocol from *W3C*, namely *OWLlink*, emerged as the standard interface to *Description Logics* reasoners.

As far as no *OWL 2* features are required to express PoSecCo ontologies and since *OWLlink* expressiveness is sufficient to express *OWL* axioms no mandatory choice is necessary. Each plugin that require semantic elaboration can use both *Jena* and *OWL Api* and can rely on both *Pellet* and *HermIT* reasoning capabilities.

[8] Renamed Abox and Concept Extension Reasoner: http://www.sts.tu-harburg.de/ r.f.moeller/racer/
[9] http://www.cerebra.com

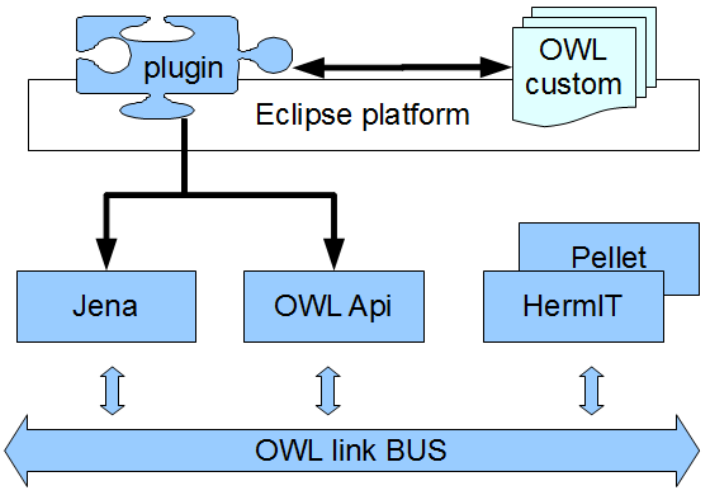

Figure 23: Libraries and reasoners for Semantic Web interface

6 CONCLUSIONS

The deliverable discussed the crucial role of ontologies in PoSecCo. The report has first presented the motivation of the interest that is going to be paid to ontologies in the project. Then, details about the use of ontologies in the project have been illustrated, with an analysis of the languages, tools and configurations that offer the best fit with the security management scenario considered by PoSecCo. Then, the status of the ontology for Access control, not presented in other deliverables, has been reported. Several examples of the use of advanced ontology management features in the management of the Application security model have been reported, with a coverage also of other portions of the models. The deliverable finally presented a concise analysis of technical implementation aspects. In conclusion, several results have been described that will provide guidance in the prosecution of the project.

The definition of the ontologies for the different layers and views of the system will continue, offering a robust confirmation of the role of ontologies for security management. This activity has significant impact for the communities operating in research, standards, and industry. In terms of research, attention has been paid in the past few years to the integration of ontologies and security; the focused goal and the effort that PoSecCo plans to spend in that direction, together with the integration of partners with complementary profiles (from university and industrial research centres, to large software developers and security auditors), promises to produce significant results and important research contributions in this area. Standards represent a necessary avenue for the concrete dissemination of the results into real systems. Current standards in this space are starting to consider ontological aspects. The greater attention dedicated to them in PoSecCo is a well justified innovation that fills a gap in the current set of proposals. The impact in this area can be expected to occur beyond the timescale of the project, but it represents a component that has to be considered during the life of the project. Finally, the impact on industry practices will take the longest time to materialize and will occur only after successful steps in research and standards. Apart from the work in PoSecCo, there is a clear indication from the analysis of the state of the art that ontologies are going to have a critical role in the security management area. PoSecCo has the ambition to offer a visible contribution to this development.

REFERENCES

[1] W3C, “OWL web ontology language,” 2004.

[2] ISO/IEC, “Iso/iec 24707:2007 - information technology common logic (cl): a framework for a family of logic-based languages,” 2007.

[3] C. Corp, “Ontological engineer’s handbook,” 2002.

[4] R. et al., “Gellish,” 2011.

[5] P. C. B. et al., “Idef5 method report,” tech. rep., Knowledge Based Systems, Inc., 1994.

[6] R. E. Kent, “The IFF basic KIF ontology,” tech. rep., Knowledge Systems Laboratory Stanford, 2002.

[7] W3C, “Rif overview,” 2010.

[8] W3C, “Owl 2 web ontology language,” 2009.

[9] W3C, “Resource description framework (rdf): Concepts and abstract syntax,” 2004.

[10] W3C, “Rdf vocabulary description language 1.0: Rdf schema,” 2004.

[11] M. Horridge and S. B. et al., “Owl-api,” 2011.

[12] S. C. for Biomedical Informatics Research, “Prote’ge’,” 2011.

[13] t. I. o. S. Mizoguchi Lab. and O. U. Industrial Research, “Hozo ontology editor,” 2011.

[14] R. S. et al., “Kaon,” 2011.

[15] E. M. et al., “The neon toolkit,” 2011.

[16] L. Clark & Parsia, “Pellet,” 2011.

[17] Openjena, “Jena,” 2011.

[18] B. M. et al., “Kaon2,” 2011.

[19] S. B. et al., “Dig,” 2006.

[20] A. A. et al., “Sofa,” 2011.

[21] J. Zhou, L. Ma, Q. Liu, L. Zhang, Y. Yu, and Y. Pan, “Minerva: A scalable owl ontology storage and inference system,” in *Asian Semantic Web Conference*, pp. 429–443, 2006.

[22] J. Lu, L. Ma, L. Zhang, J.-S. Brunner, C. Wang, Y. Pan, and Y. Yu, “SOR: a practical system for ontology storage, reasoning and search,” in *Proceedings of the 33rd international conference on Very large data bases*, VLDB ’07, pp. 1402–1405, VLDB Endowment, 2007.

[23] A. Buttner and N. Ziring, “Common Platform Enumeration (CPE),” 2008.