

Dealing with the Formal Analysis of Information Security Policies through Ontologies: A Case Study

G. M. H. da Silva¹, A. Rademaker¹,
D. R. Vasconcelos^{1,2}, Fernando Náufel do Amaral³, Carlos Bazílio³
V. Gonçalves¹ and E. H. Haeusler¹

¹ TECMF – Departamento de Informática – PUC-Rio
Catholic University of Rio de Janeiro,

Email: {hamazaki, arademaker, davirv, vaston, hermann}@inf.puc-rio.br

² Departamento de Computação – UFC

³ Departamento de Ciência e Tecnologia – PURO – UFF
Email: {fnaufel, bazilio}@ic.uff.br

Abstract

We present the structure of an ontology for Information Security (IS), applied to the extraction of knowledge from Natural Language texts (IS standards, security policies and security control descriptions). This ontology is composed of the vocabulary for the IS Domain, and a particular kind of ontology description, logical forms to determine the structure of the DL formulas associated with the texts. We also discuss the relationship between the structure of the formulas and the efficiency of the reasoner.

Keywords: Information Security Policies, Description Logic, Ontologies

1 Introduction

Information Security (IS) is a non-trivial problem that usually comes in different levels of abstraction. At the *Business* level of abstraction we can find the information related to *Processes* and *Persons* involved in any of the enterprise's activities. At the *System* level we find the *Software* and *Hardware* under safeguard. The *Enterprise's* or *Organization's* security management recognizes (Caralli 2004) this problem as an important question in its own right, not merely as a technological problem to be solved by means of software installation (as *Firewalls*, for example). Social aspects should be also taken into account¹ when studying security problems at the *Process* or *Person* level. On the other hand, there are many attacks reported at the *System* level², enough to take the technological problem also into account. Needless to say, a broken security protocol can prevent an *organization* from fulfilling its social role.

In order to solve the IS problem, two main approaches have been adopted: one based on the defense against predicted (or rather, hopefully predicted) threats, and the other based on maintaining the organization at previously established security levels. The former, named *Threat-Based* IS approach, tries to mount a strong defense against likely attacks, while the latter maintains the behavior of each entity in the organization at ever-controlled states. Most of the *security standards*, as well as *security protocols*, seem to adhere to each of the two approaches at some level, not necessarily to the exclusion of the other.

The IS community has created sets of *rules*, in a quite artificial variety of Natural Language, in the form of se-

curity conditions to be verified. These rules have been continuously updated, and an organization must satisfy them in order to be considered *secure*. Well-known examples of these sets of rules are the *standards* provided by some committees (ISO, COSO, ANSI, Brazil's ABNT, etc). Each set of rules, designed to a specific level of abstraction, carries a *terminology*, which is basically formed by the *linguistic* terms that denote the concepts involved in the specific domain of IS. One can verify by reading a typical *standard*, for example ISO-27001 (ISO 2005b), that it is presented as a set of phrases in a quite artificial Natural Language pattern. The reason for that is the (intended) lack of ambiguity that such texts must have. Besides the *standards*, each organization has its IS Policy (**ISP**), whose level of abstraction is significantly lower than that of a *standard*. One can easily verify that a condition in a *standard* must be turned into an obligation at the **ISP** level. And of course, in order to be implemented, this obligation must be written in a way that will facilitate the task of verifying the status of its own implementation. Thus, in *IS* terms we have two sets of rules and we should ensure that each rule of the higher-level set is covered by the rules of the lower-level set in every viewpoint for the application of the former. We call this *compliance testing*. Thus, besides the task of designing rules and proving them to be adequate to "reality" (the organization and its surroundings), there is the task of comparing, by means of some form of compliance testing, two or more sets of rules. This is what we call the **formal** statement of the IS problem.

For the people in the Ontology community, the above pictured scenario makes for an invitation to work. Ontologies are formalizations of conceptual worlds, and they also serve to prove certain properties about them. These proofs may show the adequacy of the Ontology to its object. Among those important properties one may cite (1) consistency testing, which ensures that the ontology has a model (i.e., it "speaks" about something), and (2) hierarchy classification (subsumption of concepts), which verifies the partial order among concepts. These tests are quite useful in our formal statement of the IS problem. In fact, there are currently many projects and initiatives worldwide that do just that. The references would be too numerous for us to cite them all. Only as a matter of preference, we point out the repository of ontologies placed by the users of the Protégé Ontology Editor (Stanford University 2006).

In (do Amaral et al. 2006) we describe the **Anubis** project, an architecture and a set of formal tools, as well as a methodology, to help a Security Company in designing, validating and maintaining a knowledge base on **ISPs**. As a case study and with the purpose of preserving industrial property (do Amaral et al. 2006), we omit many details,

⁰Research partially supported by CNPq grant 550652/05-1, The Anubis Project.

¹Social Engineering techniques can be envisaged in order to design an attack to access the Enterprises ordinary trash, by means of an intruder personified as an employee or acting with the help of other employees.

²CERT/CC (CERT/CC 2007) keeps records on most of them.

and in certain moments the actual names and denominations inside the architecture were presented in a more general way. The relevant information is briefly presented in Section 2. The structure used to represent Natural Language constructs in order to facilitate the task of formalizing IS concepts and rules from NL Norms is presented in Section 3. The aim of the present article is to report noteworthy conclusions that have been drawn during the development of the above mentioned project.

We focus on the two main conclusions discussed at the end of this article. Firstly, the particular style of Ontology Description, as a set of DL formulae, has become critical in terms of efficiency. It is interesting to note how the presentation style of “equivalent” sets of DL formulas (“equivalent” by means of a syntactic homomorphism) can produce highly different performance results in validation. Although linguistic features of the text to be formalized could argue in favor of a **nested logical form** representation, the use of **flat** representations has shown better computational performance. This is detailed in Section 4. Secondly, and not surprisingly, many *actions* conceived from a set of controls by a domain specialist have not been successfully validated against their respective sets of controls, at least not without the addition of particular axioms stating either obvious or automatic effects, transparent to the domain specialist. Section 4 lists some examples where this situation happened.

2 A brief presentation of the case study

The formalization of text-based information is an important issue in the deployment of semantics-aware technologies in the enterprise. It is very common to encounter situations where knowledge stored in natural-language documents must be made available to agents (human or software-based) for processing and decision-making. This case study can be seen as an attempt to provide an ontology-based approach to the formalization of normative texts in the domain of Information Security (IS), such as security policies defined by organizations and *standards* defined by Security Committees. In (do Amaral et al. 2006) we discuss the principles involved in the development of this approach.

Because the IS-related terminology tends to vary according to the source, we adopt the following definitions: a *standard* is a public document consisting of a set of *control objectives*, which are goals to be attained by the organization if a great level of security is desired. Roughly speaking, control objectives state *what* should be achieved; being expressed at a rather high level of abstraction, they do not lend themselves to direct application to the organization’s processes and practices. It is by means of *security controls* that the organization actually specifies *how* to achieve the security requirements laid out by the control objectives. Security controls (or simply *controls*) are low-level technical measures that can be deployed in order to protect the organization’s devices and processes against potential threats. To bridge the gap between high-level control objectives and low-level controls, the organization defines its *security policy*, consisting of *actions* to be taken in order to comply with the adopted *standards* and possibly with other security requirements identified by a process of risk analysis. In this scenario, one control objective may give rise to several different actions in the security policy, and each of those actions may be implemented by a set of different controls.

Many tasks are involved in the formalization of the IS domain as described above: for example, *standards* must be selected, actions must be formulated, controls must be defined, deployed and managed. Furthermore, all levels must support maintenance: updates in the *standards* must be followed, policies must be revised, and controls must be replaced or incremented because they become ineffective, inapplicable or simply insufficient. It should be clear

that security experts can greatly benefit from the use of semi-automatic, knowledge-based and formal tools to assist them in these activities. From the computer science community viewpoint, we would say that we focus on the use of CAV³ tools. In fact, in our case study we already have a set of *controls*, which comprise the knowledge base of a security analysis system marketed by our industrial partner. By an abuse of language, we call this set of *controls* the IS Knowledge Base (**ISKB**). Thus, the approach to be followed is to group *controls* into *actions*, checking their respective consistency and verifying the compliance of the group of *controls* with regard to the respective *actions*. Subsequently, *actions* and *control objectives* should be checked for compliance too.

Before explaining our approach in more detail, it would be interesting to mention the almost natural boundary conditions of our industrial scenario: (1) Modification of *Controls* (*Security Controls*) should be avoided, and their level of abstraction should be the lowest possible; (2) *Control objectives* cannot be modified neither in their contents nor in their form, since they are rigid documents (*the standards*); (3) *Actions* are designed by the human being for better clustering and understanding of the Base of *controls* ISKB; actions can and must be modified as a way to easily reach an understandable compliance between the *KB* and the chosen *standard*.

Our approach consists of the following elements:

- *Actions* are represented at the *logical form* level, a concept from the area of natural language understanding (Allen 1995). Basically, logical forms are constructs in some suitable formalism used to represent the context-independent semantics of natural language utterances. Currently the edition of *actions* and *controls* (and *control objectives* in the future) is accomplished by means of a Protégé Plug-in developed as an Ontology-Driven editor guided by the *logical forms* ontology.
- The inference capabilities of the proposed framework are based on a description logic (DL) (Baader et al. 2003). Logical forms representing actions are actually stored as DL concepts, and the facts that must hold about these concepts are stored as DL axioms. The user may pose queries to a DL reasoner, which will provide answers based on these axioms. Some background on DL is assumed in this article. The reasoner used in the experiments here reported is the Pellet implementation of the tableaux proof method for DL. Note that the IS domain and the need for additional axioms as reported in Section 4 could require a more powerful DL, but this has not happened in practice.
- Ontologies (Gruber 1993) serve as the unifying structure for the above two elements. An ontology consists of concepts, properties and logical expressions denoting constraints that hold between these concepts and properties. In our approach, actions in logical forms and axioms about them are expressed in terms of concepts, properties and constraints. One language for representing ontologies is OWL DL (Dean & Schreiber 2004), which can be translated in a straightforward way to the language used by DL reasoners, providing an easy interface between the ontology and the inference services of our framework.

The goal architecture to help formalize IS from NL texts guided by a domain specialist is depicted in Figure 1. The life-cycle of our IS ontology development is shown in Figure 2.

³Computer Aided Validation.



Figure 1: Elements of our ontology-based approach

Figure 2: Life-cycle of IS ontology development.

3 The rationale on the IS Ontology defined in the project

In (do Amaral et al. 2006) we discuss the decision to use a structured representation of NL constructs in order to facilitate the task of formalizing IS concepts and rules from NL standards. In fact, almost every rule has a verb (only one verb)⁴ as its most important word, and other elements are linked to it by attributes that are usually (but not always) determined by the syntactic roles they play in the sentence. For example, **hasTheme** is normally associated to the direct object, **hasAgent** to the subject, **hasPurpose** to a subordinated phrase and so on. These structures are described using DL formulas as part of the *Logical forms*, envisaged just in order to solve this situation in a NL processing scenario. We use these as the core of the Ontology. Thus, our IS ontology has a linguistic core that is responsible for describing concepts and relationships essential in structuring Normative texts as an indexed set of sentences (the *actions*, *controls* and *control objectives*). This core is semantically neutral. In order to reason about IS concepts, there must be an Ontology of the IS vocabulary. For example, *Server*, *Firewall*, *Operating System*, *CEO*, *Meeting Room* and so on are concepts belonging to this IS vocabulary.

One of the points that were strongly stressed during the project was the construction of the ontology by the domain specialist himself. The IS specialist is used to working with Natural Language texts, namely, *standards*, laws, company policies, security protocols (not only software, but also human-based security protocols) and so on. It was noted that the specialist prefers to keep working this way: his focus is on the written material. Discussing the general extraction of knowledge from Natural Language texts is outside the scope of this project; however, there is clearly a need for some methodology and a tool to support it. In order to solve this, a Protégé plug-in was implemented. This plug-in is an editor guided by the core ontology (an OWL-DL document), which, by interacting with an IS specialist, produces a set of DL-formulae representing either an IS vocabulary, or an *action*, or a *control*, or a *control objective*. The editing process works by marking and typing with DL meaning the linguistic elements of a normative text. In fact, the plug-in was not used for the edition of the almost 2000 controls and actions that make up the last version of the Ontology. Besides that, part of the vocabulary related to IS on the *Linux Operating System* was defined by an IS specialist (an employee of the Industrial Partner).

The main advantage of the use of *logical forms* is its compliance with the guidelines for Norms constructions (see (ISO 2005a)). For example, in most cases our ontology does not, and should not, distinguish between different parts of speech, so as to render the constructed logical forms as general as possible; thus, a noun whose root is shared with a verb is represented by the concept associated to the verb; for example, “connect” and “connection” are both associated to the single concept *Connect*, and “configure” and “configuration” are both associated to the single concept *Configure*. In (ISO 2005a), Sect. 6.4 states that all concepts should be preferentially stored in noun form; we have chosen verb form instead of noun form because the actions in a security policy have verbs as their

most important words.

Before we describe some examples that have allowed us to draw the conclusions mentioned in sections 1 and 5, a few words about our IS ontology (from the logical point of view) are in order. The main feature of our IS ontology is that it is a T-Box ontology. There are no individuals asserted, nor is there any mention of them. This is justified by observing that from the point of view of IS, individuals seem to have no importance. For example, a *Person* invading a private area of a corporation equals any other invader in the same conditions. It is not important to name the invader, but to describe him or her. The same can be said about the facilities of the corporation (organization), about the CEO, the *Operating System*. A password instance is unimportant whenever comparing the two concepts *valid password* and *invalid password*. Besides, because we deal with negation, disjunctions and conjunctions, and the core ontology uses restrictions, we use *ALC* (Baader et al. 2003) as our logic language. This has the (desirable) side-effect of not going so high in the complexity hierarchy of DLs. Basically we are inside PSPACE-complete worst-case complexity. This is not so far⁵ from the usual reasoning complexity for knowledge representation systems.

The next section is the core of this article, showing some interesting examples that appear in the IS ontology under construction. Both conclusions mentioned in the introduction have been drawn from almost the same kind of formalization. Thus, both claims are analyzed in the sequel.

4 Formalization of Controls and Actions of IS

In this section we illustrate the formalization of IS actions and controls from normative texts. We consider a few examples of actions and controls formalized by nested and flat representations. These examples provide samples of the different types of axioms that are needed in the ISKB in order to accomplish compliance tests. It is worth mentioning that the automatic process is simpler in flat form (in which the axioms are more modular) than in the nested style. In addition, the validation is also more efficient using flat forms than nested forms. At the end of this section, we show the performance of both approaches.

Before going into the examples, a short explanation. The *nested* style of specifying *Logical Forms (LF)* was our first choice; it was induced by the nested style of the modifiers (modalities, adverbs, subordinated phrases, etc.) usually found in Natural Language sentences. The use of attributes (roles) in our *LF* linguistic ontology as a way of specifying the role of each phrasal element in the phrase is a natural one. We recall that if R_1 and R_2 are DL roles, and C and D are concepts, there is no logical equivalence between the concepts $\exists R_1.(C \sqcap D)$ and $\exists R_1.C \sqcap \exists R_1.D$. Analogously, for general R_1 and R_2 , $\exists R_1 \exists R_2.C$ and $\exists R_2 \exists R_1.C$ are not logically equivalent concepts either. Note that in this last case we can say that $\exists R_1.C$ is in the context of R_2 and not the other way around. We have an explicit dependence of R_1 on R_2 .

Thus, by taking the *nested* way of specifying the *LF* ontology, we force the dependence everywhere. This is not bad, if this is done on the whole set of *actions* and *controls* consistently. This has the advantage of generating a better explanation of how a *control* is subsumed by a particular *action*. Another advantage of the nested style is the fact that it provides a more automatic way of rendering NL sentences into *LF*, for the sequence of modifiers is already in the sentence itself. However, as anyone can observe, the use of passive voice, indirect styles of speech and similar features that Natural Languages display, might force us to consider the advantages of the *nested* style as *apparent* advantages. In fact, the first experiments conducted

⁴Modal forms must be distilled, since they are in general redundant.

⁵We believe $NP \neq PSPACE$.

with our knowledge base (KB) were based on the use of the *nested* style. This style often allowed one to visually check subsumption of a *control* by an *action* by simply placing both descriptions side by side. Such descriptions were, in most cases, of the same form when considering the replacements induced by the *IS* ontology. For example, from *Linux* \sqsubseteq *System* and *Execute* \sqsubseteq *Manage* we have:

$$\begin{aligned} & \exists \text{hasVerb.}(\text{Execute} \sqcap \exists \text{hasTheme.} \text{Linux}) \\ & \sqcap \exists \text{hasPurpose.}(\exists \text{hasVerb.} \text{Update} \sqcap \exists \text{hasTheme.} \text{Linux}) \\ & \sqsubseteq \\ & \exists \text{hasVerb.}(\text{Manage} \sqcap \exists \text{hasTheme.} \text{System}) \\ & \sqcap \exists \text{hasPurpose.}(\exists \text{hasVerb.} \text{Update} \sqcap \exists \text{hasTheme.} \text{System}) \end{aligned}$$

However, when the ontology became bigger (more than 100 *controls*, *circa* 5000 concepts and 27 roles), the classification task (by means of subsumption testing) became extremely heavy, taking in extreme cases more than a couple of hours. Because of this we have refactored the *LF* ontology into a flat style. Knowing the tableau-based proof method used by the Pellet DL reasoner, we expected a decrease in the time spent on the classification of concepts. We turn to this discussion after the following examples on the need for additional axioms.

Example 4.1. Suppose the organization has deployed an action with the following description:

Configure the “type” parameter with the value “Nt5DS” in the [Windows OS].

The nested and flat logical form expressions that represent such action are the concepts defined by:

$$\begin{aligned} \text{Action1Nested} \equiv & \\ & 1 \exists \text{hasVerb.}(\text{Configure} \sqcap \\ & 2 \exists \text{hasTheme.}(\text{TypeParameter} \sqcap \\ & 3 \exists \text{hasValue.}(\text{Nt5DS})) \sqcap \\ & 4 \exists \text{hasLocation.} \text{Windows}) \end{aligned}$$

$$\begin{aligned} \text{Action1Flat} \equiv & \\ & 1 \exists \text{hasVerb.} \text{Configure} \sqcap \\ & 2 \exists \text{hasTheme.}(\text{TypeParameter} \sqcap \\ & 3 \exists \text{hasValue.}(\text{Nt5DS})) \sqcap \\ & 4 \exists \text{hasLocation.} \text{Windows} \end{aligned}$$

Now consider the organization has deployed a security control with the following description:

The Registry “type” parameter must be configured with the value “Nt5DS”.

The nested and flat logical form expressions that represent this control are the concepts defined by:

$$\begin{aligned} \text{Control1Nested} \equiv & \\ & 1 \exists \text{hasVerb.}(\text{Configure} \sqcap \\ & 2 \exists \text{hasTheme.}(\text{TypeParameter} \sqcap \\ & 3 \exists \text{hasValue.}(\text{Nt5DS} \sqcap \\ & 4 \exists \text{hasPossessor.} \text{WindowsRegistry}))) \end{aligned}$$

$$\begin{aligned} \text{Control1Flat} \equiv & \\ & 1 \exists \text{hasVerb.} \text{Configure} \sqcap \\ & 2 \exists \text{hasTheme.}(\text{TypeParameter} \sqcap \\ & 3 \exists \text{hasValue.}(\text{Nt5DS})) \sqcap \\ & 4 \exists \text{hasPossessor.} \text{WindowsRegistry} \end{aligned}$$

To prove the compliance of our structures, we need to add axioms that correlate the location of the Windows parameter with the Windows Registry. As we already said, the nested approach is more complicated than flat approach even in the used axioms.

The following axioms are necessary for the nested formalization.

$$\text{WindowsTypeParam} \equiv \text{RegWindowsTypeParam}$$

$$\begin{aligned} \text{WindowsTypeParam} \equiv & \\ & 1 \exists \text{hasTheme.}(\text{TypeParameter} \sqcap \\ & 2 \exists \text{hasValue.}(\text{Nt5DS})) \sqcap \\ & 3 \exists \text{hasLocation.} \text{Windows} \end{aligned}$$

$$\begin{aligned} \text{RegWindowsTypeParam} \equiv & \\ & 1 \exists \text{hasTheme.}(\text{TypeParameter} \sqcap \\ & 2 \exists \text{hasValue.}(\text{Nt5DS} \sqcap \\ & 3 \exists \text{hasPossessor.} \text{WindowsRegistry})) \end{aligned}$$

The following axioms are necessary for the flat formalization.

$$\text{WindowsLocation} \equiv \text{WindowsRegPossessor}$$

$$\text{WindowsLocation} \equiv \exists \text{hasLocation.} \text{Windows}$$

$$\text{WindowsRegPossessor} \equiv \exists \text{hasPossessor.} \text{WindowsRegistry}$$

In the following example, we illustrate that some representations do not need additional axioms.

Example 4.2. Suppose the following action is part of the organization’s security police:

Set the Date and Time of [system xyz].

The nested and flat logical form expressions that represent such action are the concepts defined by:

$$\begin{aligned} \text{Action2-Nested} \equiv & \\ & 1 \exists \text{hasVerb.}(\text{Set} \sqcap \\ & 2 \exists \text{hasTheme.}(\text{DateTime} \sqcap \\ & 3 \exists \text{hasLocation.}(\text{System}))) \end{aligned}$$

$$\begin{aligned} \text{Action2-Flat} \equiv & \\ & 1 \exists \text{hasVerb.} \text{Set} \sqcap \\ & 2 \exists \text{hasTheme.} \text{DateTime} \sqcap \\ & 3 \exists \text{hasLocation.} \text{System} \end{aligned}$$

Suppose the organization has deployed a security control with the following description:

The date and time of Linux Red Hat must be configured.

The nested and flat logical form expressions that represent such control are the concepts defined by:

$$\begin{aligned} \text{Control2-Nested} \equiv & \\ & 1 \exists \text{hasVerb.}(\text{Configure} \sqcap \\ & 2 \exists \text{hasTheme.}(\text{DateTime} \sqcap \\ & 3 \exists \text{hasLocation.}(\text{LinuxRedHat}))) \end{aligned}$$

$$\begin{aligned} \text{Control2-Flat} \equiv & \\ & 1 \exists \text{hasVerb.} \text{Configure} \sqcap \\ & 2 \exists \text{hasTheme.} \text{DateTime} \sqcap \\ & 3 \exists \text{hasLocation.} \text{LinuxRedHat} \end{aligned}$$

In this example, there was no need for additional axioms to prove compliance of this control with this action, since we have *Set* \equiv *Configure* and *LinuxRedHat* \sqsubseteq *OperatingSystem* \sqsubseteq *System* in the *ISKB*.

The next example illustrates the need for a more sophisticated set of axioms in order to show that an action subsumes a control or a set of controls.

Example 4.3. Suppose the following action is part of the organization’s security policy:

Define TCP port with a non-default value for the execution of [xyz service].

The nested and flat logical form expressions that represent this action are the concepts defined by:

$$\begin{aligned} \text{Action3-Nested} \equiv & \\ & 1 \exists \text{hasVerb.}(\text{Define} \sqcap \\ & 2 \exists \text{hasTheme.} \text{TCPPort} \sqcap \\ & 3 \exists \text{hasValue.}(\text{TCPNonDefaultValue})) \sqcap \\ & 4 \exists \text{hasPurpose.}(\text{Execute} \sqcap \\ & 5 \exists \text{hasTheme.} \text{SoftwareService})) \end{aligned}$$

Action3-Flat \equiv
 1 $\exists hasVerb.Define \sqcap$
 2 $\exists hasTheme.TCPPort \sqcap$
 3 $\exists hasValue.(TCPNonDefaultValue) \sqcap$
 4 $\exists hasPurpose.(Execute \sqcap$
 5 $\exists hasTheme.SoftwareService)$

Now suppose the organization has deployed a security control with the following description:

Apache Tomcat must be executed in a non-default port and in a port not allocated to reserved services.

The nested and flat logical form expressions that represent such security control are the concepts defined by:

Control3-Nested \equiv
 1 $\exists hasVerb.(Execute \sqcap$
 2 $\exists hasTheme.(ApacheTomcat \sqcap$
 3 $\exists hasLocation.(TCPPort \sqcap$
 4 $\exists hasValue.(TCPNotReservedServicePort) \sqcap$
 5 $\exists hasValue.(TCPNonDefaultValue)))$

Control3-Flat \equiv
 1 $\exists hasVerb.(Execute \sqcap$
 2 $\exists hasTheme.(ApacheTomcat \sqcap$
 3 $\exists hasLocation.(TCPPort \sqcap$
 4 $\exists hasValue.(TCPNotReservedServicePort) \sqcap$
 5 $\exists hasValue.(TCPNonDefaultValue)))$

In any case, the structure of the control is quite different from the structure of the action. In fact, the abstraction level of the action seems to be higher. However, taking into account the meaning of both sentences, we can see that the subsumption would hold if we established that “To define some X in a way Z in order to execute some Y ” is subsumed by “To execute some Y in a way Z (different from the default) running in location X ”. In our specific case, this is expressed by the following axioms:

RightHand-on-Defining \equiv
 1 $\exists hasVerb.Define \sqcap$
 2 $\exists hasTheme.TCPPort \sqcap$
 3 $\exists hasValue.TCPNonDefaultValue \sqcap$
 4 $\exists hasPurpose.(Execute \sqcap$
 5 $\exists hasTheme.SoftwareService)$

LeftHand-on-Defining \equiv
 1 $\exists hasVerb.Execute \sqcap$
 2 $\exists hasTheme.SoftwareService \sqcap$
 3 $\exists hasLocation.(TCPPort \sqcap$
 4 $\exists hasValue.(TCPNotReservedServicePort \sqcap$
 5 $\exists hasValue.TCPNonDefaultValue))$

LeftHand-on-Defining \sqsubseteq *RightHand-on-Defining*

The above axiom, together with the following axiom belonging to the ISKB, finally prove the required compliance between the control (flat) and the action (flat). For the nested version the axioms are analogous.

ApacheTomCat \sqsubseteq *SoftwareService*

The last thing to note in this section is the difference of performance between the two styles of formalization, nested and flat. In order to compare performances, a small part of the ISKB was extracted. The classified ontology has 46 controls, 8 actions and 10 (additional) axioms. The time spent by **Pellet** was 22.844 seconds, the nested case, and, 4.734 seconds, the flat case. These experiments were executed on the same machine, a Pentium IV with 1 GB RAM, under similar conditions (no other application was running together with **Pellet**, but **Protege** itself).

5 Conclusion

In section 4, we have shown how two stylistically different but homomorphic⁶ DL representations for *logical forms* (*LF*) can exhibit quite different performances when used for automatic verification of compliance. The initial set of *LF* attributes had 27 different roles, and it was designed with the explicit purpose of covering almost linguistic aspect of a phrasal utterance in the IS domain. But at the beginning of the studies, it was realized that a smaller set would be enough. A subset of the initial set, containing only 8 *LF* attributes, was defined. The *actions* that are shown in this article already use only 8 *LF* attributes. The current stage of the IS ontology has been tuned according to the main observations on under-specification (lack of essential formalization, as for example the axioms shown in section 4), and according to performance.

During the development of this case study, besides the abovementioned conclusions, a methodological conclusion concerning the interaction with the specialist is worth noting: the tools must be designed to serve the needs of those who have asked for the tool. In this sense, we plan to add to our architecture and set of tools a proof explanation generator to allow the specialist to analyze the validation at almost the same level of abstraction as the specification (i.e., the ontology).

Last, but not least, it is important to note that the overall approach shown here can, in principle, also be applied to any other domain with similar features, especially with a body of knowledge written in adequate NL form.

References

- Allen, J. (1995), *Natural Language Understanding*, 2nd edn, Benjamin Cummings.
- Baader, F., Calvanese, D., McGuinness, D., Nardi, D. & Patel-Schneider, P., eds (2003), *The Description Logic Handbook*, Cambridge University Press.
- Caralli, R. A. (2004), ‘Managing for enterprise security’, Technical Note CMU/SEI-2004-TN-046. Available at <http://www.sei.cmu.edu/publications/documents/04.reports/04tn046.html>.
- CERT/CC (2007), ‘Incident notes’. http://www.cert.org/incident_notes/.
- Dean, M. & Schreiber, G. (2004), ‘OWL Web Ontology Language Reference’, <http://www.w3.org/TR/owl-ref/>.
- do Amaral, F. N., Bazílio, C., da Silva, G. M. H., Rademaker, A. & Haeusler, E. H. (2006), An ontology-based approach to the formalization of information security policies, in ‘VORTE - Workshop on Vocabularies, Ontologies, and Rules for the Enterprise’, Hong Kong.
- Gruber, T. R. (1993), Towards principles for the design of ontologies used for knowledge sharing, in N. Guarino & R. Poli, eds, ‘Formal Ontology in Conceptual Analysis and Knowledge Representation’, Kluwer Academic Publishers.
- ISO (2005a), *ANSI/NISO Z39.19-2005, Guidelines for the Construction, Format, and Management of Monolingual Controlled Vocabularies*, NISO.
- ISO (2005b), *BS ISO/IEC 27001 Stand Alone*, ISO/IEC.
- Stanford University (2006), ‘The Protégé Ontology Editor and Knowledge Acquisition System’, <http://protege.stanford.edu>.

⁶This means that there is a function mapping one style into the other which preserves the subsumption relationship.