An Ontology-based Approach to the Formalization of Information Security Policies

Fernando Náufel do Amaral
TecMF, DI, PUC-Rio, Brazil
DICC, IME, UERJ, Brazil
Email: fnaufel@inf.puc-rio.br
fnaufel@ime.uerj.br

Carlos Bazílio
Geiza Maria Hamazaki da Silva
Alexandre Rademaker
Edward Hermann Haeusler
TecMF, DI, PUC-Rio, Brazil
Email: {bazilio, hamazaki, arademaker, hermann}@inf.puc-rio.br

***Abstract*— We present the structure of an ontology for Information Security (IS) and discuss a paradigm whereby it can be used to extract knowledge from natural language texts such as IS standards, security policies and security control descriptions. Besides providing a vocabulary for the IS domain, the proposed ontology stores logical forms corresponding to statements in the text, as well as a set of axioms used for inference in description logic (DL). We also describe a tool to provide automatic support for the formalization process.**

I. Introduction

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 paper discusses the principles involved in an ontology-based approach to the formalization of normative texts in the domain of Information Security (IS), such as security policies defined by organizations.

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 process 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 tools to assist them in these activities.

In this paper, we discuss the use of tools and techniques from the fields of natural language understanding, description logics and ontologies to formalize (and extract knowledge from) natural-language texts. We argue that these tools and techniques can be useful not only in the semi-automatic formulation of security policies, but also in the *validation* of security controls against the policies which the controls are supposed to implement. It should be noted, however, that we do not touch on issues related to the extraction or the deployment of technical controls from the actions in the policies.

The remainder of this paper is organized as follows: Sect. II presents an overview of our proposed approach; Sect. III provides some of the necessary background on natural language understanding, description logics and ontologies; Sect. IV details the IS ontology on which our approach is based; Sect. V illustrates our ideas with a simple example; Sect. VI considers the capabilities of an integrated tool to assist in the formalization of texts; Sect. VII discusses related work and offers our concluding remarks.

II. Formal Specification of Security Policies

At the center of our proposal is the idea that defining formal objects to represent the actions in security policies can be beneficial in more than one way: from the point of view of documentation, formalized policies are more precise than their informal counterparts; in what concerns communication, formalized policies may be automatically converted to whatever presentation formats are convenient on a given occasion (hypertext, graphs, diagrams etc.); as for adequacy, the fact that formalized policies have a well-defined semantics allows one to check them for logical consistency; furthermore, the relationships between the actions of a formalized policy can

be explored (so as to detect redundancy, for example); finally, if low-level security controls are also formally specified, each action in the policy can be automatically associated (via logical inference) to the set of controls that implement it.

As for the relationships between IS standards, actions and controls, we envision two paradigms for the application of our formal approach: the *top-down* paradigm starts with the analysis of IS standards (which can also be made formal through the use of techniques similar to the ones discussed in this paper) in order to generate the actions in a policy, which in turn can be refined to produce sets of (descriptions of) technical controls; the *bottom-up* paradigm presupposes the existence of a (formal or informal) database of controls, which are analyzed and aggregated into actions; the resulting policy can be checked for compliance with the standards adopted by the organization, and the preexisting control database can then be checked for consistency and completeness with respect to the adopted standards.

Our proposed approach consists of the following elements, to be detailed in the subsequent sections:

- Actions are represented at the *logical form* level, a concept from the area of natural language understanding [1]. Basically, logical forms are constructs in some suitable formalism used to represent the context-independent semantics of natural language utterances. Conceptual graphs [2] are frequently used as such a formalism. The logical forms employed in our approach somewhat resemble conceptual graphs, but we also draw from other sources, as described in Sect. III-A below.
- The inference capabilities of the proposed framework are based on a description logic (DL) [3]. 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 provided in Sect. III-B below.
- Ontologies [4] serve as the unifying structure for the above two elements. As briefly described in Sect. III-C, an ontology consists of concepts, properties and logical expressions denoting constraints that hold between these concepts and properties. In our approach, actions in logical form and axioms about them are expressed in terms of these concepts, properties and constraints. One language for representing ontologies is OWL DL [5], which can be translated in a straightforward way to the language used by DL reasoners, providing for an easy interface between the ontology and the inference services of our framework.

Fig. 1 depicts these elements.

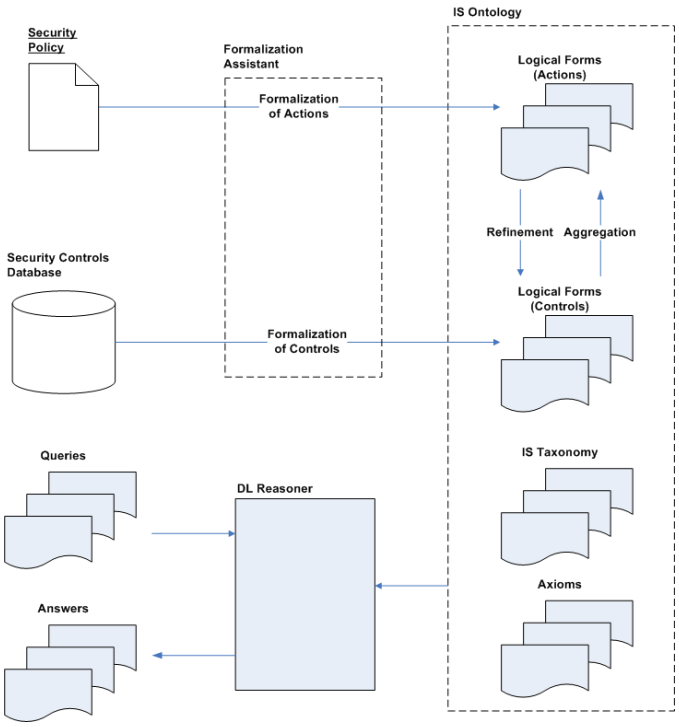

Fig. 1. Elements of our ontology-based approach

III. Background

This section provides a very brief introduction to natural language understanding techniques, description logics and ontologies. Potential readers of this paper come from widely varying backgrounds, so readers already familiar with these areas may find the exposition overly superficial and simplistic at places.

A. Natural Language Understanding

Natural language understanding involves several forms of knowledge, such as phonology, morphology, syntax, semantics, pragmatics, discourse knowledge and world knowledge. Since the 1960's, a host of techniques have been created to deal with the problems posed by each of these aspects of natural language analysis.

As mentioned, we propose a formalization assistant to help the user extract the semantics of actions in security policies. Since the input texts are in written form, phonology plays no role here; as for morphology and syntax, the proposed tool (being an interactive assistant) provides only very rudimentary resources for tokenization and sentence splitting, relying on the user for the rest.

Most relevant to our application are problems related to the extraction and representation of *semantics* (i.e., the context-independent meaning of natural language sentences and texts). As described later on in the paper, the meaning of an action in the policy must be converted to a *logical form* and stored in our IS ontology.

We say we are concerned with the *context-independent* meaning of the actions because the formalization tool does not have to worry about pragmatics, discourse knowledge and world knowledge. For example, the tool does not have to verify whether a well-formed and meaningful sentence is relevant to the subject matter of the policy (pragmatics); neither does it have to discover the referents of the pronouns found in a

sentence (discourse analysis).

As for world knowledge, useful information about the IS domain is stored in our ontology and available to the formalization tool, which can (to a certain extent) guide and restrain the user from creating semantically ill-formed logical forms.

The logical form we use is based on *thematic roles* [1]. The idea is that the meaning of a word is *not* so intimately related to its grammatical function in the sentence (e.g., subject, direct object, indirect object, etc.), but rather to the *semantic role* it plays with respect to the verb. One semantic role may manifest itself as different grammatical functions: for example, in the sentences "The manager configures the system" and "The system is configured by the manager", the role of *agent* is played by "the manager", which appears as subject in one sentence and agent (of the passive voice) in the other. Likewise, the role of *theme* (i.e., the entity which is affected by the event) is played by "the system", which functions as direct object in the first sentence and as subject in the second sentence.

Thematic roles are at a higher level of abstraction than grammatical functions. This provides for more general representations of sentences, allowing for a deeper analysis of (the relationships between) the meanings of sentences.

Thematic roles have much to do with the notion of *case relations* (e.g., nominative, genitive, dative, accusative, etc.), but there is no one-to-one mapping between the two notions, as one thematic role can correspond to different cases in different sentences.

Thematic roles also appear in conceptual graphs [2], in the form of *case frames*. We have extended the notion so as to allow thematic roles to be associated to parts of speech other than verbs.

In our approach, we encode logical forms as DL concept expressions. Examples of this encoding and of the thematic roles used in our work are shown in Sect. IV-B and in Sect. V.

B. Description Logics and Ontologies

DL [3] is a name that refers to any of several logical languages commonly used in knowledge representation. Such languages have evolved from knowledge representation schemes from the 1970's, like semantic networks and frames.

In DL, *concept terms* describe classes of individuals in some universe, while *role terms* (also called *properties*) represent binary relations connecting individuals. For our purposes, the syntax of concept terms and properties is defined by the grammar

$$
\begin{array}{rcll}
C, D & \rightarrow & A & \text{(atomic concept)} \\
 & | & \top & \text{(universal concept)} \\
 & | & \bot & \text{(empty concept)} \\
 & | & \neg C & \text{(negation)} \\
 & | & C \sqcap D & \text{(intersection)} \\
 & | & C \sqcup D & \text{(union)} \\
 & | & \forall R.C & \text{(value restriction)} \\
 & | & \exists R.C & \text{(existential quantification)}
\end{array}
$$

where R represents role names (the only kind of role terms). This corresponds to the logical language known as $\mathcal{ALC}$.

One of the usual ways to interpret concept terms is by translating them to set-theoretical expressions. Formally, an interpretation is defined as a mapping $\mathcal{I}$ from concept terms to sets of individuals in a universe Δ and from role terms to binary relations over Δ satisfying the following conditions:

$$
\begin{array}{rcl}
\mathcal{I}(\top) & = & \Delta \\
\mathcal{I}(\bot) & = & \varnothing \\
\mathcal{I}(\neg C) & = & \Delta - \mathcal{I}(C) \\
\mathcal{I}(C \sqcap D) & = & \mathcal{I}(C) \cap \mathcal{I}(D) \\
\mathcal{I}(C \sqcup D) & = & \mathcal{I}(C) \cup \mathcal{I}(D) \\
\mathcal{I}(\forall R.C) & = & \{a \in \Delta \mid \forall b.[(a,b) \in \mathcal{I}(R) \Rightarrow b \in \mathcal{I}(C)]\} \\
\mathcal{I}(\exists R.C) & = & \{a \in \Delta \mid \exists b.[(a,b) \in \mathcal{I}(R) \wedge b \in \mathcal{I}(C)]\}
\end{array}
$$

If an individual a belongs to the set represented by a concept term C, we say that a is *in* C. If a pair (a, b) of individuals are in the binary relation represented by R, we say that b is a *filler* for property R of a.

Informally, a concept term of the form $\forall R.C$ represents the set of all individuals having all fillers for R (if any) in C. A concept term of the form $\exists R.C$ represents the set of all individuals having at least one filler for R in C.

A DL *subsumption* formula, written $C \sqsubseteq D$, represents a statement to the effect that the class denoted by C is contained in the class denoted by D. An equivalence formula $C \equiv D$ represents the statement that C subsumes D and D subsumes C; i.e., both C and D denote the same class.

A *TBox* (for *terminology*) is a collection of DL subsumption and equivalence formulae taken as axioms. The process of *inference* or *reasoning* consists in discovering which DL formulae are logical consequences of a TBox T.

C. Ontologies

For our purposes, an ontology [4] is simply a DL TBox. There certainly are broader definitions of ontologies, where, for example, a concept can be seen as an individual of a higher-order universe, but they will not be necessary in our approach.

The most common use of ontologies is in storing a taxonomy of concepts in a certain domain of knowledge. From a DL point of view, this taxonomy is defined in terms of *inheritance*, which is exactly the subsumption relation between concepts. The concepts and properties in an ontology can be characterized by means of constraints expressed as DL formulae. The reader should consult [3] for numerous examples.

Ontologies are usually defined in OWL [5] (Web Ontology Language). OWL has a sublanguage (OWL DL) whose constructs coincide with those allowed in DL. The present article uses only OWL DL.

An ontology can also contain *annotation* or *metadata properties*. These are properties whose fillers are character strings or other values that serve as documentation. One use of an annotation property in our IS ontology is to store sets of synonyms of words and phrases from the IS domain, as described in Sect. IV-A below. Although technically not part of DL, annotation properties are allowed in OWL DL.

IV. An Ontology for the IS Domain

Our IS ontology is more than a mere vocabulary of IS-related concepts. Its purpose is threefold: first, to store a taxonomy for the IS domain; second, to store the logical forms that represent actions in the organization's security policy (and possibly also the logical forms that represent control descriptions); and third, to store axioms (DL subsumption and equivalence formulae) in order to support inference in DL. We discuss each of these elements below. Of course, these elements (taxonomy, logical forms and axioms) are ultimately stored in the ontology in the form of concepts, properties and DL constraints, as described in the previous section.

A. IS Taxonomy

Besides defining a common vocabulary that users must commit to, this section of the ontology is also meant to aid in the task of extracting knowledge from natural language texts. The idea of having the ontology include resources useful to specific tasks follows recommendations found in [6], for example, which promote that "these semantic resources cannot be universal but should rather be domain- and even task-specific in most cases."

All concepts in this section of the ontology are subsumed by the concept *ISEntity*. Each such concept represents an entity from the IS domain. Most concepts here are named after nouns or verbs, such as *SoftwareClient*, *System*, *Configure*, *Encrypt*, etc.

To aid in the process of knowledge extraction from texts, each concept is associated to a set of synonyms, much in the style of Wordnet's synsets [7]. In our implementation, a concept is linked to its synset through the annotation property *hasSynSet*. Our tool to assist in the formalization of policies, described in Sect. VI below, sets up a dictionary containing the strings that fill this property; whenever the user selects a word or phrase in the text, the tool will search the dictionary and show the concepts associated to the selected word or phrase.

Incidentally, while Wordnet offers a splendid collection of resources for lexical resolution in knowledge extraction, it is not well suited for our needs: for one thing, it does not contain many of the simple terms and collocations frequently used in the IS domain (e.g., "access control"). One other difference is that 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*. Of course, when a noun is not related to any verbs, we represent the noun by its own concept (e.g., *Resource*, *Account*, *System*). This design decision is closely related to a guideline for thesaurus construction contained in [8, Sect. 6.4], which 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. Analogously, adjectives and adverbs which share the same root and have similar senses, like "automatic" and "automatically", are both associated to an adjective concept — in this case, *Automatic*. When formalizing a sentence with the help of our tool, the distinction between noun and verb senses, or between adjective and adverb senses, will be made clear upon inspection of the role played by the concept in the logical form representing the sentence.

B. Statements

This section of our ontology stores the formal representations of the actions that make up a security policy, and possibly the formal representations of the security control descriptions contained in a control database. Recall that actions are statements pertaining to *what* should be done to protect the organization against security threats, while security controls are more concrete statements pertaining to *how* the actions should be implemented. This arrangement suggests that one of the main applications of the formalized statements is the *validation* of actions and controls: one would like to verify, for example, whether all actions are correctly implemented by the controls, or whether every control implements exactly one action. As will be detailed below, these questions can be answered by classifying the ontology and checking which controls are subsumed by which actions.

As mentioned, we store the sentences in actions and controls in *logical form*. In order for validation to be possible, the logical forms of actions and controls must be abstract enough to allow for the detection of information common to high-level and low-level statements, but not so abstract that important details related to the low-level statements may be lost. Our design decisions have been guided by the search for a satisfactory tradeoff with respect to these conflicting requirements. In particular, the following considerations have been made, among others:

- A sentence containing terms from a certain part of speech should yield the same logical form as a paraphrase based on related terms from other parts of speech (e.g., "The manager is responsible for controlling access to the server" and "Access control to the server is the manager's responsibility");
- A sentence in the active voice and an equivalent sentence in the passive voice should generate the same logical form (e.g., "Access to the server should be controlled by the manager" and "The manager should control access to the server");
- Certain sentences in the declarative mood should be considered equivalent to sentences in the imperative mood (e.g., "Access to the server is controlled" and "Control access to the server");
- Certain sentences in the causative voice are subsumed by the corresponding sentence in the active voice (e.g. "Have the manager control access to the server" is subsumed by "Control access to the server").

Some of these requirements are taken care of by the constraints imposed by the logical representation; others are

satisfied via the adopted taxonomy (i.e., the choice of concepts in the section of the ontology discussed in Sect. IV-A above), while still others are enforced by axioms, as described in Sect. IV-C below. In all cases, however, the logical form must be flexible enough to allow this interaction between the taxonomy and the axiom base.

It is perhaps impossible to include in the ontology a precise definition of sentence equivalence that handles all the situations occurring in practice. In the cases that are not covered, we can still rely on the formalization tool (Sect. VI) to provide some degree of guidance by means of examples, tutorials or online help.

We now describe how a statement is formalized in our ontology. We will discuss only actions, but the general principles apply to control descriptions as well. Sect. V below presents a detailed example that illustrates most considerations made here.

An action is represented in the ontology by a subconcept (say, *Action0001*) of the *Action* concept. The text of the action is transformed into DL constraints that serve as necessary and sufficient conditions defining the concept associated to the action.

The most important element in an action is probably a verb; so, the property *hasVerb* is provided in order to relate the action concept to the appropriate verb concepts. For example, if the text of the action represented by *Action0001* is "The manager should control access to the server", the condition

$$\exists \mathit{hasVerb}.\mathit{ControlAccess}$$

is added as a condition defining *Action0001*. Note, incidentally, that "control access" is a collocation frequent enough in the IS domain to merit its own atomic concept.

The agent of the verb, when present, is represented by the *hasAgent* property. In the example, the condition defining *Action0001* is strengthened to

$$\exists \mathit{hasVerb}.(\mathit{ControlAccess} \sqcap \exists \mathit{hasAgent}.\mathit{Manager})$$

Finally, the theme of the verb is represented by the *hasTheme* property, yielding the logical form:

$$\exists \mathit{hasVerb}.(\mathit{ControlAccess} \sqcap \exists \mathit{hasAgent}.\mathit{Manager} \sqcap \exists \mathit{hasTheme}.\mathit{Server})$$

Note that the result of the formalization is a concept definition. No instances of any concepts are created. In other words, our ontology is never populated with individuals, but rather consists of a TBox (i.e., a set of terminological axioms, as described in Sect. III-B above). It is worth mentioning that some implementations of DL reasoners currently available do not work well when individuals are present in the ontology, being only equipped to receive TBoxes as input.

Of course, the effectiveness of this formalization scheme depends on the available set of properties (thematic roles). It is a common belief in the natural language understanding literature [1] that an adequate set of thematic roles for most situations need contain no more than some two dozen roles. Some of the properties representing thematic roles in our ontology (besides the ones already mentioned) are *hasBeneficiary*, *hasExperiencer*, *hasInstrument*, *hasLocation*, *hasManner*, *hasPossessor*, *hasPurpose*, *hasRecipient*, and *hasState*. This set has been drawn from [1], from the relations present in examples using conceptual graphs [2], and from our own practice. To facilitate the construction of the logical forms by the user, the ontology also keeps the information of which roles are more "naturally" associated to each concept — the so-called *inner roles* of the concept. The concept *Give*, for example, has the property *hasRecipient* as one of its inner roles.

Our logical forms can be represented by rooted trees. This will turn out to be convenient when presenting formalized actions to a user, as discussed in Sect. VI below.

C. Axioms

The equivalences and subsumptions necessary to ensure that sentences with closely related meanings are mapped to the same logical form, as mentioned in Sect. IV-A above, are stored in the ontology in the form of DL axioms. As an example, the axiom that states that "configuring X to achieve Y" is equivalent to "achieving Y" is encoded as the DL equivalence formula

$$\exists \mathit{hasVerb}.(\mathit{Configure} \sqcap \exists \mathit{hasTheme}.X \sqcap \exists \mathit{hasPurpose}.Y) \equiv \exists \mathit{hasVerb}.Y$$

To be more precise, this is a DL *schema* that represents a (potentially infinite) number of DL formulae, and the inference algorithm must be able to handle this kind of expression.

V. A Simple Example

Suppose the following action is part of the organization's security policy:

> *Configure every system to encrypt connections used for remote access to the system.*

The logical form representing this action is the concept *Action0002*, defined as

$$\begin{array}{ll} & \mathit{Action0002} \equiv \\ 1 & \exists \mathit{hasVerb}.(Configure \sqcap \\ 2 & \quad \exists hasTheme.System \sqcap \\ 3 & \quad \exists hasPurpose.(Encrypt \sqcap \\ 4 & \quad\quad \exists hasTheme.(NetworkConnect \sqcap \\ 5 & \quad\quad\quad \exists isInstrumentOf.(AccessRemotely \sqcap \\ 6 & \quad\quad\quad\quad \exists hasTheme.System)))) \end{array}$$

where lines are numbered for reference. Note that the quantifier "every" does not appear in the logical form. We establish the convention that all concepts are universally quantified by default. This stands in contrast with the semantics of conceptual graphs [2], where the default is existential quantification. Note also that the filler of property *hasPurpose* in line 3 is an embedded sentence ("encrypt connections used for remote access to the system"). In line 4, concept *NetworkConnect*, named after the verb, is used to represent the noun phrase "network connection". In line 5, the characterization of the

connections in question can be read as "those that serve as instruments of remote access to the system".

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

> *Network traffic for the remote administration of the Netware server must be encrypted using SSL.*

Being a control description, this statement is more concrete than the action mentioned above. It can be formalized as

$$
\begin{aligned}
&\mathit{Control}0001 \equiv \\
1\quad &\exists \mathit{hasVerb}.(\mathit{Encrypt} \sqcap \\
2\quad &\quad \exists \mathit{hasTheme}.\mathit{NetworkTraffic} \sqcap \\
3\quad &\quad \exists \mathit{hasInstrument}.\mathit{SSL} \sqcap \\
4\quad &\quad \exists \mathit{hasPurpose}.(\mathit{AdministerRemotely} \sqcap \\
5\quad &\quad\quad \exists \mathit{hasTheme}.\mathit{NetwareServer}))
\end{aligned}
$$

The sentence has been "converted" to the active voice as a (desirable) artifact of the formalization.

We would like to infer that *Control0001* actually implements *Action0002*. This is done by proving that the concept representing the control is subsumed by the concept representing the action, i.e., that the DL formula

$$\mathit{Control0001} \sqsubseteq \mathit{Action0002}$$

is provable. This can indeed be done by a reasoner, since the following axioms are contained in the ontology:

- The schema given in Sect. IV-C stating that "configuring X to achieve Y" is equivalent to "achieving Y";
- The self-evident subsumption formula (actually, part of the taxonomy)

$$\mathit{AdministerRemotely} \sqsubseteq \mathit{AccessRemotely}$$

- The self-evident subsumption formula (actually, part of the taxonomy)

$$\mathit{NetwareServer} \sqsubseteq \mathit{System}$$

- The equivalence

$$\mathit{Encrypt} \sqcap \exists \mathit{hasTheme}.\mathit{NetworkConnect} \equiv \mathit{Encrypt} \sqcap \exists \mathit{hasTheme}.\mathit{NetworkTraffic}$$

  stating that encrypting a network connection is synonymous with encrypting the network traffic.

Note that the presence of additional information in the formalized control description (namely, in line 3, that the encryption is done via SSL) does not interfere with the truth of $\mathit{Control0001} \sqsubseteq \mathit{Action0002}$. In fact, the logical form corresponding to a control description will usually mention more properties (in this example, *hasInstrument*) than the formalized action, since control descriptions are more detailed than actions.

VI. A Formalization Assistant

We are currently developing a tool to provide automatic support for the formalization of actions and control descriptions. This section briefly describes some of features of the tool and the principles involved.

Many tools have been proposed for the formalization and/or annotation of natural language texts (see Sect. VII below), with varying degrees of automation. We stress that the tool we are developing is meant to be a *formalization assistant*, i.e., a user-friendly front end providing convenient means to browse our IS ontology and to scan a natural language text in order to associate words and phrases in the text to the concepts and properties that will make up the logical forms. During this process, the tool suggests different context-sensitive choices to the user, based on the information contained both in the ontology and in the text. Moreover, the tool's user interface is designed so as to hide the DL syntax and the underlying formal mechanisms from the user unfamiliar with logical notation and inference.

Fig. 2 shows a snapshot of one of the perspectives of the user interface, whose components we now describe.

- The Concepts and Properties frames on the left-hand side allow the user to browse the IS taxonomy contained in our ontology, obtaining not only a compact view of the inheritance relationship, but also useful information on the concepts and properties, such as usage hints, associated synsets, occurrences in the axioms, etc., in the form of tool tips or pop-up windows.
- The Text frame in the middle is a simple editor holding the text or texts to be formalized. The user has the option of constructing the text incrementally, alternating the formulation of sentences with their formalization, or of importing a fully-written text (e.g., a document describing a security policy) in read-only mode and then proceeding to its formalization. By selecting a word or phrase in the text and clicking a mouse button, the user can see the associated concepts and properties in the taxonomy; i.e., those containing in their synsets the selected text (or text similar to the selected text, as the tool provides fuzzy search in order to handle morphological inflections and even spelling mistakes).
- The Logical Form frame presents a tree-based, user-friendly rendition of the DL formulae corresponding to natural language sentences that have already been formalized or are undergoing formalization. Branches of the trees can be conveniently collapsed or expanded according to the user's purposes. The user can choose between having the labels in the trees contain the names of the concepts in the taxonomy (e.g., *NetworkConnect*, *AccessRemotely*) or the corresponding strings extracted from the text (e.g., "connections", "remote access"). Whichever the choice, when the user selects a term in the tree, the associated string is highlighted in the text.

Other features (not shown in the snapshot) include tree-based views of the DL axioms contained in the ontology and support for collaborative work, such as change logs and versioning resources. Depending on the privileges of the user, the formalization assistant can allow the IS taxonomy or the axioms to be modified. Such modifications, however, are stored separately so they can await revision and approval by

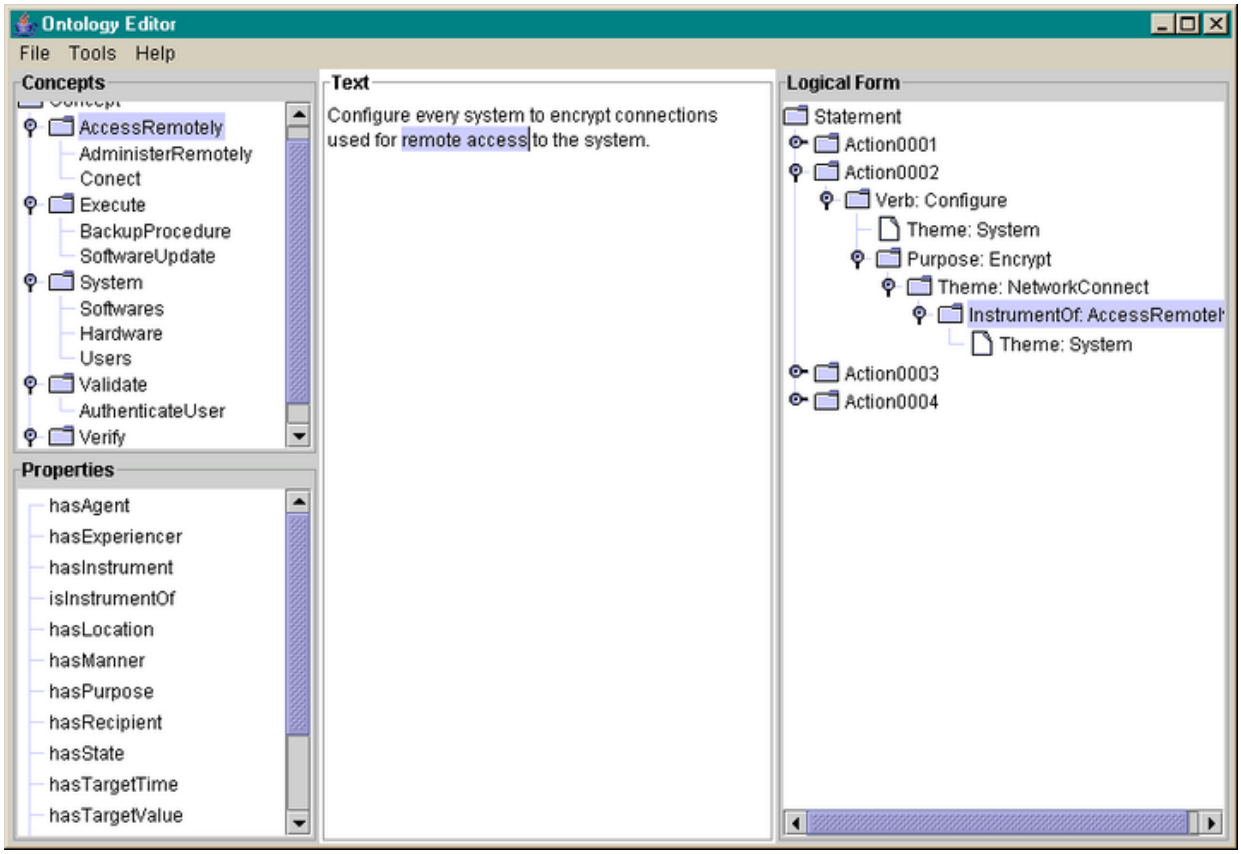

Fig. 2. A perspective of the formalization assistant

users possessing higher privileges (e.g., domain experts or knowledge engineers).

We have decided in favor of a highly interactive formalization assistant (as opposed to a tool with a greater degree of automation) in order to maximize the use of human expertise and in order to simplify the design and the implementation of the tool.

IS policies contain very important and sensitive information, and few, if any, security experts would agree to letting such a policy be formalized in a fully automatic process, without human intervention. Even for formalization and annotation activities in other, less crucial domains, there are arguments in favor of our human-centered approach, as can be found in [6]: "*text cannot be the only semantic resource* of an annotation process (...) *but human expertise is required to add either more expertise or some common sense knowledge*" (italics in the original).

It could be argued that general-purpose ontology construction and browsing programs, such as Protégé [9], could be used in the formalization process, but such programs do not hide the underlying formalisms from the user, although, depending on the architecture, dedicated plug-ins could be developed.

From the viewpoint of system design and implementation, note that our formalization assistant does not need any complicated algorithms for handling parsing, word sense disambiguation, named entity recognition, or anaphora resolution, for example. Given an adequate interface, all these tasks can be easily and effortlessly carried out — if necessary — by the user. Of course, the texts to be formalized in the IS domain tend to be rather objective and linguistically simple, so this is not really a burden on the user.

The software architecture of our formalization assistant is such that the underlying IS ontology can be replaced with other ontologies, perhaps for different domains, provided some architectural conventions are obeyed. This gives a nice degree of genericity and portability to our tool, endowing it with great potential for reuse.

VII. Conclusions

We are currently defining and studying the properties of natural deduction and sequent-based inference engines for DL. Traditionally, inference in DL has been based on tableaux [3], but we believe that we have much to gain by turning to other, non-refutation-based deductive calculi. One of our goals is to provide the user with natural language explanations extracted from DL proofs and failed proof attempts concerning the relationships between actions and control descriptions in logical form; natural deduction and sequents seem to be more suitable for that than tableaux [10].

One way to profit from significant advances in the architecture of natural language processing tools is to continue the development of our formalization assistant within the

context of GATE [11], which also offers ontology-related resources [12].

We are aware of a number of articles describing IS-related ontology-based models such as [13] and policy representation and reasoning languages — see [14] for a comparative study. Some of the approaches (see, e.g. [15]), like ours, are strongly based on natural language processing techniques. Incidentally, ontology-based text annotation [16] and ontology extraction from text [17] are currently very busy areas of study, given their importance for the deployment of the Semantic Web.

As our research project is still in its initial stage, it is perhaps too early to draw definitive conclusions from a comparison with these other approaches. There are quite a few different directions our work may follow; input from academic work inside and outside our group, and feedback from our industrial partners are major influences in this process.

REFERENCES

[1] J. Allen, *Natural Language Understanding*, 2nd ed. Benjamin Cummings, 1995.

[2] J. F. Sowa, *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Brooks Cole Publishing Co., 2000.

[3] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, Eds., *The Description Logic Handbook*. Cambridge University Press, 2003.

[4] T. R. Gruber, "Towards principles for the design of ontologies used for knowledge sharing," in *Formal Ontology in Conceptual Analysis and Knowledge Representation*, N. Guarino and R. Poli, Eds. Kluwer Academic Publishers, 1993.

[5] M. Dean and G. Schreiber, "OWL Web Ontology Language Reference," http://www.w3.org/TR/owl-ref/, Feb. 2004.

[6] N. Aussenac-Gilles, "Supervised text analysis for ontology and terminology engineering," in *Proc. of the Dagstuhl Seminar on Machine Learning for the Semantic Web*, N. Kushmerick, F. Ciravegna, A. Doan, C. Knoblock, and S. Staab, Eds., 2005, available at http://www.smi.ucd.ie/Dagstuhl-MLSW/proceedings/.

[7] C. Fellbaum, Ed., *Wordnet: an Electronic Lexical Database*. MIT Press, 1998.

[8] *ANSI/NISO Z39.19-2005, Guidelines for the Construction, Format, and Management of Monolingual Controlled Vocabularies*, NISO, 2005.

[9] Stanford University, "The Protégé Ontology Editor and Knowledge Acquisition System," http://protege.stanford.edu, 2006.

[10] D. A. S. e Oliveira, C. S. de Souza, and E. H. Haeusler, "Structured Argument Generation in a Logic-Based KB-System," in *Logic, Language and Computation*, ser. CSLI Lecture Notes, L. S. Moss, J. Ginzburg, and M. de Rijke, Eds. CSLI, 1999, vol. 2, no. 96, pp. 237–265.

[11] H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan, "GATE: A framework and graphical development environment for robust NLP tools and applications," in *Proc. of the 40th Anniversary Meeting of the Association for Computational Linguistics*, 2002.

[12] K. Bontcheva, V. Tablan, D. Maynard, and H. Cunningham, "Evolving GATE to Meet New Challenges in Language Engineering," *Natural Language Engineering*, vol. 10, no. 3/4, pp. 349–373, 2004.

[13] B. Tsoumas, S. Dritsas, and D. Gritzalis, "An Ontology-Based Approach to Information Systems Security Management," in *Proc. of the Third International Workshop on Mathematical Models, Architectures and Protocols for Computer Network Security*, 2005, pp. 151–164.

[14] G. Tonti, J. M. Bradshaw, R. Jeffers, R. Montanari, N. Suri, and A. Uszok, "Semantic Web Languages for Policy Representation and Reasoning: A Comparison of KAoS, Rei, and Ponder," in *Proc. of the International Semantic Web Conference*, 2003, pp. 419–437.

[15] V. Raskin, C. Hempelmann, K. E. Triezenberg, and S. Nirenburg, "Ontology in information security: a useful theoretical foundation and methodological tool," in *Proc. of the New Security Paradigms Workshop*, 2001, pp. 53–59.

[16] P. Cimiano and S. Handschuh, "Ontology-based linguistic annotation," in *Proc. of the ACL 2003 workshop on Linguistic annotation*, 2003, pp. 14–21.

[17] P. Buitelaar, D. Olejnik, and M. Sintek, "A Protégé Plug-In for Ontology Extraction from Text Based on Linguistic Analysis," in *Proc. of the First European Semantic Web Symposium*, 2004, pp. 31–44.