**SEVENTH FRAMEWORK PROGRAMME**
**Challenge 1**
**Information and Communication Technologies**

**Trusted Architecture for Securely Shared Services**

| | |
|---|---|
| **Document Type:** | Deliverable |
| **Title:** | **Controlled Natural Language Policies** |
| **Work Package:** | WP6 |
| **Deliverable Nr:** | D6.3 |
| **Dissemination:** | Public |
| **Preparation Date:** | 25 June, 2011 |
| **Version:** | 2.2 |

## The TAS3 Consortium

| | Beneficiary Name | Country | Short | Role |
|---|---|---|---|---|
| 1 | KU Leuven | BE | KUL | Coordinator |
| 2 | Synergetics NV/SA | BE | SYN | Partner |
| 3 | University of Kent | UK | KENT | Partner |
| 4 | University of Karlsruhe | DE | KARL | Partner |
| 5 | Technische Universiteit Eindhoven | NL | TUE | Partner |
| 6 | CNR/ISTI | IT | CNR | Partner |
| 7 | University of Koblenz-Landau | DE | UNIKOL | Partner |
| 8 | Vrije Universiteit Brussel | BE | VUB | Partner |
| 9 | University of Zaragoza | ES | UNIZAR | Partner |
| 10 | University of Nottingham | UK | NOT | Partner |
| 11 | SAP Research | DE | SAP | Project Mgr |
| 12 | EIfEL | FR | EIF | Partner |
| 13 | Intalio | UK | INT | Partner |
| 14 | Risaris | IR | RIS | Partner |
| 15 | Kenteq | NL | KETQ | Partner |
| 16 | Oracle | UK | ORACLE | Partner |
| 17 | Custodix | BE | CUS | Partner |
| 18 | Medisoft | NL | MEDI | Partner |
| 19 | KIT | DE | KARL | Partner |
| 20 | Symlabs | PT | SYM | Partner |

## Contributors

| | Name | Organisation |
|---|---|---|
| 1 | David Chadwick | KENT |
| 2 | Leilei Shi | KENT |

## Reviewers

| | Name | Organisation |
|---|---|---|
| 1 | Dries Pruis | Kenteq |
| 2 | Gang Zhao | VUB |
| 3 | Ioana Ciuciu | VUB |

# Contents

# Executive Summary

Policies are relatively easy for humans to specify in natural language, but are much more difficult for them to specify in computer based languages e.g. XML or ASN.1. Consequently humans usually need some sort of human computer interface (HCI) in order to ease the task of policy specification. Conversely, computers can easily process policies written in computer based languages, but find it much more difficult to parse and understand natural language. The usual solution to this problem is to devise a graphical user interface (GUI) that is relatively easy for humans to use, and that is capable of converting the chosen icons, menu items and entered text strings into the computer based policy language. This procedure was adopted by the authors in a project prior to TAS3, and the result was the PERMIS Policy Editor. However, users still have to learn how to use the GUI, and this can be difficult for them, especially for novice users. Consequently, in TAS3 we wanted to remove as much of this barrier to policy writing as possible, by allowing humans to enter their access control policy in a Controlled Natural Language (CNL). A CNL is a subset of natural language, which removes many ambiguities and complexities of a full natural language, thereby making it much easier for computers to parse and understand, without losing the ease of expression of natural language. Using a CNL is an important step in the journey to full natural language use for policy specification.

This document details the research that was performed in order to allow human users to specify an access control policy in a CNL, in which the human computer interface (HCI) performs the difficult task of conversion from the CNL into the computer based language. This document describes:

- The vocabulary and syntax of the access control policy CNL that was developed for use by the human users (Section 1).
- The underlying, computer language independent, ontology that the HCI uses to store the policy that is being specified by the human user (Section 1.4 and Appendix D).
- The architecture of the HCI (section 2).
- The various CNL tools and packages that were used (section 3) and the modifications that needed to be made to them in order to convert the policy CNL into the underlying policy ontology (section 4 and Appendix A).
- Conversion of the policy ontology into the user's chosen export computer based policy language (section 5). Three export languages are currently supported: XACML, PERMIS XML and RDF, but the modular design of the HCI system allows other export languages to be easily added.
- The graphical user interface that was developed to allow the CNL policy to be input and edited by the human user (section 6).
- The various APIs that are built into the HCI, which will allow future developers to add new features to the CNL HCI (section 7)
- The user trials that were carried out with novice users, in which they were asked to specify an access control policy using both the CNL HCI developed in this project and the PERMIS GUI developed previously (section 8 and

Appendices B and C). The results showed that the CNL HCI was significantly more effective than the GUI, in that both accuracy and completeness improved significantly with the CNL interface.

- Section 9 concludes this document and indicates where future research is still needed

# 1 Policies in Controlled Natural Language

## 1.1 Controlled Natural Language (CNL)

Research has shown [1] that when users are asked, they already intuitively know who should access their resources and who should not. In other words, they already know the access control policy, or a set of rules, that are needed to make correct access control decisions. However, certain specialized knowledge, such as computer science jargon, or machine languages, or complex interfaces, are usually required in order to write these rules as an executable policy for an access control system. It would be much easier for the users if they could express these rules in their own natural language. This document describes the use and enhancements by TAS3 to a technique developed in earlier EC and UK projects [4, 5] which helps users to author their access control policies using a restrictive form of English, called controlled natural language (CNL).

Compared with machine languages such as XML and Java, a big advantage of natural languages is their usability. As the name implies, a natural language is the natural way that humans read, write and think. Users are already familiar with its grammar and semantics, since they have usually learnt it from birth and practice it every day. So natural language, when used as a user interface, would have high effectiveness and a low (or even zero) learning effort.

However, natural languages are not an ideal interface for computers, because of their lack of computability. Most machine languages are defined in a formal way, and are based on mathematical models, so that computability can be guaranteed. Unlike machine languages, natural languages are defined in an informal way, which is often ambiguous and under-specified. Natural Language Processing (NLP) is consequently very hard to perform, due to the complexity, flexibility and ambiguity of natural languages. It is very difficult, if possible at all, to make machines understand the full set of a natural language like a human being can. Hence in the area of natural language processing, it is often easier to limit the grammar and vocabulary to a subset of the whole natural language. This subset is called a Controlled Natural Language (CNL).

A CNL is a sub-language of the corresponding natural language. The grammar and semantics of a CNL are usually the same as those of the (complete) natural language, but sometimes the grammar of a CNL can be slightly different to that of the natural language. It might be looser or stricter, depending upon the purpose and implementation of the application. But it must be very close to the original grammar in order to keep its features as a natural language.

CNL is therefore a compromise between computability and usability. With a restricted vocabulary, grammar and syntax, its semantics and computability can be defined explicitly and unambiguously. This makes it possible for computers to process expressions in the CNL, and understand their corresponding semantics. By sharing the same grammar and semantics as the (complete) natural language, a CNL is intuitive to read and write. This makes a CNL relatively easy to learn, compared with machine programming languages or complex interfaces. Furthermore, no jargon is needed. But due to the nature of the gap between a CNL and its (complete) natural language, some learning effort is still required since many normal sentences which a user might typically use are out of the scope of the CNL. This learning effort though should be minor. The amount of effort needed to learn a CNL depends on the subset it takes from the natural language. A small subset of grammar and vocabulary might reduce the effort needed to learn it, but on the other hand, this limits its capability and/or flexibility of expression. Here, we say “might” because a large subset doesn’t necessarily mean more effort in learning. In fact, a CNL which has a rich subset of grammar and vocabulary that covers virtually all combinations in the complete natural language would require zero effort from users to learn. But this would make it very hard to implement. So defining a CNL involves a balance between its expressive capability, the learning effort of the user, the complexity of the implementation and its feasibility for the task in hand.

### 1.1.1 The RBAC and ABAC Policy Models

In the TAS3 project, we have defined a CNL which is dedicated to the task of authoring access control policies. It currently provides the capability of expressing basic access control policies which follow the role based access control (RBAC) and attribute based access control (ABAC) models. RBAC is an American standard for access controls [2]. ABAC [3] is a more flexible version of RBAC.

The RBAC model has the following features.

Users are assigned to roles e.g. David is a project manager

Permissions are assigned to roles. A permission comprises an action on a resource e.g. a project manager can edit the TAS3 project plan.

Role holders inherit the permissions assigned to their roles e.g. David, a project manager, inherits the permission to edit the TAS3 project plan

Roles may optionally be structured into a role hierarchy. The feature of this hierarchy is that superior roles inherit the permissions of all their subordinate roles in the hierarchy e.g. project managers are superior to team leaders which are superior to project members. Project managers inherit the permissions of both team leaders and project members, whilst team leaders inherit the permissions of project members.

Constraints or conditions may be placed upon when roles are assigned their permissions e.g. project managers can only edit the TAS3 project plan between the hours of 9am and 5pm.

The current version of the software supports 1 to 4 above, but not 5.

In addition to the above features of the standard RBAC model, in order for it to be useful in a federated or distributed environment, there need to be rules for who is allowed to assign which roles to which users. (This is not needed in a centralised RBAC system where it is assumed that there is just one system administrator who assigns all roles to all users.) This leads to the following features:

Users may need to be separated into domains e.g. the domain of the University of Kent.

Each role must have a Source of Authority, being the entity that is trusted by a resource owner (i.e. policy author) to assign this role to users from one or more domains e.g. Danny de Cock is the Source of Authority for the role "TAS3 project member".

The current version of the software supports both of the above features.

The ABAC model has the following additional features to standard RBAC:

Users may be assigned any attribute, not just a role e.g. David is assigned a Visa credit card attribute.

Resources may be identified by their attributes rather than their names e.g. a resource of type project plan, a medical record resource created in 2008.

Obligations may be assigned to permissions, so that when the permission is granted (or denied) the obligation should be carried out e.g. if a user is granted access to File X then notify the security officer about this.

The current version of the software supports 8 and partially supports 9.

### 1.1.2 The CNL for RBAC and ABAC policies

### 1.1.3 Lexicon and Vocabulary

The lexicon for our CNL is the Latin alphabet, the Arabic numerals, English punctuation marks and other printable symbols such as brackets, ampersand etc. i.e. the ASCII printable character set. All nouns, verbs and prepositions in the English vocabulary remain in the vocabulary of the CNL. Furthermore arbitrary strings with space, hyphens or underscores can be used as names. But other symbols or punctuation marks are not allowed to be used in names.

Some words have been reserved as key words (see Table 1). These are used to construct sentences in the CNL policy. All other words have to be properly defined as policy elements by the user, before they can be used in further policy sentences.

| | | | | |
|---|---|---|---|---|
| action | assignment | have | role | which |
| administrator | assign | if | say | who |
| all | be | know | specify | with |
| allocate | call | my | superior | |
| allow | can | only | target | |
| also | condition | parameter | there | |
| anyone | domain | permission | trust | |
| anything | every | permit | type | |
| anywhere | everything | policy | user | |
| as | from | resource | value | |

**Table 1 - Reserved key words**

Unlike most programming languages, plurals of nouns and different tenses of verbs can be recognised as the corresponding key word or defined term.

## 1.1.4 Syntax

The syntax of the policy CNL is summarised below using the BNF notation:

<sentence> ::= <phrase> "be a type of" <class>

<sentence> ::= <phrase> "be" <class>

<sentence> ::= <definedTerm> "have" <definedTerm>

<sentence> ::= <role> "be" <hierarchy> <preposition> <role>

<sentence> ::= <subject> <can> <action> <resource>

<sentence> ::= <subject> <can> <action> <preposition> <resource>

<sentence> ::= <subject> <can> <action> <preposition> <resource> "if" <condition>

<sentence> ::= <administrator> <can> <assign> <role> <preposition> [ "user" <preposition> ] <userDomain>

<sentence> ::= "I trust" <administrator> <preposition> <say> "who" <role> "be"

<class> ::= "action" | "administrator" | "condition" | "parameter" | "permission"| "policy" | "resource" | "role" | "target" | "user" | "user domain" | <definedClass>

<definedClass> ::= <phrase> that has already been defined as a class in a previous sentence

<definedTerm> ::= <class> | <subject> |<action> |<resource> |<administrator> |<condition>

<subject> ::= <user> |<userDomain> | <role>

<user> ::= "user" | <definedUser>

<definedUser> ::= <phrase> that has already been defined as a user in a previous sentence

<userDomain> ::= "user domain" | <definedUserDomain>

<definedUserDomain> ::= <phrase> that has already been defined as a user domain in a previous sentence

<role> ::= "role" | <definedRole>

<definedRole> ::= <phrase> that has already been defined as a role in an earlier sentence

<action> ::= "action" | <definedAction>

<definedAction> ::= <phrase> that has already been defined as an action in an earlier sentence

<resource> ::= “resource” |<definedResource>

<definedResource> ::= <phrase> that has already been defined as a resource in an earlier sentence

<administrator> ::= “administrator” | <definedAdministrator>

<definedAdministrator> ::= <phrase> that has already been defined as an administrator in an earlier sentence

<condition> ::= <phrase>

<hierarchy> ::= “superior” | “subordinate”

<can> ::= “can” | “be allowed to” | “be permitted to” | “be granted to” | “have permission to”

<assign> ::= “assign” | “allocate” | “grant”

<say> ::= “say” | “specify”

<preposition> ::= any preposition from English

<phrase> ::= any noun phrase or verb phrase from English

The parser used to parse the CNL policies matches patterns in a more sophisticated way than the BNF above implies. For example, it matches verbs such as “be” by using all the different tenses and conjugations. Furthermore any phrase is treated as a single policy element providing it does not contain any key word or previously defined policy element. Phrases that do contain key words or other policy elements can be encapsulated in speech marks and then they will be treated as single policy elements. All words and phrases are labelled with POS (Part of Speech) tags, such as noun, verb, and preposition. These POS tags are then used in the pattern matching, as shown in the BNF syntax listed above.

### 1.1.5 Semantics

A policy which is written in the CNL defined above is parsed sentence by sentence. If a sentence fails to parse, the whole sentence is dropped. Hence the policy semantics are defined at the sentence level. Based on the semantics of a sentence, it is categorised into one of three groups, as specified below. Each group is used for the purpose of defining different parts of an access control policy.

#### 1.1.5.1 Policy Elements

Sentences in the first groups are for the general purpose of ontology editing. These sentences are used to describe elements (i.e. roles, actions, resources, sources of authority and user domains) of the policy. Here are examples of each sentence, along with their associated semantics:

<sentence> ::= <phrase> "be a type of" <class>

Printers and Files are types of resources.

Printers are a type of resource.

Printer is a type of resource.

This group of sentences define one or more new classes, and specify the super class at the same time. In the above examples, the *Printer* and *File* class are created as a sub class of *Resource*.

<sentence> ::= <phrase> "be" <class>

Company profile and price list are files.

Read and write are actions.

Manager, staff and guest are roles.

"Resource repository" is a resource.

University of Kent is a user domain.

Bob is an administrator.

Instances of ontology classes are created by these sentences. The class element must have been defined already, before it can be used in sentences that create instances of the class. Examples of newly defined instances are: *company profile, price list, read, write, manager, resource repository, staff, guest, University of Kent and Bob.*

<sentence> ::= <definedTerm> "have" <definedTerm>

Files have read and write.

<sentence> ::= <role> "be" <hierarchy> <preposition> <role>

Managers are superior to staff and guests.

The sentences above are used to define simple relations connecting elements together. The relation can be either named as "have" or, in the case of roles, a hierarchical relationship of superior or subordinate can be defined.

All these sentences are used to define and classify the new policy elements within the underlying policy ontology (see 1.4). These new elements will be classified by the CNL engine as either classes or instances of the pre-existing built-in ontology classes (see 1.4). With these sentences, policy elements, simple relationships and role hierarchies are defined, so that they can be subsequently used in defining the policy rules.

#### 1.1.5.2 Permission assignment rules

Permission assignment rules are the core of an RBAC policy and they assign permissions to roles (or attributes). The semantics of the second group of sentences is to create permission assignment rules in the ontology. Optionally they may also say under what conditions a permission may be assigned, although this feature has not yet been fully implemented. Here are some example sentences that can be used to assign permissions to roles (or attributes):

<sentence> ::= <subject> <can> <action> <resource>

Managers can update "resource repositories".

Guests can read company profile.

<sentence> ::= <subject> <can> <action> <preposition> <resource>

"Staff are allowed to print on all colour printers."

<sentence> ::= <subject> <can> <action> <preposition> <resource> "if" <condition>

"Students are permitted to print on printers if the number of pages is less than 50."

All three sentences above share the same semantic – creating a permission instance associated with corresponding actions on resources, then assigning it to the subject of the sentence.

Please note that since the project initially aims to output policies in either the PERMIS XML format or XACML XML format, the CNL tool is designed to fit the RBAC and ABAC models. But the CNL processing is not limited by this. For example, in the following sentence:

"Alice is allowed to read the confidential directory."

the permission is directly assigned to a user (Alice) rather than to a role or an attribute. This is fine as an English sentence, so this sentence is still allowed by the CNL. Whilst it would be possible to express this rule in an Access Control List (ACL) of users vs. permissions, this is currently not available for export, since ACLs are not supported by the tool.

#### 1.1.5.3 Role assignment rules

The semantic of the sentences in the third group is to create role assignment rules in the ontology. Role assignment rules say who is trusted to assign which roles to which domain of users. Example sentences are:

<sentence> ::= <administrator> <can> <assign> <role> <preposition> [ "user" <preposition> ] <userDomain>

Bob can assign staff to the company users.

Bob can assign staff to users from anywhere.

Bob has permission to allocate the manager role to users from the University of Kent.

<sentence> ::= "I trust" <administrator> <preposition> <say> "who" <role> "be"

I trust Bob to say who staff are.

Both forms express the similar rule of trusting *Bob* (a Source of Authority) to allocate the staff role to users from some domain. The first three sentences allow the user domain to be specified, namely: "the company users", "anywhere" and the "University of Kent", whereas the latter sentence does not (meaning that the domain is the entire world signified by the reserved word "anywhere").

## 1.2 The In-Built Policy Ontology

The most important pre-existing classes that have been built into the CNL tool, are as follows:

Policy: This is the top level class representing an access control policy;

User | User Domain: This is the class representing users who perform actions;

Attribute: Attribute is the high level class used to describe users, actions and resources;

Role: Role is a sub class of attribute, which is associated with users. Roles are assigned by administrators and used in RBAC rules;

Action: this is the high level action that users can perform on resources;

Resource: this is the high level resource that users can access;

Permission: this class is associated with the role, action and resource classes. It indicates that a user with a role is allowed to perform an action on a certain resource;

Administrator: administrators are authorities who may assign attributes to users;

Role_Assignment_Permission: this is a sub class of permission, which is associated with administrators. It declares that an administrator is allowed to assign a role or attribute to a class of users.

A full description of the built-in ontology can be found in project deliverable D7.1 [9] which is aligned with the ontology model in D2.2 [10] and D2.3 [11].

# 2 Overall Architecture for CNL Processing

## 2.1 Design Requirements

In order to build a good quality system, it is necessary to specify the system requirements. These requirements must be satisfied by the design. The requirements can be categorized into two groups, functional requirements and non-functional requirements.

Functional requirements are more obvious and directly affect user experiences. It defines the superficial behaviour of the software. These requirements include:

Information extraction: the software should be able to parse texts written in the CNL defined in the previous section, and extract policy-related information from it. As much information as possible should be extracted, though maybe not all of it will be used.

Policy generation: with the information extracted from the texts, the software should be able to generate access control policies based on it. The output policy should be in the format of a machine processable language, so that it can be loaded and executed by a policy based access control system.

Error diagnostics: if the system fails to output a policy for some reason, then it should output one or more error messages that are meaningful to the user and that allow the user to correct his mistakes without needing to call a helpdesk or system administrator.

Though non-functional requirements may not directly define the software's behaviour, nevertheless they are equally important since they affect the overall performance, maintenance and utility of the software. In our project, the non-functional requirements are as follows:

Usability: the whole system aims to improve the overall usability of access control systems. The CNL interface must be easy to use and intuitive, involving very little effort to learn, or ideally no effort to learn.

Extensibility: both new CNL sentences and new target policy formats are expected in future developments. Hence extensibility is an important feature of the system. It should be possible to add new sentences and policy formats with minimum affects on other parts of the system.

Transparency: the whole process should be transparent to the users. Users do not need to know how to control the process of information extraction and policy generation. All the users need to know is to feed the system a piece of text as input, and the system should output a policy in the target format.

Traceability: since NLP is always a complex task, it is important to make the system traceable so that when a bug occurs it can be traced to its source. This will make the system easy to maintain.

## 2.2 Overall Architecture

To satisfy the requirements above, the Mediator design pattern [18] is adopted. In the mediator pattern, process objects do not directly communicate with each other. Instead, communication between objects is encapsulated with mediator objects. This reduces the dependencies between communication objects, thereby lowering the coupling and increasing the extensibility. As a result, the following design is adopted (see figure 1).

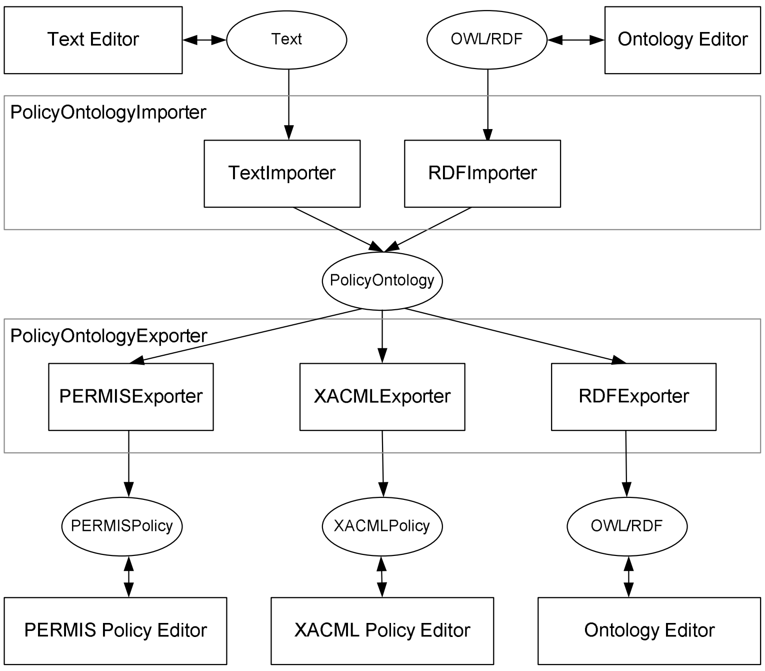


**Figure 1 - Overall Architecture Design**

In this design, the whole process is divided into two parts, information extraction and policy generation. An intermediate layer, an ontology of the access control policy, is the mediator object used to bridge them together. The system is designed in a modular fashion. Circles represent data models, or mediator objects, while rectangles represent process modules. Since all process modules are isolated by data models, problems and exceptions encountered by each process module are isolated as well. This makes it easier to trace problems at runtime. Only editor modules are visible to users, all other modules and data are transparent to users.

The *PolicyOntologyImporter* module is responsible for parsing the sentences written in the CNL and extracting the policy ontology from them. Alternatively, this module may directly import a policy ontology by loading ontology data from an OWL file. Finally this module, may perform a combination of both methods by loading OWL ontology data first and then supplementing it with CNL data .

The *PolicyOntologyExporter* module will go through the ontology model and build up the security policy in the chosen target format.

The *PolicyOntology* acts as a media to store the security policy in an abstract model, and provides access to both importers and exporters.

Data models are defined by existing standards and application-specific data structures:

- the *Text* model is defined by the CNL described in the previous section;
- the *OWL* model is defined by the W3C standard [6];
- the *PolicyOntology* is defined by the ontology described in D7.1 [9];
- the *PERMISPolicy* is defined by the PERMIS Policy schema [7];
- the *XACMLPolicy* is defined by the OASIS standard [8].

These data models act as interfaces between different processing modules.

By conforming to these data models, it is easy to modify an existing process module or add a new process module to extend the functionality. If any of these data models evolve or new data models are added, only the processing modules interacting with this data model need to be modified. For example, we only need to update the *PolicyOntologyImport* model for *Text* if new sentences are added to the CNL; and we only need to write a new *PolicyOntologyExportor* to support a new target format.

The whole system can be wrapped as java packages. It can then be called either from a command line or embedded with other applications via the java API. In general, only 2 API calls are required to convert a text to a policy in a target format, These are:

- - a call to the *PolicyOntologyImporter* to parse the input;
- - a call to *PolicyOntologyExporter* to get the policy in the target format.

An application of an embedded version of this CNL interface can be found in the PolicyEditor software described in D7.2 [10]. The design details of the CNL interface can be found in section 6.1 of this document. A web service has been developed, which makes the API available via the SOAP protocol. A web service interface has been developed and is available for public download.

# 3 Software Tools for CNL Processing

## 3.1 Natural Language Processing

Natural language processing can be classified into two categories: statistical and pattern. In the statistical method, language models are built from the statistics of the frequencies and positions of words in sentences. In the pattern method, language models are defined as patterns of phrases and sentences. We chose the latter method in our project, since it better fits small language models, such as those for Controlled Natural Languages.

There are some tools that have already been developed for natural language processing, such as Attempto [17], Apache UIMA [12] and GATE [13]. Attempto defines a CNL (together with a parser) for formalizing knowledge in general, but it doesn't provide an API to enable further development for specific applications. Apache UIMA (Unstructured Information Management Architecture) is an Apache-licensed open source implementation of the UIMA specification by OASIS. It provides a framework to develop NLP components and pipelines with Java or C++. In addition, there are several NLP components shipped with the Apache UIMA package. But the functions are currently limited due to the small number of available components. GATE (General Architecture for Text Engineering) is another platform for developing NLP components and pipelines. As a mature platform with a longer history and a rich set of ready to use NLP components, GATE has been chosen as the NLP platform in this project. Details of GATE will be given in the next section.

## 3.2 GATE

The GATE project is lead by the University of Sheffield, and is used worldwide by thousands of scientists, companies, teachers and students. A very important feature of GATE is that it brings software engineering methods into the natural language processing world. There is an engineering process to be followed so that quality can be assured. As an infrastructure, GATE defines a methodology for text engineering, and provides a set of tools for developing, debugging and deploying software components for all sorts of language processing tasks, including Information Extraction in many languages. It is composed of an open source development kit and a graphical development environment.

Components, also called Resources in GATE, are defined in a standard and reusable way. A group of components are packaged as Java archive files called CREOLE (a Collection of REusable Objects for Language Engineering). GATE itself has a set of components named ANNIE (a Nearly-New Information Extraction System). The following components are provided for general information extraction tasks:

**Tokeniser**: a tokeniser splits the text into very simple tokens such as numbers, punctuation and words of different types. An English tokeniser is shipped with ANNIE.

**Sentence Splitter**: splits the text into sentences. The splitter uses a gazetteer list of abbreviations such as e.g., .net, cf., etc. to help distinguish sentence-marking full stops from other kinds of full stops. It also has a list of other sentence ending characters such as ? and !

**Part of Speech Tagger**: the POS tagger produces a part-of-speech tag as an annotation on each word or symbol. These tags include noun, verb, adjective, determiner, etc.

**Gazetteer**: gazetteer lists are plain text files or ontology classes. Each list represents a set of names, such as names of cities, organisations, days of the week, etc.

Developments on the GATE platform are based on JAPE (Java Annotation Patterns Engine). Application specific grammar rules can be defined in JAPE Each grammar rule consists of two parts, a LHS (Left Hand Side) and a RHS (Right Hand Side). The LHS is a pattern in a regular expression of annotations of the text, while the RHS is a piece of Java code interpreting the pattern. Here is an example of the JAPE grammar:

```
 (

  ({Lookup.majorType == "Action"}:actionPhrase

  (({Prep})?):prep

  ({Lookup.majorType == "Resource"}):resourcePhrase

)

-->

{issrg.ontology.Policy.addPermision(actionPhrase, resourcePhrase);}
```

In the TAS3 project, a NLP component specific for access control policies was firstly developed with the utilities provided by GATE; and then it was embedded into the PERMIS Policy Editor by the SDK from GATE.

## 3.3 CLOnE

The NLP component in this project is built on an existing GATE application called CLOnE (Controlled Language for Ontology Editing). CLOnE is built upon earlier works of the GATE team, called CLIE (Controlled Language for Information Extractions)[1]. CLOnE has been developed to enable users to author ontologies in controlled language, which is easier than writing ontologies in RDF/OWL languages or with tools like Protégé.

In CLOnE, there are two parts: the Java API and the JAPE grammar. The Java API is designed to support ontology operations, such as creating a class or adding a property to the underlying ontology implementation. The JAPE grammar rules are used to match sentences for basic ontology statements, such as defining a

[1] CLIE was developed under the EU-IST Integrated Project (IP) IST-2003-506826 SEKT

class. If a sentence is matched with a defined JAPE grammar, then the Java API is called to manipulate the ontology according to the matched words. For example, "There are projects" matches the "There are <CLASSES>" pattern. Therefore the word "projects" is extracted and passed to the Java API to create a class named "Project" in the internal ontology model. Words are always formalized into their canonical form during this process, which means singular and title case.

With CLOnE, users may declare classes and instances with sentences like "There are projects. SEKT is a project". Properties between two classes or two instances can be stated as "A has B", e.g. "Companies have employees" (classes) or "IBM has LeiLei Shi" (instances). It also supports a sentence to delete a certain entry from an existing ontology, by using "Forget that". Here are sentences that can be directly used in specifying a security policy:

1. Template:There is/are **CLASSES**.

   Example : There are **users** and **roles**.

   Semantic: Create a new class immediately under the top class for each word.

2. Template: **INSTANCES** is a/are **CLASS**.

   Example : **Alice** and **Bob** are **users**.

   Semantic: For each word create an instance of the class specified.

3. Template: **CLASSES** is/are a type/types of **CLASS**.

   Example : **Printer** is a type of **resource**.

   Semantic: For each word in **CLASSES**, if it already exists as a class, make it a subclass of the class named by **CLASS**; if it does not exist, create a new class as a subclass of **CLASS**. If **CLASS** does not name an existing class, generate an error.

4. Template: **CLASSES/INSTANCES** have **CLASSES/INSTANCES**.

   Example : **Users** have **roles**. **Alice** has **staff**.

   Semantic: Iterate through the cross-product of words in the first list and words in the second list. For each pair, if both are classes, create a property of the form **Domain_has_Range**. If both are instances, find a suitable property and instantiate it with those instances; if there is a class-instance mismatch or a suitable property does not exist, generate an error.

These sentences are designed for general information extraction, and can be used to define a set of entities and relations between them. So the relations are defined in a rather generic way. More details of the rules of CLOnE are given in [14]. In fact with these sentences we are able to build ontologies for permission assignment rules, as follows:

*'Print on HP1' is a permission.* – this defines a new RBAC permission instance "Print on HP1".

*'Print on HP1' has print.* – this associates the print action to the new permission.

*'Print on HP1' has HP Laserjet 1.* – this associates the HP laserjet resource object to the new permission.

*Staff has 'Print on HP Laserjet 1'.* – this assigns the new permission to the Staff role.

Obviously, these are not sentences that users normally use when speaking and writing, and it looks more like another form of programming language rather than natural language. To build a policy ontology in this way, the user needs to know the restrictions imposed by CLOnE and how to construct an ontology. More essentially, the user needs to know the security model he is going to build, which is actually the problem we are try to address by this project.

Whilst CLOnE is a good starting point for authoring ontologies in controlled language, in its original form, it is not enough for writing access control policies, since it only supports a limited set of sentence patterns and limited relationships. In the next section, we will show how we have made further developments to CLOnE in order to support the authoring of access control policies in controlled language.

# 4 Modification and Integration of CLOnE

## 4.1 Ontology API

The Java API from CLOnE has been enhanced to enable the writing of more complicated rules and parsing of more complex text within the context of access control policies. In the original CLOnE API, it was only possible to access and update the ontology with literal information taken directly from the text. It was only possible to have a single type of relationship in a sentence, and the name of the relationship must be able to be built from classes and instances in the text e.g. "Subject_has_Role". For example, the following API is used to create an instance of an existing class from the sentence "*HP Laserjet 4 is a printer*":

boolean createInstance(**Ontology** ontology,

**Annotation** classChunk, **Annotation** instanceChunk)

The first parameter is a reference to an ontology repository, while the later two are an existing class name (*printer*) and a new instance name (*HP Laserjet 4*), taken from the text. This is fine for creating instances with names (annotations) from the text. But sentences for permission assignment rules and role assignment rules are more complicated than the base CLOnE sentences allow. They typically contain 3 or more classes or instances and several relationships, which are related to the knowledge domain of access control policies. For example in following sentence:

"Staff **can** print on HP Laserjet 1"

we have a relationship between staff and the permission to "print on HP Laserjet 1", and two relationships associated with the permission, one with the print action and one with the HP Laserjet 1 resource, which together define the permission. Whilst all literals from the text could be passed to the Java API in combinations from the set {*"Staff", "can", "print", "on", "HP Laserjet 1"*} this is not what we require. Though we know the word "can" actually implies the "permission" concept in the ontology, we were not able to use the permission concept in the original Java API since the word does not occur explicitly in the sentence and the original API only accepts parameters consisting of words from the text. Therefore the following new methods have been added to the Java API[2]:

boolean createInstance(**Ontology** ontology,

**Annotation** classChunk, **String** instanceName)

boolean createInstance(**Ontology** ontology,

**String** className, **Annotation** instanceChunk)

boolean createInstance(**Ontology** ontology,

**String** className, **String** instanceName)

[2] The full set of new methods that we have added to the API are listed in Appendix A.

With this set of methods, we are able to create instances of existing classes using arbitrary strings for either the instance name, the class name or both. With these new methods we can manipulate the "permission" ontology class and its instances, while parsing sentences involving the words “can” or “are allowed to” and that don’t contain the word “permission”. This is useful for writing JAPE rules which are able to parse texts where some of the information is implied by the context but is not explicitly stated in the text. The context in our case is represented in the form of an ontology for access control policies. An ontology template (see Appendix D for details) is loaded into GATE before parsing the text, and then JAPE rules are used to match the text and populate the ontology template by using the new API. Terms in the ontology, such as “permission” and “administrator” are directly used in the JAPE rules, though these terms never appear in the text typed by users.

But the new rules are even more complicated than this. With the original CLOnE API, although we could say “Alice and Bob have the Staff and the Manager roles”, it only involves a single relationship “Subject_has_Role”. It was not possible to manipulate more than one relationship or more than two classes in a single CLOnE sentence. However, in the new sentences which are used to express RBAC rules, we need to be able to parse “Role can Action on Target”, which involves four classes and three relationships, namely: “Permission_has_Action”, “Permission_has_Target” and “Role_has_Permission”. These three relationships can be directly referred to in the enhanced API, even though their names cannot be constructed directly from the text. This was infeasible with the original API.

Another important feature introduced by the new API is the **generic instance**, to address the problem of one-to-many relations, or to put it another way, we sometimes need to be able to associate an instance of one class with the entire set of instances of another class (or simply with the entire class). For example, the Print action, which is an instance of the action class, should be associated with all instances of the Printer class, since all printers support the print action. In the original CLOnE model, it was only possible to relate a single instance of one class to a single instance of another class or one class to another class, but not an instance of one class to another class (or all its instances). For example, we may want the “Manager” role (an instance of the role class) to have permission to print on all “Printers” (all instances of the printer class). Once this permission is granted, it should happen automatically for any new created “Printer” instance, such as HP Laserjet 4. But with the original CLOnE API, a relation existed only between two classes or two instances, but not between a class and an instance. Hence when a new instance of a class was created in the original text, the user would also have had to type a sentence to add the relation to the new instance, which would have reduced the user friendliness of the interface. With our enhancement, which is described below, the relation is automatically added into the ontology instantiation for all new instances as and when they are created.

To solve this problem, we invented a generic instance named “All_XXX” which is automatically created for each class XXX when the class is created. Whilst All_XXX is a single instance of XXX in the ontology model, semantically it actually represents all instances of this class. It allows the user to create a relationship between a single instance of one class with all the instances of another class, whilst the ontology model thinks the relationship is simply between two single instances. When processing the ontology, the properties held by the generic instance will be applied to all the sibling instances and child instances. The creating of the generic instance happens automatically with the enhanced API when a class is created or accessed for the first time. When the “Printer” class is created as a sub class of “Resource”, a generic instance "All_Printer" is created automatically for it. After associating the generic instance with the action instance "Print", (as in the text “staff can print on printers”) the Print action subsequently applies to all “Printer” instances when they are defined (as in the text “HP Laserjet 4 is a printer”). Another advantage of this is that it allows users to directly refer to all resources of a certain type, with sentences like: "David can print on all printers." Since the instance representing "all printers" already exists in the ontology model.

These changes have been necessary since users typically do not understand the fundamental difference between classes and instances, and typically they want to use these concepts interchangeably in sentences. For example, sometimes a user might use the word printer to mean a generic type of object (i.e. a class), and other times a specific object (i.e. an instance). A strict ontology modelling tool will not allow them to do this. Our enhancements now mean that users do not need to worry about the differences between instances and classes, and can use the same term in the text, when sometimes a class is required and other times an instance is required.

## 4.2 JAPE Rules

To support the parsing of sentences of access control policies, new JAPE rules have been developed with the enhanced ontology API. Here is a list of sentences accepted by the new JAPE rules, and their corresponding semantics.

1. Template: Role can Action on Resource.

   Example : Staff can print on LaserJet 1.

   Semantic: A Permission instance is created for each combination of Action and Resource instances. Permission_has_Action and Permission_has_Resource relations are used to link these instances. And then the Permission instances are associated with each listed Role instance, with the Role_has_Permission property.

2. Template: Role can Action Resource.

   Example : Staff can read files.

   Semantic: This has exactly the same semantic as the previous template, except that the proposition is omitted in the pattern. It is added as a separated rule, due the very different implementation of pattern matching.

3. Template: Role can Action on Resource if Condition.[3]

[3] In the current ontology design and implementation, conditions are represented in the form of strings. So currently the conditions ontology cannot be exported to and evaluated by real access control systems. This will be done in the next phase of the project.

Example: Students are allowed to print on LaserJet 1 if number of pages is less than 100.

Semantic: This template creates Permission instances with instances of Action and Resources, and then the Permission instances are associated with the Role instances. In addition, a Condition instance is created according to the condition specified in the sentence. This Condition instance is associated with the Permission instance with the Permission_has_Codition relation.

4. Template: Administrator can assign roles to users.

   Example : Alice can assign the Student role to users from the University of Kent."

   Semantic: Instances of Role_Assignment_Permission are created and associated with instances of Role and User. Then instances of Role_Assignment_Permission are assigned to the Administrator instances.

5. Template: I trust Administrator to say who Role are.

   Example : I trust Bob to say who Staff are.

   Semantic: This is another form of creating and assigning Role_Assignment_Permission to Administrators. Since there is no Users specified in this template, it is interpreted as the administrator has privileges to assign the roles to any user. So the Role_Assignment_Permission is associated with the generic instance All_User.

Please note that the reserved words in the grammar may have more than one form. For example, “are allowed to”, “is granted to” and “has permission to” are all treated the same as “can”. Gazetteer lists are used to identify these reserved words from the text.

The original JAPE grammar can still be used within the context of access control policies since we have not removed these rules. For example, with the original rule of “INSTANCE has INSTANCE”, users can write “Printers have print.” (even though it is not very intuitive). Users do not need to define the relation “Resources have Actions” since this is included in the pre-loaded ontology as background knowledge.

The use of a pre-loaded ontology means that users do not need to build the whole ontology model from scratch. First, it is very verbose to declare the basic model with a set of statements such as “There are roles”, “There are actions”, “There are permissions”, “Administrators have permission to assign roles” etc. Second it is a waste of time to repeat it every time a policy is created. Third it is a way to standardize the security model and the information that is required from users. Fourth, many users lack knowledge about security policies, are not expert or familiar with the role based access control model, so “Administrator” “Permission” and “Role” will not be a natural way to express their policies. A novice user might use a very different vocabulary and format to express the same idea e.g. “I trust someone to say who can do something”. So we need to standardize the model and concepts and allow alternative texts to be used for further processing.

## 4.3 Integration

A process pipeline has been defined in GATE. With the pipe line, the English Tokeniser, Sentence Splitter, POS Tagger, Gazetteer are called in sequence, to perform common tasks of text analysis. As the last step of the pipe line, the new JAPE grammar is applied. If a sentence matches the JAPE grammar, corresponding Java code will be called to build the ontology of the access control policy. Matched parts from the sentence will be passed as parameters.

With the SDK provided with GATE, the pipeline can be executed via Java code. Firstly, the GATE environment is initialised with the necessary settings. Then the pipeline is initialised from an XML file. After that, the pre-defined ontology template is loaded. The text is passed to the pipeline when everything is ready, and then the execute method is called. The last step may take a few seconds to complete, due to the time taken to match the patterns of the JAPE rules. After the whole process is finished, the resulting ontology is retrieved for querying or exporting.

The whole code is packaged in jar files, including the necessary files for GATE. All classes for parsing texts and generating the ontology are packaged as a jar package called policy importer. Also, the policy importer package handles some extra tasks besides executing the pipeline. For example, it is also in charge of reading the content if the text is from a file. More importantly, error handling takes place in the policy importer package. If the pipeline fails, any information returned from GATE is caught. Firstly, the policy importer package tries to see if there is anything that could be done to recover from the failure, such as resetting the ontology repository. If not, then information about the failure will be wrapped as an exception and returned to the caller.

The exception includes the failure information returned from GATE along with a more human readable description of the failure which can be shown to end users to make them more aware of what they should do next. Also, to make the problem easier to trace from the caller's side, the line number of the faulty text is attached (if the text passed in has multiple lines). This line number is counted by the policy importer code while feeding the text into the pipeline.

The policy importer package can be called either from a Java program, or via a command line to execute the jar file. A corresponding export package is also required to export the resulting ontology into a policy in one of the supported machine languages.

# 5 Exporting the Policies to Machine Languages

## 5.1 Ontology Exporter

Ontology exporters are designed to output policies in a target machine language format, so that the policies can be loaded and used in the Policy Decision Points (PDPs) of access control systems. There is an abstract class named PolicyOntologyExporter, which is the super class of all ontology exporters. It defines three common methods:

PolicyOntologyExporter(**PolicyOntology** ontology)

abstract Object getPolicy() throws PolicyExportException

abstract void writeToFile(**String** filename) throws PolicyExportException

The first method is implemented in an abstract class for all exporters, since the way of loading the ontology is always the same. The getPolicy method queries the loaded ontology and generates the policy in the target format, while the writeToFile method writes the result to a file. Due to the differences in the policy models and the syntaxes of the target formats, the latter two methods vary for each exporter.

## 5.2 RDF Exporter

A simple RDF exporter (RDFExporter) is provided as a subclass of the PolicyOntologyExporter. It outputs and writes the policy ontology in RDF format. The RDF exporter allows the user to save a policy ontology, created from CNL input, as RDF triples, which can then be viewed and edited using an ontology editor such as Protege. Currently we use the RDF repository in the OWL standard way, so the result can be saved and loaded as an OWL file. But there is no guarantee that it will always be fully compatible with OWL. So the export format is called RDF rather than OWL.

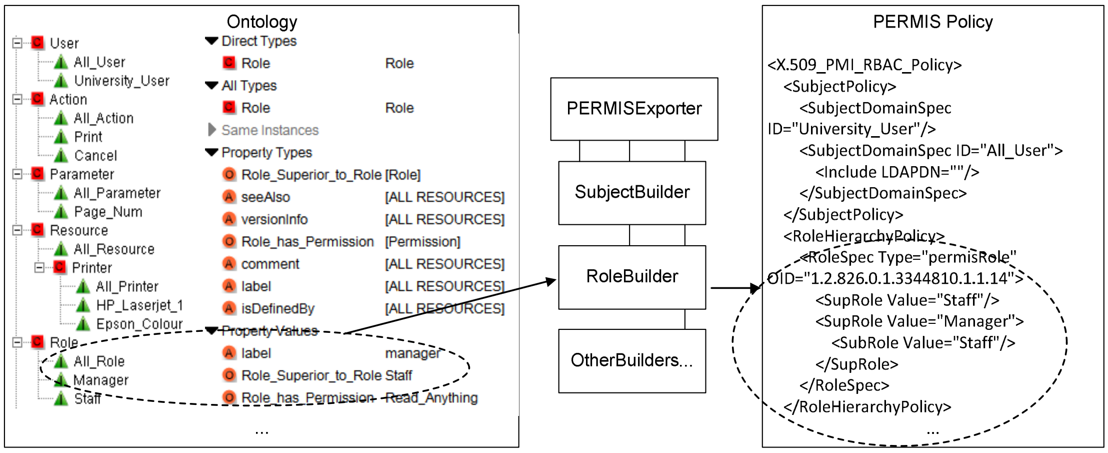
## 5.3 PERMIS Exporter

A PERMIS policy exporter (PERMISExporter) is implemented as a subclass of the PolicyOntologyExporter for exporting the policy in the PERMIS policy language. PERMIS policies are based on the RBAC model, and are written in XML. Further details of the PERMIS policy format can be found in. [7]

A PERMIS policy consists of several parts, namely: SubjectPolicy, ActionPolicy, TargetPolicy, RoleHierarchyPolicy, SOAPolicy, RoleAssignmentPolicy and TargetAccessPolicy. For each part of a PERMIS policy, a builder class is made. A builder class reads ontology instances related to the corresponding part of the policy, then generates XML tags with the names and relations of instances. Figure 2 illustrates how the PERMISExporter works.

For example, when the RoleBuilder class is called by the PERMISExporter, firstly it generates tags for all the roles. Then it iterates all instances of the Role class in the policy ontology, except the generic instance (ALL_Role). For each instance, a <SupRole> node is added, with the value attribute of the name of the instance. If there are any superiorTo relations between this role instance and others, the subordinate roles are added as <SubRole> nodes under the <SupRole> node.

Dependencies exist among different parts of a PERMIS policy. For example, the TargetAccessPolicy refers to information in the ActionPolicy, TargetPolicy and RoleHierarchyPolicy. To satisfy these dependencies, the following builders are called in sequence: SubjectBuilder, RoleHierarchyBuilder, SOABuilder, RoleAssignmentBuilder, TargetBuilder, ActionBuilder and TargetAccessBuilder.



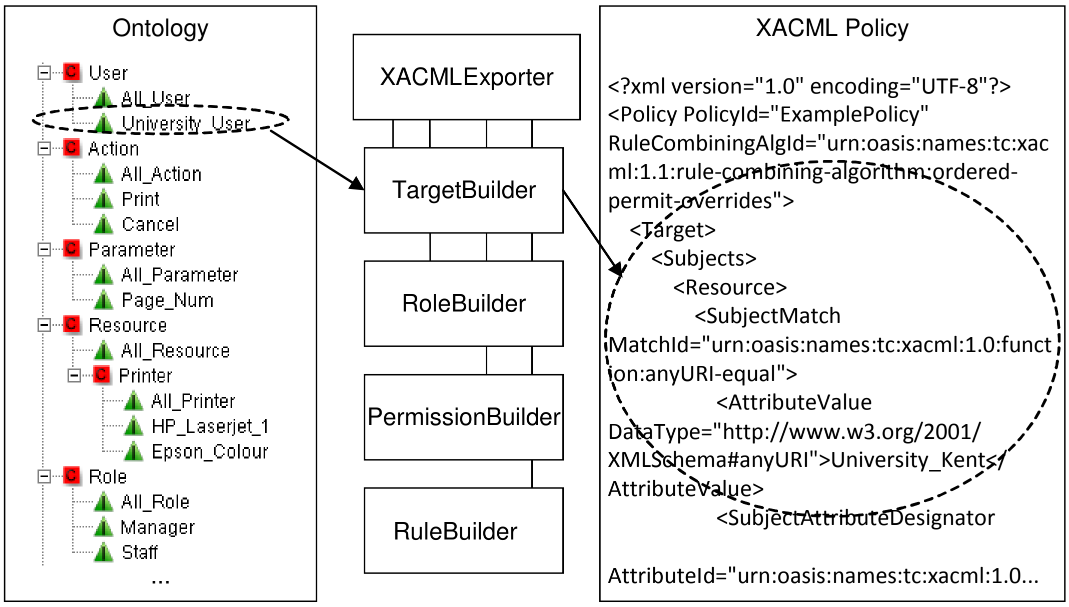
```
<X.509_PMI_RBAC_Policy>
   <SubjectPolicy>
      <SubjectDomainSpec
ID="University_User"/>
      <SubjectDomainSpec ID="All_User">
         <Include LDAPDN=""/>
      </SubjectDomainSpec>
   </SubjectPolicy>
   <RoleHierarchyPolicy>
      <RoleSpec Type="permisRole"
OID="1.2.826.0.1.3344810.1.1.14">
         <SupRole Value="Staff"/>
         <SupRole Value="Manager">
            <SubRole Value="Staff"/>
         </SupRole>
      </RoleSpec>
   </RoleHierarchyPolicy>
                     ...
```

**Figure 2 - PERMIS Policy Exporter**

Generic instances are treated differently based on the situation. If a generic instance is not used at all, then it doesn't appear in the exported policy. For example, in the sentence “Alice is allowed to assign the staff role to all users”, the generic instance “All_User” is added to the policy as a subject domain which includes all users (see Figure 2). But in the sentence “Alice is allowed to assign all roles to users from the University”, then “All_Role” is actually replaced with all the defined role instances and “All_Role” is not used in the exported policy. The precise handling of the generic instances depends in part on the semantics of the export format.

## 5.4 XACML Exporter

An XACMLv2 policy exporter has been implemented for generating XACMLv2 policies from ontologies. As shown in Figure 3, the design of the XACMLv2 exporter is similar to the exporter for PERMIS policies. The task of generating target policies is broken into sub tasks. Each task builds a portion of the target policy according to the structure of XACMLv2 policies.



```
<?xml version="1.0" encoding="UTF-8"?>
<Policy PolicyId="ExamplePolicy"
RuleCombiningAlgId="urn:oasis:names:tc:xac
ml:1.1:rule-combining-algorithm:ordered-
permit-overrides">
  <Target>
    <Subjects>
      <Resource>
        <SubjectMatch
MatchId="urn:oasis:names:tc:xacml:1.0:funct
ion:anyURI-equal">
          <AttributeValue
DataType="http://www.w3.org/2001/
XMLSchema#anyURI">University_Kent</
AttributeValue>
          <SubjectAttributeDesignator

AttributeId="urn:oasis:names:tc:xacml:1.0...
```

**Figure 3 – XACMLv2 Policy Exporter**

An XACMLv2 policy is structured as a tree of PolicySets, Policies and Rules. The attributes of Subjects, Actions and Resources are used to match against the Rules of Policies and PolicySets, to see if the rules apply. If one or more rules are matched, the specified Effect (permit, deny) is returned. Conflict resolution rules are used to determine the overall effect if there are multiple applicable ones. XACML is based on the ABAC model, and doesn't directly support the roles defined in RBAC. The RBAC profile [21] for XACMLv2 is used to support RBAC policies represented by the policy ontology. Figure 4 show the structure of the XACMLv2 policies generated by the exporter.

At the top level there are two PolicySets, the Role Permission Assignment PolicySet and the User Role Assignment PolicySet. The Role Permission Assignment PolicySet specifies the role hierarchy and the permissions associated with these roles. A Role PolicySet is generated for each Role instance defined in the policy ontology. According to the RBAC profile for XACMLv2, the Target element in the Role PolicySet specifies the matching between the Subject's presented role attribute and the policy role value. The set of Permission PolicySets states the permissions associated with this role. Each Permission PolicySet consists of a Permission Policy with a set of Permission Rules. A Permission Rule specifies the combination of Action and Resource which defines the permission.. To support role hierarchies, the Role PolicySet of a superior role holds references to all Permission PolicySets associated with its subordinate roles.

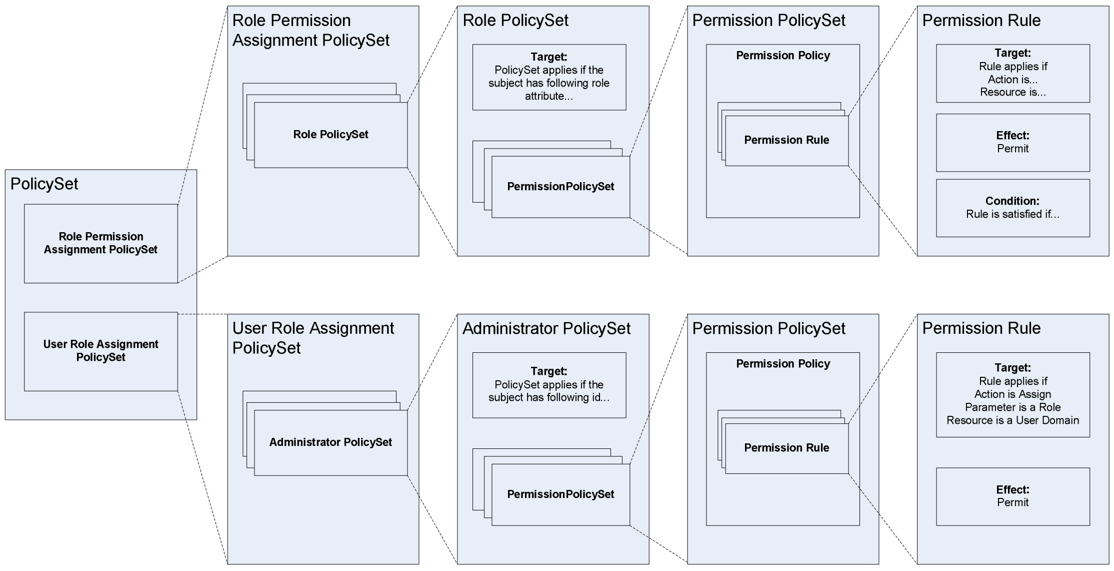

PolicySet
Role Permission Assignment PolicySet
User Role Assignment PolicySet
Role Permission Assignment PolicySet
Role PolicySet
Role PolicySet
Target: PolicySet applies if the subject has following role attribute...
PermissionPolicySet
Permission PolicySet
Permission Policy
Permission Rule
Permission Rule
Target: Rule applies if Action is... Resource is...
Effect: Permit
Condition: Rule is satisfied if...
User Role Assignment PolicySet
Administrator PolicySet
Administrator PolicySet
Target: PolicySet applies if the subject has following id...
PermissionPolicySet
Permission PolicySet
Permission Policy
Permission Rule
Permission Rule
Target: Rule applies if Action is Assign Parameter is a Role Resource is a User Domain
Effect: Permit

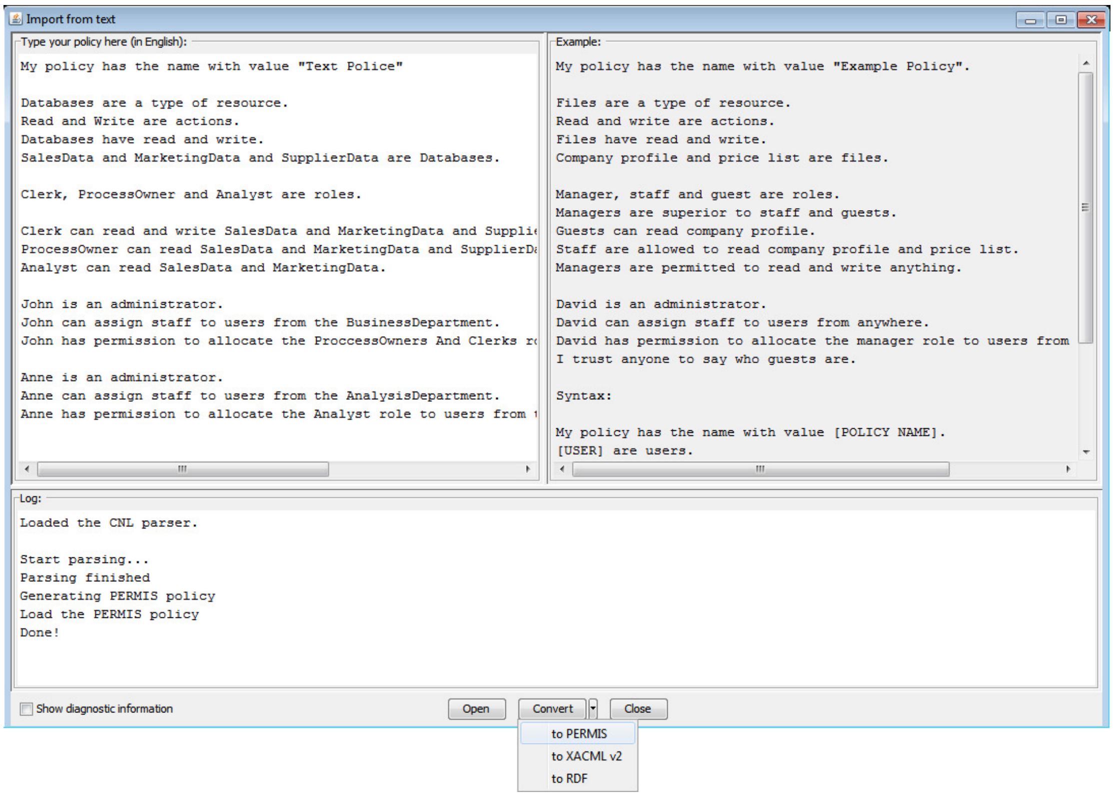
**Figure 4 - XACML Policy Structure**

In the standard XACMLv2 specification, all attributes are passed in by the PIPs and are trusted by the PDP. An XACMLv2 policy cannot serve the purpose of validating role attributes, a function which is currently performed by the credential validation service defined in D7.1 [9]. To support the specification of credential validation policies using the XACMLv2 language, we extended the standard RBAC profile by defining an Administrator PolicySet and a new form of Rule. An Administrator PolicySet is created for each Administrator instance defined in the policy ontology. (The Administrator is the person trusted to assign roles to users.) The Action of the Permission Rule is fixed with the "Assign" action and has a single Role parameter, being the role the administrator is trusted to assign to users. The Resource of the Permission Rule is set to the user domain defined in the policy ontology. The PERMIS CVS (Credential Validation Service) will be extended to consume these XACMLv2 policies for role validation requests. Then these XACML v2 user role assignment policy sets can be used in two ways. They can be used as an access control policy for controlling the action of assigning roles to users. Also, they can be used to verify whether a role attribute is valid when the role attribute is contained inside a credential that is being used in an access request e.g. as in the CVS

# 6 GUI for CNL Policies

## 6.1 GUI Design

In order to integrate the CNL packages with the existing Graphical User Interface (GUI) editor for PERMIS policies, an additional GUI screen has been developed (see Figure 5). With this GUI screen, users can type and edit policy text in a text area, then convert it to the target format with a single button click. The GUI wraps all calls to the policy importer and policy exporter. Also, it handles information returned from the CNL package.



```
My policy has the name with value "Text Police"

Databases are a type of resource.
Read and Write are actions.
Databases have read and write.
SalesData and MarketingData and SupplierData are Databases.

Clerk, ProcessOwner and Analyst are roles.

Clerk can read and write SalesData and MarketingData and Supplie
ProcessOwner can read SalesData and MarketingData and SupplierDa
Analyst can read SalesData and MarketingData.

John is an administrator.
John can assign staff to users from the BusinessDepartment.
John has permission to allocate the ProccessOwners And Clerks ro

Anne is an administrator.
Anne can assign staff to users from the AnalysisDepartment.
Anne has permission to allocate the Analyst role to users from t
```

```
My policy has the name with value "Example Policy".

Files are a type of resource.
Read and write are actions.
Files have read and write.
Company profile and price list are files.

Manager, staff and guest are roles.
Managers are superior to staff and guests.
Guests can read company profile.
Staff are allowed to read company profile and price list.
Managers are permitted to read and write anything.

David is an administrator.
David can assign staff to users from anywhere.
David has permission to allocate the manager role to users from
I trust anyone to say who guests are.

Syntax:

My policy has the name with value [POLICY NAME].
[USER] are users.
```

```
Loaded the CNL parser.

Start parsing...
Parsing finished
Generating PERMIS policy
Load the PERMIS policy
Done!
```

**Figure 5 – CNL GUI Interface built into the PERMIS Policy Editor**

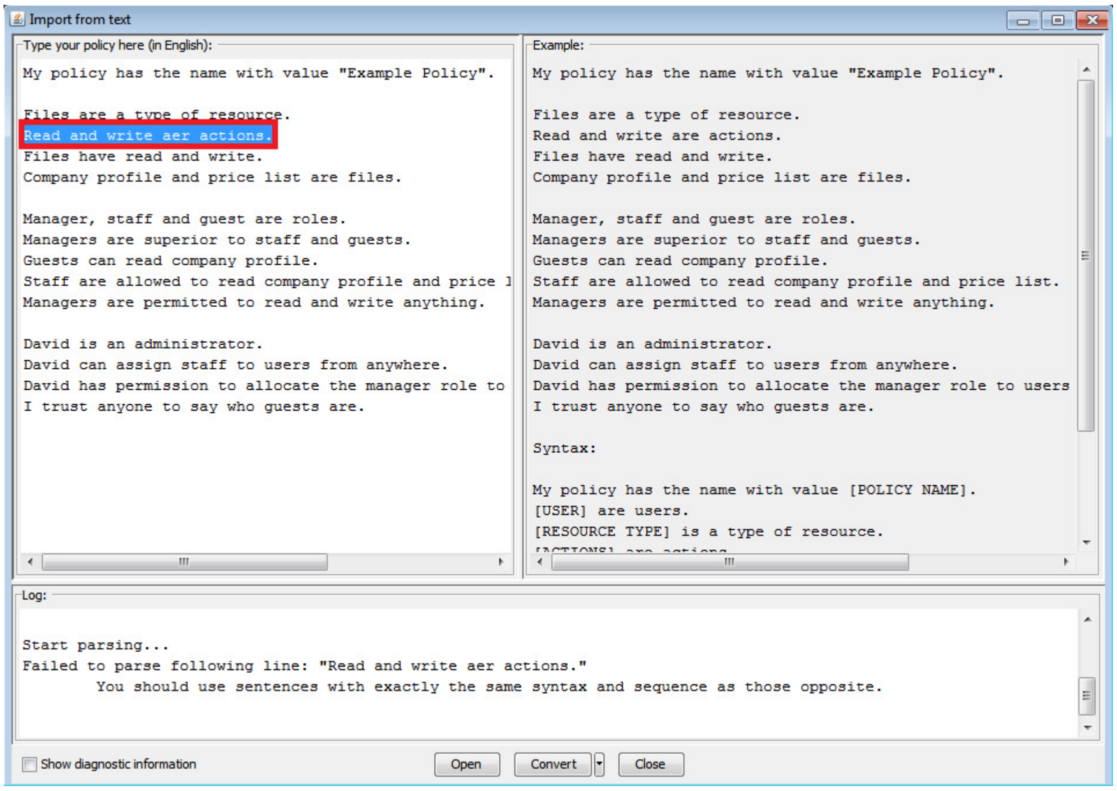
To make the GUI simple and easy to use, the whole interface is divided into three areas: edit area, guide area and output area. There are three buttons at the bottom, namely Open, Convert and Close. When the Convert button is clicked, the CNL package is called to import the text to its equivalent policy ontology, and export the ontology as a policy in the chosen target format (PERMIS XML, OWL/RDF or XACML). The target format can be chosen by clicking the drop down arrow attached to the Convert button. The last chosen option will be used when the Convert button is clicked again. If PERMIS XML is selected as the target format, the resulting policy is loaded and shown in the GUI screens of the PERMIS Policy Editor. For other formats, a simple text editor displaying the resulting policy is shown, and this allows the user to view and save the policy in a file of their choice. When the Open button is clicked, a system dialog is popped up to ask the user to pick a text file, which will be loaded to the edit area. The Close button is used to close the CNL interface. After the CNL interface is closed, the

CNL system still stays in the system memory, so that there is no need to initialise the GATE system again if the user wishes to use the CNL interface again.

In the guide area, example sentences with the correct CNL syntax are provided to help users to learn how to structure and write sentences for access control polices using the grammar of the controlled language. The guide area is always greyed out to distinguish it from the edit area, since in most systems the greyed out style is used to mark the components that are non-changeable. The sentences are designed to demonstrate as many combinations of the CNL syntax as possible in the space available. The format of the allowed syntax is listed below all the example sentences, as a reference guide for the user. Also, the example sentences demonstrate the structure and sequence of text that is needed to produce a correct access control policy. For example, resources and actions must be defined before they are referenced by the access control rules.

In the edit area, users can type and edit the text of their policy statements. Simple editing functions are provided, such as cut, copy and paste. It is also possible to load an existing text file to the edit area (using Open), then make further edits to this before finally converting it to the policy ontology. During the process of conversion, the edit area is disabled and greyed out, and users cannot make any changes to the text. This means that if any sentence cannot be parsed by the CNL importer, the interface is then able to highlight the offending sentence in the text area before the user has had chance to change it. The user may then edit it straight away (as Figure 6 shows). Another purpose of greying out the interface is to indicate the status of the conversion process. When conversion ends, the edit area becomes editable again.

Users may add new sentences to existing text that has already been converted and then press Convert again. In this case the new sentences will be added to the existing policy ontology. This is actually implemented by removing the existing ontology and converting the entire text again, so that all text changes are captured in the revised ontology. Thus if a user edits any arbitrary text, this change will be captured in the revised ontology.


Import from text
Type your policy here (in English):
Example:
Log:
Show diagnostic information
Open
Convert
Close

```
My policy has the name with value "Example Policy".

Files are a type of resource.
Read and write aer actions.
Files have read and write.
Company profile and price list are files.

Manager, staff and guest are roles.
Managers are superior to staff and guests.
Guests can read company profile.
Staff are allowed to read company profile and price l
Managers are permitted to read and write anything.

David is an administrator.
David can assign staff to users from anywhere.
David has permission to allocate the manager role to
I trust anyone to say who guests are.
```

```
My policy has the name with value "Example Policy".

Files are a type of resource.
Read and write are actions.
Files have read and write.
Company profile and price list are files.

Manager, staff and guest are roles.
Managers are superior to staff and guests.
Guests can read company profile.
Staff are allowed to read company profile and price list.
Managers are permitted to read and write anything.

David is an administrator.
David can assign staff to users from anywhere.
David has permission to allocate the manager role to users
I trust anyone to say who guests are.

Syntax:

My policy has the name with value [POLICY NAME].
[USER] are users.
[RESOURCE TYPE] is a type of resource.
```

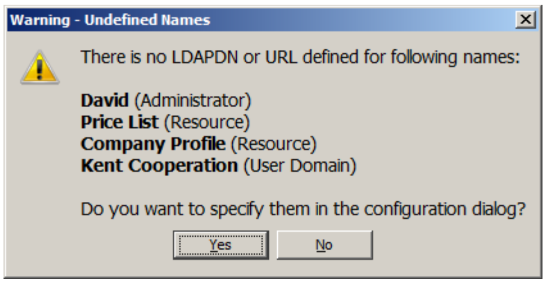
```
Start parsing...
Failed to parse following line: "Read and write aer actions."
        You should use sentences with exactly the same syntax and sequence as those opposite.
```

**Figure 6 – Highlight the Sentence Failed To Parse**

The output area provides feedback information to the user. Since some stages of the conversion may take seconds to complete, such as initialising the GATE software and converting the sentences, it is necessary to let users know what the current status is. Otherwise, users may get confused as to whether the interface has hung or is still processing. Also, when things go wrong, the output area tells the user what the problem is and what steps may help to fix it, in a human readable way. An option is provided, by clicking the *Show diagnostic information* check box, to provide more technical and verbose information for advanced users, such as exceptions thrown by the GATE system. This makes it possible to troubleshoot problems and bugs in the code at runtime.

## 6.2 URL/DN Auto Completion and Configuration

XML policies often require names to be formatted as URNs, URLs or DNs (Distinguish Names). Whilst it would be possible to allow CNL policies to use URLs and DNs as the names of users and resources etc., this would make the CNL more unusable for average users. To make the task of entering complex names more user-friendly, a feature has been implemented to automatically detect the use of friendly names in policies, and allow users to subsequently complete them with the complex structures that may be required. This happens before a target policy is generated. When unidentified friendly names are detected in a policy, they are listed in a dialog box (Figure 7) which asks the user to specify URLs or DNs as appropriate:


Warning - Undefined Names

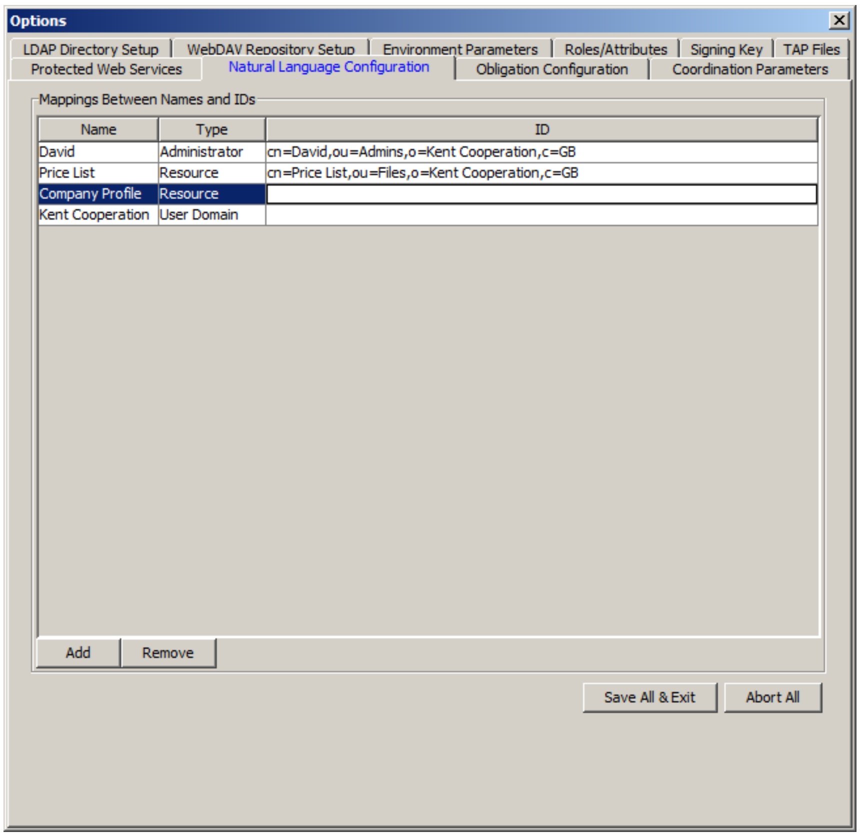
There is no LDAPDN or URL defined for following names:

**David** (Administrator)
**Price List** (Resource)
**Company Profile** (Resource)
**Kent Cooperation** (User Domain)

Do you want to specify them in the configuration dialog?

**Figure 7 - Prompt for Missing Identities**

If the user chooses yes, a configuration dialog (Figure 8) is shown listing the friendly name and its type (Resource, User Domain etc.). The user can then specify the appropriate URL or DN for each friendly name. It is also possible to access this configuration screen by choosing the Options menu item. Users can add/remove friendly names and their URLs/DNs, or amend existing entries. This list is saved into the configuration file and used for future policy authoring tasks, so that the user doesn't need to enter these details again.



| Name | Type | ID |
|---|---|---|
| David | Administrator | cn=David,ou=Admins,o=Kent Cooperation,c=GB |
| Price List | Resource | cn=Price List,ou=Files,o=Kent Cooperation,c=GB |
| Company Profile | Resource | |
| Kent Cooperation | User Domain | |

**Figure 8 - Configuration of URLs and DNs for Friendly Names**

The feature has been implemented with two pre-processors for the policy exporters. One is the EntitySupplement, which loads the configuration file and then populates the policy ontology with the URLs and DNs stored in it. The EntitySupplement pre-processor enumerates the friendly names in the policy ontology and checks whether a URL/DN is needed but has been missed. If yes, then it tries to match the friendly name with one stored in the configuration file. If a match is found, it then replaces the friendly name with the URL/DN. The other pre-processor is the EntityChecker, which checks if there are still any friendly names with missing URLs/DNs, after EntitySupplement has run. If there are, then EntityChecker displays the list of names to the user (as in Figure 7) followed by the configuration screen allowing the user to enter the URL/DN (Figure 8). Since these two pre-processors run on the policy ontology, they can be applied to any output target policy language.

A limitation in the current scheme is that only one URL/DN is allowed for each friendly name. This is not enough for some cases, for example a user domain in a PERMIS policy can refer to multiple LDAP DNs. In these cases the user will have to make further edits using others tools, such as the PERMIS GUI, or an XML editor.

# 7 API and Extension

## 7.1 API for Applications

An API is provided to integrate the CNL process with other applications. According to the overall design, there are two parts of the API, importers and exporters.

PolicyOntologyImporter is defined as the super class for all other importer classes. We have implemented three importer classes: TextImporter, TextFileImporter and RDFFileImporter. Each class has a different constructor, depending on the source of the policy that is passed to the constructor:

TextImporter(String text)

TextFileImporter(String filename)

RDFFileImporter(String filename)

They share the same method for the retrieval of the ontology result. This method is declared in the super class, PolicyOntologyImporter.

PolicyOntology getOntology()

An application can instantiate a PolicyOntologyImporter with a constructor according to the source of the ontology, and then call the getOntology method to retrieve the result.

For example, firstly the caller creates an importer instance of the TextFileImporter class, which is used to import a policy ontology from a text file called “Fred.txt”. The filename “Fred.txt” is passed to the constructor as a parameter. Then the caller calls the getOntology method to retrieve the imported policy ontology:

PolicyOntologyImporter importer = new TextFileImporter(“Fred.txt”);

return importer.getOntology();

To make the interface even simpler to use, three static methods are defined in the super class. In each method, a corresponding importer is constructed and then the result is returned:

PolicyOntology fromTextString(String content)

PolicyOntology fromTextFile(String filename)

PolicyOntology fromRDFFile(String filename)

In this way the whole import process can be completed with a single line of Java code. Using the example shown above, importing a policy ontology from a text file named “Fred.txt”, now the user only needs to write one line to do the same task:

return fromTextFile (“Fred.txt”);

PolicyOntologyExporter is the super class for all exporter classes. There are three sub classes of it: PERMISExporter, RDFExporter and XACMLExporter. At the current time PERMISExporter, XACMLExporter and RDFExporter have been implemented to export a policy ontology to PERMIS XML, XACML or RDF XML format. Unlike the importer classes, all the exporter classes have a constructor with a single argument of type PolicyOntology, since all these classes export policies from the same ontology:

PolicyOntologyExporter(PolicyOntology ontology)

RDFExporter(PolicyOntology ontology)

PERMISExporter(PolicyOntology ontology)

XACMLExporter(PolicyOntology ontology)

Two abstract methods have been defined in the super class, and have to be implemented in the sub classes according to the format of the target machine language. These are:

Object getPolicy() throws PolicyExportException

void writeToFile(String filename) throws PolicyExportException

In an application, we can export a policy ontology by firstly constructing an exporter class, then either getting the result object or writing the policy to a file:

PolicyOntologyExporter  exporter = new PERMISExporter (ontology);

return exporter.getPolicy();

Similarly, static methods have been defined in the super class in order to make the API easier to use:

Document getRDFXML(PolicyOntology ontology)

Document getPERMISXML(PolicyOntology ontology)

Document getXACMLXML(PolicyOntology ontology)

void writeToRDFFile(PolicyOntology ontology, String filename)

void writeToPERMISFile(PolicyOntology ontology, String filename)

void writeToXACMLFile(PolicyOntology ontology, String filename)

So in a typical application, the following two lines of Java code are needed to convert a controlled natural language access control policy into a policy in the PERMIS XML format:

PolicyOntology ontology = PolicyOntologyImporter.fromTextFile(inputFileName);

PolicyOntologyExporter.writeToPERMISFile(ontology, outputFileName);

## 7.2 API for New Exporters

New export language formats can be supported by deriving a sub class from the PolicyOntologyExporter super class and implementing two abstract methods, one to export the policy as a Java object and one to export it to a file. These two methods must query the policy ontology in order to build the policy in the new format.

The policy ontology is stored and manipulated using a low level API for accessing an RDF repository, which treats classes and properties in a different way to that used in Object-Oriented systems. In OO systems, a property is always understood to be part of an object. Hence, to retrieve a property value, we have to first get a reference to the object, then retrieve the value of the corresponding property. But in the ontology world, properties are first class citizens. They are not parts of an object, but instead are connections between two objects. A connection belongs to neither the start point (known as the domain in ontology terminology), nor the end point (known as the range in ontology terminology). Properties are also called relations in ontology terminology. They are stored and queried separately from objects in an ontology repository. For example, it is possible to query for properties/connections that start from a set of objects, or end at a set of objects, or without any conditions attached (i.e. no constrains on their starting or ending points). In the case of querying for a property of an object A, firstly we need to construct the URI of the property. Basically this is the name of the property prefixed with a name space. Then we use the resulting URI as the property name to query all properties starting with the object A. With the returned result, which is a set of property instances, we iterate each one and collect the objects at the ending points, which is equivalent to the property value in OO models. The set of objects at the ending points can then be used as OO properties of the object A. The Java code used to construct URIs, query properties, and collect endpoint objects will use the same logic, but different parameters. Consequently it would be quite verbose to use the RDF API and it will generate many duplicate or similar Java programs. Rather than querying the ontology repository directly, we have developed a higher level API to access the classes and instances in the ontology in a way which is more similar to using classes and instances in Java. In this way the complex ontology namespaces and URIs are wrapped by the higher level API. For example, if we want to query the ontology to ask what permission is associated with a certain role in the ontology, we can simply call the higher level getPermissions method for that role instance (in Java). Without this high level method we would need to query the RDF ontology repository for all http://sec.cs.kent.ac.uk/permis/policy#Role_has_Permission relations associated with that role, and then query the repository again for the range instance of each relation.

Another benefit that comes with this high level API is that all namespaces and names are contained within the API. If there is any change that needs to be made to a namespace, class name or relation name in the ontology, we can make these changes within the API and then these changes will be propagated to all the exporters that use it.

Note that the higher level API can only access the ontology in read-only mode, since it only queries the ontology repository and returns a wrapped result. There is no way to explicitly construct an instance with this API, since all constructors are protected and invisible outside the class. There following methods are provided to get instances from a policy ontology:

OntologyInstance PolicyOntology.getOntologyInstance

(String instanceName)

OntologyInstance OntologyClass.getGenericInstance()

Set<OntologyInstance> OntologyClass.getAllInstances

(boolean includingSubClasses)

The first method can be used get a reference of an instance by its name; the second method returns the generic instance of a certain class; the third method returns all instances of a class, and a boolean parameter is used to indicate whether instances of subclasses should be included or not in the result set. All results are returned as references to the ontology. Hence no new instances are created. All the methods are defined as get methods. There are no set methods to write back to the ontology.

## 7.3 API for JAPE rules

The enhanced CLOnE API is provided to access the ontology repository in read and write mode. It is a low level API for accessing the ontology and it is ontology oriented, rather than object oriented. So whilst there is a Java class for the ontology instance and ontology class, there is no corresponding Java class for Role or User. Nevertheless it provides a very flexible way to manipulate the ontology classes and their relations. This API can be used for writing new JAPE rules, when a new CNL text syntax is required. Details of this API can be found in Section 4 and Appendix A.

# 8 User Trials

## 8.1 Test Design

User trails were undertaken to evaluate the CNL interface, and to compare it with the pre-existing graphical user interface (GUI). There were three aims of the user trails:

- Evaluate the usability of the Controlled Natural Language CNL interface.

- Make a usability comparison of the CNL interface and the GUI interface.

- Find if the users still have any difficulties in authoring a policy, and suggest ways to overcome them.

We choose the Observation method to do the usability tests. This means that we ask users to use the software, and then we watch and record what they do. This reveals the actual behaviour of users whilst they use the software, and gives the observers direct information about the interaction between the users and the computer. Since the functionalities of the two interfaces are the same i.e. authoring access control policies, then each interface is tested as a whole rather than in a series of tests of sub-functionalities. A test scenario (see Appendix C) was provided based on a typical RBAC scenario within a virtual organisation. The scenario was developed to be compact, easy to understand, and to represent the most fundamental concepts of the RBAC model, such as users, roles and permissions. Each RBAC concept appears at least twice in the scenario to make sure it is not missed by the users. The same scenario is used in all the trails, so the results are comparable.

The whole evaluation was carried out in four stages:

**Preparation**: this is the stage for designing the trials and the test scenarios, preparing the documents and the environment for the evaluation.

**Pilot Trials**: these are performed before the real user trials are carried out. This stage is for tuning the environment and the instructions for the real user trails, and the result will not be counted. The instructions and scenarios may need to be adjusted according to feedback from the pilot trials. If the changes are numerous, a second set of pilot trials will need to be carried out.

**User Trials**: Recruit the participants (only native English speakers should be recruited) carry out the user trials and record all the results.

**Analysis**: Analyse the results from the user trials and write them up.

Participants were recruited from undergraduates, postgraduates and support staff from different departments of the university, in order to represent a wide population of users. There is no ideal number for participants. Studies [19] [20] show that a usability test with 10±2 users may discovery around 80% of overall usability issues. Since we are doing a comparative study, we decided to use 20 participants, so that 10 could be assigned to each user interface first. In order to make a fair comparison of the two interfaces (GUI and CNL), the users were divided into two groups. The two groups of users were matched demographically to make the cross group comparison valid. This means that the distributions of gender, age, education and occupation are very close in the two groups.

Each group was asked to write an access control policy for the same scenario. Group A used the graphical interface first, and then used the CNL interface; Group B used the CNL interface first and then the graphical interface. The results can then be used to compare the effectiveness and learning curves of the two interfaces. The two groups were the same size and matched in other demographic ways such as: gender, age, education, technical experience etc. as far as was possible.

## 8.2 Usability Test

According to the definition in ISO 9241-11 [15], usability is evaluated in three aspects: effectiveness, efficiency and satisfaction. Effectiveness and efficiency are objective, which can be observed and recorded directly. Satisfaction is a subjective indicator. A questionnaire of Likert scales is used to collect data of user's subjective feelings of the two interfaces. Here is how the three aspects of usability were measured through the user trials:

**Effectiveness**: Effectiveness was measured in three ways: completeness, accuracy and correctness. A reference XML policy was produced which implemented the test scenario, and the policies produced by both groups were compared to the reference XML, element by element. Only those elements which semantically matched the reference policy were counted as correct elements, other syntactically correct, but semantically incorrect, elements were counted as wrong elements. The number of matched elements was used to calculate the completeness and the accuracy. Completeness was measured as the number of matched elements completed by users divided by the total number of elements in the reference policy, whereas accuracy was measured as the number of correctly matched elements divided by the number of all elements input by the users. Correctness was measured as the number of correct user provided elements divided by the total number of elements in the reference policy. Note that if the policies produced by the participants were structured in a different way to the reference policy, they were compared in a functional way. Any elements which were functionally equivalent to the reference policy were counted.

**Efficiency**: Efficiency was measured by the completion time, the relative completion times, the relative ease of learning and the transferrable knowledge. The completion time was recorded by screen capture software. If some users did not complete after the maximum time allowed, then an estimate of their completion time was made based on the completeness measured above, computed as (time taken / completeness). The relative completion time was computed as a simple ratio of the completion times of the first set of tests carried out by each group of users. However, this takes no account of the accuracy and therefore of the quality of the learning experience. The relative ease of learning is therefore based on the completion times and the accuracies of the completed tasks. The transferrable knowledge is a measure of how much a user can learn from one interface and transfer this to using the second interface. The greater the improvement in use of the second interface, by using the first interface initially, is a measure of the transferrable knowledge gained from using the first interface. The improvements in both completeness and accuracy were measured.

Say one group of users tries the GUI first and:

the completion time is A secs, and

the accuracy is N%;

while the other group of users tries the CNL first, with

a completion time of B secs, and

an accuracy of M%.

Say also that the first group of users tries the CNL second with

a completion time of C secs, and

an accuracy of L%,

whilst the second group of users tries the GUI second, with

a completion time of D secs, and

an accuracy of K%,

then we can say that

the relative ease of completion GUI:CNL is 1 : A/B.

A number greater than 1 indicates how much easier it is to complete a policy using the CNL interface compared to using the GUI, whilst a number smaller than 1 indicates how much more difficult it is.

the relative ease of learning GUI:CNL is 1 : AM / BN ,

A number greater than 1 indicates how much easier it is to create a correct policy using the CNL interface.

the transferrable knowledge of the CNL for completing the GUI task more quickly is (A-D)/A and for gaining accuracy is (K-N)/N. Similar equations apply to the GUI. This is a measure of how one interface helps users to learn to use the second interface.

**Satisfaction**: Satisfaction was measured by suitably adapting[4] the PSSUQ (Post Study System Usability Questionnaire) developed by IBM [16] There are 19 questions in PSSUQ. Each question is a statement about the usability of the system. Participants need to respond to each statement using a Likert scale of 7 points, where 1 indicates the user "strongly agrees" with the statement whilst 7 indicates the user "strongly disagrees". The complete questionnaire is provided in Appendix B. The questionnaire was developed with the support of a

[4] We removed the N/A option from each question, to make sure that the users gave their opinion about each question.

comprehensive psychometric analysis. Also, it provides scales for three sub factors, namely: system usefulness, system information and system interface, which are useful for evaluating the perceived satisfaction in more detail. The results were analysed in both total scores and sub factor scores. The CNL interface and GUI interface were evaluated individually and then comparatively.

## 8.3 Evaluation Result

The usability of both interfaces is analyzed and compared below:

### 8.3.1 Effectiveness

The final policies authored by the users were counted element by element, and then matched with the template policy. Only elements with semantic meanings were counted during the matching process. For example, the SubjectPolicy element is defined as the parent of all SubjectDomainSpec elements, but it doesn't define any subject domain on its own, so it is not counted. The template policy made for the scenario contained 57 elements with semantic meanings. Elements were matched in a functional way, meaning that the sequence of the elements and the spellings of the input terms were ignored in the matching process, since they don't affect the functioning of a policy. Completeness of each final policy was calculated as the percentage of matched elements against all 57 elements.

As the results in Table 2 show, first time users performed better with the CNL interface in all three aspects of effectiveness. For the users who began with the GUI (group A), the correctness rate was 35%, whilst for the users who began with the CNL interface (group B), the correctness rate was 63%. The greatest improvement happens in the ActionPolicy, where the correctness rate raises from 28% (with the GUI) to 75% (with the CNL interface). The correctness rate of the TargetAccessPolicy, which depends on the ActionPolicy, raises from 28% (with the GUI) to 47% (with the CNL interface). Similar improvements can also be observed in the RoleAssignmentPolicy.

From table 2 we can see that the difference in completeness rate is not so great as for accuracy and correctness, being less than 15%. This means the users completed similar amounts of the policy but with a much lower accuracy and correctness.

| Trails | Completeness | Accuracy | Correctness |
|---|---|---|---|
| Group A first trials with GUI | 62%(±30%) | 47%(±34%) | 35%(±37%) |
| Group A second trials with CNL | 69%(±36%) | 63%(±21%) | 49%(±27%) |
| Group B first trials with CNL | 74%(±28%) | 81%(±13%) | 63%(±33%) |
| Group B second trials with GUI | 85%(±24%) | 83%(±19%) | 76%(±29%) |

**Table 2 - Effectiveness**

Accuracy was measured as the number of correct elements out of all elements completed by the users. For example, if a user created 10 policy elements but only 6 of them correctly match the reference policy, then the accuracy is 60%. The result shows that for first time users, the accuracy of the CNL interface is much higher than that of the GUI; 81% compared to 47%. This can be explained in two ways. Firstly, users tend to structure sentences in a logical and sequential way, when they enter the information with the CNL interface. However, with the GUI, it is hard for novice users to find the logic and sequence of labels, boxes and buttons. Hence it implies more chance of mistakes. Secondly, the parser behind the CNL interface can only accept correctly structured sentences. Consequently this filters out some mistakes before the final policy is generated.

Even so, users still did not manage to create 100% complete or accurate policies with the CNL interface. Many redundant/useless elements were created containing the terms "administrator" and "manager". Some users tried to define the administrator as a role using the CNL interface, whilst very few users did that with the GUI (6 compared to 1). Probably this is because "administrator" is a general and flexible concept in natural language, whilst it already exists as a labelled tab in the GUI so does not need defining. This will be explored in more detail in a later section. The term "manager" is not mentioned anywhere in the scenario description, but many users still tried to define it in the CNL interface, which contributed to the inaccuracy. We suspect this is either because the users saw the word "manager" in the example sentences in the right hand panel, or they think that "manager" and "staff" are mandatory roles in all organisations, besides the roles that were defined in the scenario.

### 8.3.2 Efficiency

Efficiency is a measure of how much effort a user spent on completing a task. In our case we use the time spent as a measure of the effort. In most cases users never fully completed the task of policy creation, so the total time used is an estimate based on the actual time spent divided by the completeness. For first time users, the average time for completing a policy with the CNL interface was 56 ± 25 minutes, with times ranging from 3 hours 54 mins to 8 mins, while for the GUI it was 1 hour 14 mins ± 41 mins with times ranging from 3 hours 9 mins to 15 mins. This means that on average users take 32% less time to complete a policy with the CNL interface, compared with using the GUI.

This difference can be explained by observing the users' behaviour to see if they have understood what they have done. When users use the CNL interface they enter a sentence and then read it back to themselves to see if what they have entered is correct. If it is not, they can easily edit the sentence. However, with the GUI, the users appeared to have a poor understanding of their inputs. This is because they had to type a string into a field, then press Enter, and the GUI would display some information back to them in another part of the GUI screen, with perhaps a natural language sentence at the bottom of the screen to describe the information they had input. When the users do not quite understand what they have done, it appears to be harder for them to interpret the feedback from either the GUI components or the natural language displayed in the bottom panel. So even when users had input information correctly, they might subsequently amend it (wrongly) during their later actions. Users would hardly ever (incorrectly) amend correct sentence with the CNL interface.

From the trials we can see that almost all users made an improvement in the second round of the trails, no matter in which order they tried the two interfaces. These improvements are obviously due to their experiences of using the software in the first round. However, the improvements differ depending of the order in which they used the two interfaces. For group A, who used the GUI in the first trail and then used the CNL interface in the second trail, the completeness increased by 7%, the accuracy increased by 16%, and the correctness increased by 14%. For group B, who used the two interfaces in the opposite order, the completeness increased by 11% in the second trail, the accuracy by 2% and the correctness by 13%.

| Metric | GUI | CNL |
|---|---|---|
| Relative Ease of Completion | 1 | 1.32 |
| Relative Ease of Learning | 1 | 2.26 |
| Transferrable Knowledge (Speed) | -7% | 37% |
| Transferrable Knowledge (Accuracy) | -29% | 75% |

**Table 3 - Efficiency**

We also wanted to measure how much transferrable knowledge, if any, users learned from each interface. So for the CNL interface, we studied if there is any difference between users who used it prior to using the GUI (group B) and those who didn't (group A). The result shows that compared with group A, group B have 23% increase in completeness, 36% increase in accuracy, and 41% increase in correctness when using the GUI interface. This is because group B had already seen the results from the CNL interface several times[5]. More importantly, a map between their own terms/logic and the terms/logic of the GUI has been built up in this process. So they had a better understanding of the labels and boxes in the GUI, compared with those users who saw the GUI for first time without the benefit of the CNL interface first.

However, for the GUI interface, the same learning effect wasn't found. The results showed that if the users tried the GUI before using the CNL interface, then they performed worse with the CNL interface than the users who used the CNL interface without having used the GUI first. It looks as if either the GUI made the users more confused about the terms and logic or that the two groups of users were not comparable and therefore the cross group comparison is not valid. However, we tried to match the two groups of users demographically, as we described earlier in section 8.1.

### 8.3.3 Satisfaction

Both groups of users are more satisfied with the CNL interface than the GUI. The satisfaction was measured with the PSSUQ questionnaire from IBM, and the scale ranges from 1 (no effort to use) to 7 (not usable). The results of overall satisfaction of the CNL interface and the GUI were 3.01 and 3.87 respectively. The CNL interface out performs the GUI in all three sub areas, especially in the system usefulness area. The CNL interface fell below 3 in the information quality sub area only. Some users reported that the CNL interface was easy to use in general, but when a problem happened e.g. a sentence could not be parsed, there was not enough information to fix the problem.

[5] When the CNL interface has completed and turned the input policy into XML, it is then displayed via the GUI interface so that users can modify the final result. Consequently the CNL users become somewhat familiar with the GUI whilst using the CNL interface.

| Metric | GUI | CNL |
|---|---|---|
| System Usefulness | 4.14(±1.60) | 2.98(±1.33) |
| Information Quality | 3.77(±1.32) | 3.09(±1.06) |
| Interface Quality | 3.42(±1.39) | 2.93(±1.18) |
| Overall Satisfaction | 3.87(±1.35) | 3.01(±1.07) |

**Table 4 - Satisfaction**

Interestingly, when we divided the users into two groups by the sequence of using the two interfaces, we found that user satisfaction is greatly affected by the sequence. Table 5 shows the satisfaction score differences between the GUI and the CNL interface. Group A used the GUI in the first trails, and then used the CNL interface in the second trails. They felt that the GUI was not so usable, and were much more satisfied (1.86 difference) with the CNL interface. However, for Group B users who used the CNL in the first trails, and the GUI in the second trials, they felt that the CNL was only slightly better (0.47 difference) than the GUI. They rated the quality of the two interfaces quite closely. The results clearly show that users feel it is more difficult to start by using the GUI, so the CNL interface is a good starting point to get familiar with the policy creating software. Users may turn to the GUI for authoring more sophisticated policies, whilst the CNL interface provides a good introductory way of using the GUI.

| Metric | Group A (GUI) | Group A (CNL) | Group B (CNL) | Group B (GUI) |
|---|---|---|---|---|
| System Usefulness | 4.89(±1.57) | 3.03(±1.51) | 2.94(±1.33) | 3.39(±1.64) |
| Information Quality | 4.26(±1.34) | 3.50(±1.21) | 2.67(±0.97) | 3.29(±1.43) |
| Interface Quality | 3.83(±1.44) | 3.13(±1.24) | 2.73(±1.27) | 3.00(±1.47) |
| Overall Satisfaction | 4.46(±1.30) | 3.22(±1.25) | 2.81(±1.06) | 3.28(±1.47) |

**Table 5 - Satisfaction Affected by the Sequence of Using the Two Interfaces**

## 8.4 Insights

### 8.4.1 What or who is an administrator? Is it a superior role, a super user or a Source Of Authority?

The concept of "administrator" is a pre-defined concept in the software. It appears as a label on a GUI tab and can be used directly in the CNL sentences without prior user definition. However, more than one user tried to define it in the CNL with sentences such as "Administrator is a role", "Administrators are users". Some users also tried to put it in the role hierarchy, by "Administrator is superior to clerks and process owners". All of these sentences are correct in the general context of natural language. So it is natural for users to write these sentences. However, "administrator" does appear in the example text in the right hand panel of the CNL interface, and there is no sentence in which it has been defined. This is because in the underlying ontology, "administrator" is defined as a sibling class of "role". So any CNL sentence which tries to (re)define administrator as a class will fail to be parsed. Hence the users who did this got confused. They didn't know how to correct the parsing failure, since their sentences looked correct.

This reveals a general problem that the users are not aware of. They are unaware of which terms have already been defined, and which terms need to be defined by them. It would be possible to list all the predefined terms in the right hand panel. In fact this was an option of the initial design of the CNL interface. However it was omitted in order to keep the example area compact, otherwise users are more likely to ignore examples that are more than 1 page long and need scrolling down.

There are two possible ways of addressing this problem. One is to provide a Help menu with a section listing all the predefined terms. Another is to give a specific error message when users try to redefine a built-in class or instance e.g. "The term 'Administrator' has already been defined. Please delete your sentence." We could add a string description property to each pre-defined class/instance, and then insert this description into the standard error message. For example, "The term 'Administrator' has already been defined as *an authority who can assign roles to users from domains*. Please delete your sentence."

### 8.4.2 Assign is defined as a new action

Similar to 8.4.1, several users of the CNL interface tried to define the "assign" action for administrators, with the sentence "Assign is an action". This is another case of users not knowing the pre-defined terms. It is quite hard for users to tell which terms are pre-defined and which terms need to be defined, based on the example policy shown in the right hand panel.

Besides the solution proposed in 8.4.1, another way which may help is to improve the example sentences in the right hand display. One user suggested that a graphical illustration is added under or beside the current text, which depicts the scenario, in a similar way to how the trail scenario was shown. He said this will make it easier for users to learn how the example text works to solve the problem for a given scenario. The downside of this is that it leads to the same problem of extending the length of the example area, which may reduce its readability. A workaround could be to show the graphical illustration in another tab in the example area, so that users may chose to show the text or the illustration as they want, without extending the length of the example area. Another way to improve the example text is to add information to it by formatting it in different ways. For example, we could highlight or bold the user defined terms, to make them stand out. This would ensure that "print" and "assign" are shown in different formats, and then hopefully, this would make users aware that these two terms are different. Also, bolded or highlighted words should attract the users' attention more, and help them to understand how to use the terms from the example scenario shown in the graphical illustration.

### 8.4.3 Access is defined/used as a general action

"Access" is another term which users tried to define as an action. However, the reason behind this is different to the reason behind defining the assign action or the administrator role/user. "Access" is neither a specified action in the scenario, nor a pre-defined term in the built-in ontology. "Access" is a generic term which implies all types of action. Users invented this term while reading the scenario. Most of the users who defined the "access" action used it instead of the "read" action, but some users tried to use the "access" action for all actions in the scenario, i.e. read, add and change. It looks as if the users treated the "access" action as a general term for any type of action on a resource, with read being the most common one ("if you cannot read it, you cannot access it"). This is probably explained by the fact that the trial users were not security specialists, and in many cases not IT specialists, and therefore they didn't understand the scenario or access controls at a level which is specific enough to differentiate between "access" and "read". Although most users thought the scenario was quite simple, and all of them confirmed that they fully understood the scenario before taking part in the trails, never the less, the fine nuances needed by the access control policy were beyond their appreciation.

One solution to this problem could be to pre-define "access" as a synonym of All_Actions, so that whenever access is mentioned in a sentence the ontology knows what it means. For example "Managers are allowed to access all files" would be interpreted as "Managers are allowed to perform all defined actions on all files". When specific actions are defined by the users, and used in sentences, these would be used instead of the generic all actions access mode. However, the downside of making this change would be for those novice users who equate "access" with "read" mode. In this case, if they gave managers "access" to a resource, they would think they were giving read access whilst the software would in fact give all modes of access to the manager, and this is clearly a breach of security. So here we have a real conflict of interest between usability and security for novice users, which can only be solved by user education.

### 8.4.4 "Clerks have access to all files"

In a similar vein to above, two users tried to define access rules with sentences like "Clerks have access to files", when they had not previously defined "access" as an action. It is a problem similar to 8.4.3, but it is different conceptually. A "permission" is defined as an action on a resource. In 8.4.3 users tried to define a generic action termed "access," but here they are trying to define a generic permission using the generic "access" action. This implies a permission to perform all actions on a resource. In order to cater for this, we would need to change the allowed syntax to the following:

<sentence> ::= <subject> [<have> | <can> ] [ [<action> "access" ] | [<action>] | [ "access" ] ] [<preposition>] <resource>

This would allow users to specify either an action, or an action followed by the word "access" or simply the word "access". This would allow sentences such as:

"Clerks have read access to all files."

"Clerks can read all files"

"Clerks have access to all files"

When the specific action is omitted, a permission instance with all defined actions on that resource would be created and then associated to the subject instance.

### 8.4.5 Match "supplier list" with "the supplier list file"

In the CNL interface, if a user defined a file as a type of resource and then defined a "Supplier list" as a file, entries for both file and "supplier list" would be created in the ontology. If the user subsequently stated a rule such as "staff can read the supplier list file", this sentence cannot be parsed, since "the supplier list file" cannot be matched with either the "supplier list" nor the "file". With the current parser, it can match words in different forms, such as between the singular form and the plural from ("supplier list" and "supplier lists"), but it cannot match a composite term ("supplier list file") created from two different terms ("supplier list" and "file").

A solution to this problem is to add a synonym property to each instance, where the synonyms are "instance name + class name". To make this more generic, each superior class in the class hierarchy would be used to create synonyms. For example in the case of "supplier list", two synonyms are created, which are "supplier_list_file" and "supplier_list_resource", since file is a subclass of resource. Using this model, the term "staff role" would then match with the role instance "staff". Similarly both "the supplier list file" or "the supplier list resource" would match with the file resource instance named "supplier list".

### 8.4.6 "Files can be read and written"

One user tried to define permissions i.e. associate actions with resources, using the sentence "Files can be read and written". This is in a nicer and more natural way than how it is done with the current CNL parser. In the current implementation, permissions (in fact all relationships) are defined using the verb "have" in the sentence <definedTerm> "have" <definedTerm> e.g. "Files have read and write", which is incorrect in English. The following syntax should be added to support the more natural usage:

<sentence> ::= <resource> <can> ["be"] <action> [<preposition>].

Files can be read and written.

Printers can be printed on.

Printers can print.

### 8.4.7 "John is an administrator in the business department"

More than one user tried to define an administrator for a subject domain as follows, e.g. "John is an administrator in the business department". There are two possible ways to interpret this sentence: "John is an administrator, and he is allowed to assign *some* roles (but none specified) to users from the business department" or "John is an administrator, and he is allowed to assign *all* roles to users from the business department". So this is a sentence with ambiguity.

This is similar to the existing sentence "I trust John to say who clerks are", where user domains are missing. This is supported by the current CNL implementation and we assume that the policy writer implies all users domains. We can use the same technique when defining administrators. We can add the following syntax:

<sentence> ::= <phrase> "be" "administrator" <proposition> <userDomain>

John is an administrator in the business department.

The semantic is to create an administrator instance with the name of *<phrase>*. An instance of role assignment is created with *<userDomain>* only, and then associated with the administrator instance. We can name this an **open** role assignment instance. We can define an open role assignment instance as one in which either the role or the user domain is not specified (i.e. left open). An open role assignment with a user domain but no roles can be refined by open role assignments with roles but no user domain, and vice versa. In the example above, when the final policy is exported, if there is an open role assignment instance with no roles, then this role assignment instance will be interpreted as all roles. Similarly, if there is an open role assignment instances with no user domain, then this will be interpreted as all user domains.

For example, if a user types "John is an administrator in the business department", and there are no other statements regarding John, then John is allowed to assign all roles to the users from the business department. Similarly, if a user types "I trust John to say who clerks are" and there are no other statements regarding John, then John is allowed to assign the clerk role to all users. But if the above two sentences appear in the same text for a policy, then it is equivalent to the following two sentences: "John is an administrator" and "John is allowed to assign the clerk role to users from the business department".

### 8.4.8 "Clerks in the business department can …"

A user tried to type the following sentence "Clerks in the business department can read all files". This sentence failed to be parsed since the user previously defined the role as simply "Clerk"; hence the role "Clerks in the business department" was not recognised. Finally this user managed to create a correct policy by defining the whole phrase "clerks in the business department is a role". This introduced an interesting new concept: roles in user domains, which is not part of the standard RBAC model.

In the standard RBAC model, roles and users are different concepts but there is no concept of a user domain, since the standard RBAC model assumes one single organisational domain that encapsulates all concepts (users, roles, actions, resources etc.) However, in natural language it does make sense to link roles to domains e.g. "clerks in the business department" or "technicians from IT services". In natural language this means a sub set of clerks or technicians. This can be mapped to sub classes in the ontology model or superior roles in the hierarchical RBAC model. This would require a new syntax for parsing sentences containing roles in user domains, of:

<role> <preposition> <userDomain>

and creating a sub class of the <role>, called <*role_in_domain*>. However, we are not sure what demand there would be for this feature, as there may not be many use cases that require it.

### 8.4.9 Users tended to ignore the example policy

We found it is important to explicitly tell users to follow the example sentences. During the initial pilot trials, two users complained that "… no one told me that I should strictly follow the examples provided" and "… there should be instructions along with the examples". So in the software for the real user trials, a dialog of instructions is shown when the CNL interface is loaded for the first time, and the user has to click a button to close it. This forces users to have a look at it before they use the CNL interface. In the main trials, more than half of the total users stopped and read the dialog and then went on to read the example text. However, there were still quite a few users who skipped the initial dialog by clicking the OK button immediately. We suggest that this may be due to "pop up fatigue" due to too many popup dialog boxes being in use in today's software.

# 9 Conclusions

## 9.1 Conclusion

In this document, we have described an approach to authoring access control policies using a sub set of English called Controlled Natural Language (CNL). CNL reduces the complexity of natural language and removes ambiguities, thus making it relatively easy for a computer to capture the precise semantics of the policy writer. The policy in CNL can be converted to a policy in one of several machine language formats, so that it can be automatically enforced by a policy enforcement point (PEP) and policy decision point (PDP).

The design and implementation of the CNL policy authoring tool have been explained. The design is modular so that it clearly separates data from the processing modules. A set of APIs have been specified so that new modules can be added or existing modules can be extended in functionality or replaced.

A test plan was developed in order to evaluate the CNL interface, and a series of user trials were performed. Analysis of the results showed that:

a) the CNL interface is more effective for first time users than the GUI, and these users create more complete and much more accurate and correct policies than do the GUI users;

b) the CNL interface is more efficient for first time users than the GUI, with users completing their policies more quickly and more accurately;

c) the CNL interface is better at transferring knowledge to users than the GUI, whilst the GUI actually had a negative effect on transferring knowledge to the users;

d) users are more satisfied overall with the CNL interface than with the GUI. Not only that, but using the CNL interface first before the GUI actually increases the user satisfaction with the GUI as well;

e) the user trials highlighted areas in which improvements still need to be made to the CNL interface, due to the very many different ways that natural language can be used to specify the same fact or rule. The current interface only supports the most common subset of these, and therefore more syntaxes and alternate ways of specifying the same access control rules need to be added;

f) Complex policy rules containing conditions and/or obligations are still not supported, and since these sentences are more complex than the basic ones currently supported, the amount of research and development effort that is still needed in order to add these features is at least equal to the effort expended so far.

Overall these are very positive results and show that with further research and development work, natural language policy interfaces will substantially ease the burden of creating security policies.

## 9.2 Use in TAS3

The CNL interface was used by the application developers as an aid to formulating the organisation's access control policies that were built into the application pilots. However, because these policies typically were more complex that those supported by the current CNL interface e.g. they contained obligations and conditions, the CNL interface on its own was insufficient to create a complete organisational policy. Whilst it was capable of creating a basic access control policy, further editing was subsequently necessary. Consequently the developers bhad to revert to using either the PERMIS Policy Editor GUI (for PERMIS XML policies) or a tool such as the UMU XACML editor[6] (for XACML policies) or to hand editing the XML for both policy languages. None of these are as easy to use as the CNL interface, though they support greater functionality.

Because the CNL interface is application independent, it is not tailored to any specific pilot application or use case. Consequently the human user has to populate the underlying ontology with application specific classes and instances, such as names of resources, actions and roles. Any strings are allowed by the CNL interface, so this allows uneducated users to input garbage such as "Teachers can hit pupils" or "read is an action. reada is a role. readb is a resource. reada can read readb". This was seen to be too flexible for pilot application users, and therefore a very simple matrix GUI is currently being developed for end users, which contains fixed values for roles, actions and resources, allowing users to simply tick a few boxes to express their policy preferences. Early work by Kent and Koblenz has shown that if this matrix is converted into CNL sentences of the type "<role X> can <action> <resource>, and the CNL ontology is pre-populated with the definitions of the roles, actions and resources, then the CNL interface is capable of producing correctly formatted XACML or PERMIS policies for submission to the authorisation system.

## 9.3 Future Work

Further development work on the CNL interface will be terminated at the end of Month 42, apart from ensuring that the ontology is fully aligned with the ontology model in D2.2 [10], **D2.3 [11] and D 7.1 [9].** The only work remaining after that will be to fix any bugs which are notified to the developers by the worldwide community of users, as well as publishing papers and completing documentation of the research in a PhD thesis.

[6] Available from http://sourceforge.net/projects/umu-xacmleditor/

# 10 References

[1] Gollmann, D. Computer security, John Wiley &; Sons, Inc. 1999

[2] ANSI. "Information technology - Role Based Access Control". ANSI INCITS 359-2004

[3] Wang, L., D. Wijesekera, et al.. A logic-based framework for attribute based access control. Proceedings of the 2004 ACM workshop on Formal methods in security engineering. Washington DC, USA, ACM: 45-55.

[4] EU IST "Semantic Knowledge Technologies." 2006. from http://www.sekt-project.com/.

[5] Inglesant, P., M. A. Sasse, et al. Expressions of expertness: the virtuous circle of natural language for access control policy specification. Proceedings of the 4th symposium on Usable privacy and security. Pittsburgh, Pennsylvania, 2008. ACM: 77-88.

[6] Web Ontology Working Group. OWL Web Ontology Language, W3C. 10 February 2004.

[7] D.W.Chadwick, A. Otenko. "RBAC Policies in XML for X.509 Based Privilege Management" in Security in the Information Society: Visions and Perspectives: IFIP TC11 17th Int. Conf. On Information Security (SEC2002), May 7-9, 2002, Cairo, Egypt. Ed. by M. A. Ghonaimy, M. T. El-Hadidi, H.K.Aslan, Kluwer Academic Publishers, pp 39-53.

[8] OASIS "eXtensible Access Control Markup Language (XACML) Version 2.0"

OASIS Standard, 1 Feb 2005

[9] TAS3. Design of Identity Management, Authentication and Authorization Infrastructure. TAS3 Deliverable D7.1. Dec 2009.

[10] TAS3. Common Upper Ontologies. TAS3 Deliverable D2.2. <date>

[11] TAS3. Lower Upper Ontologies. TAS3 Deliverable D2.3. <date>

[12] Apache. "Unstructured Information Management Architecture (UIMA)." from http://uima.apache.org/.

[13] Cunningham, H., D. Maynard, et al. GATE: A framework and graphical development environment for robust NLP tools and applications. Proceedings of the 40th Annual Meeting of the ACL. 2002.

[14] Funk, A. D., Brian; Tablan, Valentin; Bontcheva, Kalina; Cunningham, Hamish. Controlled Language IE Components version 2. January 2007

[15 ISO. "ISO 9241-11 (1998) Ergonomic requirements for office work with visual display terminals (VDTs) -- Part 11: Guidance on usability." from http://www.iso.org/iso/catalogue_detail.htm?csnumber=16883.

[16] Lewis, J. R. "Psychometric Evaluation of the PSSUQ Using Data from Five Years of Usability Studies." International Journal of Human-Computer Interaction 14(3), 2002: 463 - 488.

[17] Fuchs, N. E., U. Schwertel, et al. Attempto Controlled English - Not Just Another Logic Specification Language. Proceedings of the 8th International Workshop on Logic Programming Synthesis and Transformation, Springer-Verlag, 1990: 1-20.

[18] Gamma, E, et al. Design Patterns. s.l. : Addison-Wesley Professional. ISBN 978-0201633610. November 1994.

[19] Virzi, R. A. . "Refining the test phase of usability evaluation: how many subjects is enough?" Hum. Factors 34(4): 457-468. 1992.

[20] Hwang, W. and G. Salvendy "Number of people required for usability evaluation: the 10±2 rule." Commun. ACM 53(5): 130-133. 2010.

[21] OASIS 2005. Core and hierarchical role based access control (RBAC) profile of XACML v2.0.

# 11 Appendix A – Modifications to CLOnE API

boolean createOrLinkClass(Ontology ontology,

Annotation classChunk, Annotation parentChunk)

This method creates a new class with the specified parent in an existing ontology, or adds a parent to an existing class in the ontology. If the classChunk's class does not exist and the parentChunk is null, then the method creates the new class directly under the top class. The function of this method has been enhanced, so that when a class is created, its generic instance is created at the same time, with the name "All_<class>".

boolean createInstance(Ontology ontology,

Annotation classChunk, Annotation instanceChunk)

boolean createInstance(Ontology ontology,

Annotation classChunk, String instanceName)

boolean createInstance(Ontology ontology,

String className, Annotation instanceChunk)

boolean createInstance(Ontology ontology,

String className, String instanceName)

This group of methods create a new instance in the ontology. The first method in the group exists in the original API, while the later three methods have been added to accept string names for either the class or instance or both.

OInstance getClassInstance(Ontology ontology, Annotation classChunk)

OInstance getClassInstance(Ontology ontology, String className)

These two methods have been added to get the generic instance (All_<class>) of a class, using either a string or an annotation from the input text.

boolean createOrSetObjectProperty(Ontology ontology,

Annotation domainChunk, String middlePart, Annotation rangeChunk)

boolean createOrSetObjectProperty(Ontology ontology,

String domainName, String middlePart, Annotation rangeChunk)

boolean createOrSetObjectProperty(Ontology ontology, Annotation domainChunk, String middlePart, String rangeName)

boolean createOrSetObjectProperty(Ontology ontology,

String domainName, String middlePart, String rangeName)

This group of methods create a property between two classes. Three new methods have been added to the original one, in order to accept strings as class names for either the domain, range or both.

boolean setObjectProperty(Ontology ontology,

Annotation domainChunk, String middlePart, Annotation rangeChunk)

boolean setObjectProperty(Ontology ontology,

String domainName, String middlePart, Annotation rangeChunk)

boolean setObjectProperty(Ontology ontology,

Annotation domainChunk, String middlePart, String rangeName)

boolean setObjectProperty(Ontology ontology,

String domainName, String middlePart, String rangeName)

This group of methods is used to set a property (or a relation, in ontology terms) between two instances. The original API only accepted annotations as object names, and it has been extended to accept strings for either or both of the domain and the range.

boolean setClassHasInstanceProperty(Ontology ontology,

Annotation domainChunk, String middlePart, Annotation rangeChunk)

boolean setClassHasInstanceProperty(Ontology ontology,

String domainName, String middlePart, Annotation rangeChunk)

boolean setClassHasInstanceProperty(Ontology ontology,

Annotation domainChunk, String middlePart, String rangeName)

boolean setClassHasInstanceProperty(Ontology ontology,

String domainName, String middlePart, String rangeName)

boolean setInstanceHasClassProperty(Ontology ontology,

Annotation domainChunk, String middlePart, Annotation rangeChunk)

boolean setInstanceHasClassProperty(Ontology ontology,

String domainName, String middlePart, Annotation rangeChunk) boolean setInstanceHasClassProperty(Ontology ontology,

Annotation domainChunk, String middlePart, String rangeName)

boolean setInstanceHasClassProperty(Ontology ontology,

String domainName, String middlePart, String rangeName)

These 8 new methods have been added to link a class with an instance. The generic instance (All_<class>) of the class parameter is used to create the property.

# 12 Appendix B – The PSSUQ questionnaire

Based on your experience with similar products, please rate your agreement with the following statements about how you feel in general. Just circle or X out the level of agreement that applies (where 1 means strongly agree, 4 means neither agree nor disagree, and 7 means strongly disagree), as in the example.

Strongly Agree 1---2---3---X---5---6---7 Strongly Disagree

**********Statements about the product you used**********

1. Overall, I am satisfied with how easy it is to use this system.

Strongly Agree 1---2---3---4---5---6---7 Strongly Disagree

2. It was simple to use this system.

Strongly Agree 1---2---3---4---5---6---7 Strongly Disagree

3. I could effectively complete the tasks and scenarios using this system.

Strongly Agree 1---2---3---4---5---6---7 Strongly Disagree

4. I was able to complete the tasks and scenarios quickly using this system.

Strongly Agree 1---2---3---4---5---6---7 Strongly Disagree

5. I was able to efficiently complete the tasks and scenarios using this system.

Strongly Agree 1---2---3---4---5---6---7 Strongly Disagree

6. I felt comfortable using this system.

Strongly Agree 1---2---3---4---5---6---7 Strongly Disagree

7. It was easy to learn to use this system.

Strongly Agree 1---2---3---4---5---6---7 Strongly Disagree

8. I believe I could become productive quickly using this system.

Strongly Agree 1---2---3---4---5---6---7 Strongly Disagree

9. The system gave error messages that clearly told me how to fix problems.

Strongly Agree 1---2---3---4---5---6---7 Strongly Disagree

10. Whenever I made a mistake using the system, I could recover easily and quickly.

Strongly Agree 1---2---3---4---5---6---7 Strongly Disagree

11. The information (such as on-line help, on-screen messages and other documentation) provided with this system was clear.

Strongly Agree 1---2---3---4---5---6---7 Strongly Disagree

12. It was easy to find the information I needed.

Strongly Agree 1---2---3---4---5---6---7 Strongly Disagree

13. The information provided for the system was easy to understand.

Strongly Agree 1---2---3---4---5---6---7 Strongly Disagree

14. The information was effective in helping me complete the tasks and scenarios.

Strongly Agree 1---2---3---4---5---6---7 Strongly Disagree

15. The organization of information on the system screens was clear.

Strongly Agree 1---2---3---4---5---6---7 Strongly Disagree

16. The interface of this system was pleasant.

Strongly Agree 1---2---3---4---5---6---7 Strongly Disagree

17. I liked using the interface of this system.

Strongly Agree 1---2---3---4---5---6---7 Strongly Disagree

18. This system has all the functions and capabilities I expect it to have.

Strongly Agree 1---2---3---4---5---6---7 Strongly Disagree

19. Overall, I am satisfied with this system.

Strongly Agree 1---2---3---4---5---6---7 Strongly Disagree

# 13 Appendix C – The Test Scenario

This is a simple scenario of the security requirements of a hypothetical organisation. It is for illustration only and should not be taken to represent the situation in any real-world organisation.

In this organisation, there are two departments, the Business Department and the Analysis Department. There are three databases containing sales information, marketing information and supplier information.

Users in the organisation are restricted in what they are allowed to do with the three databases depending upon their *roles* in the organisation.

**Analysts** can only read the sales database and the marketing database. They cannot access the suppliers database

**Clerks** can add and change information in all three databases.

**Process owners** cannot change any data but can read all the information in all three databases.

Clerks and Process owners are in the Business Department.

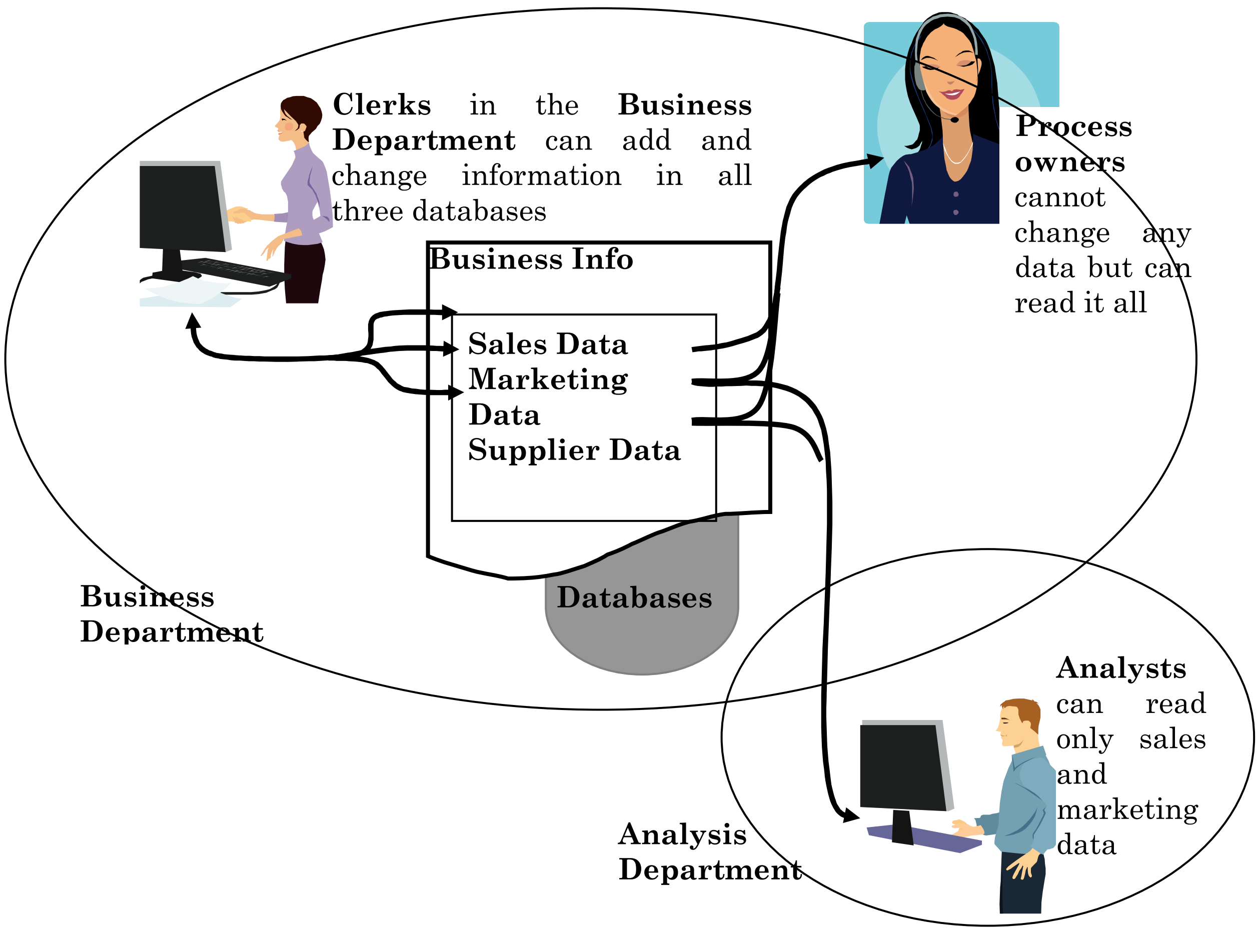
Analysts are in Analysis Department.

Each department has an administrator who can add and remove people to and from the various roles:

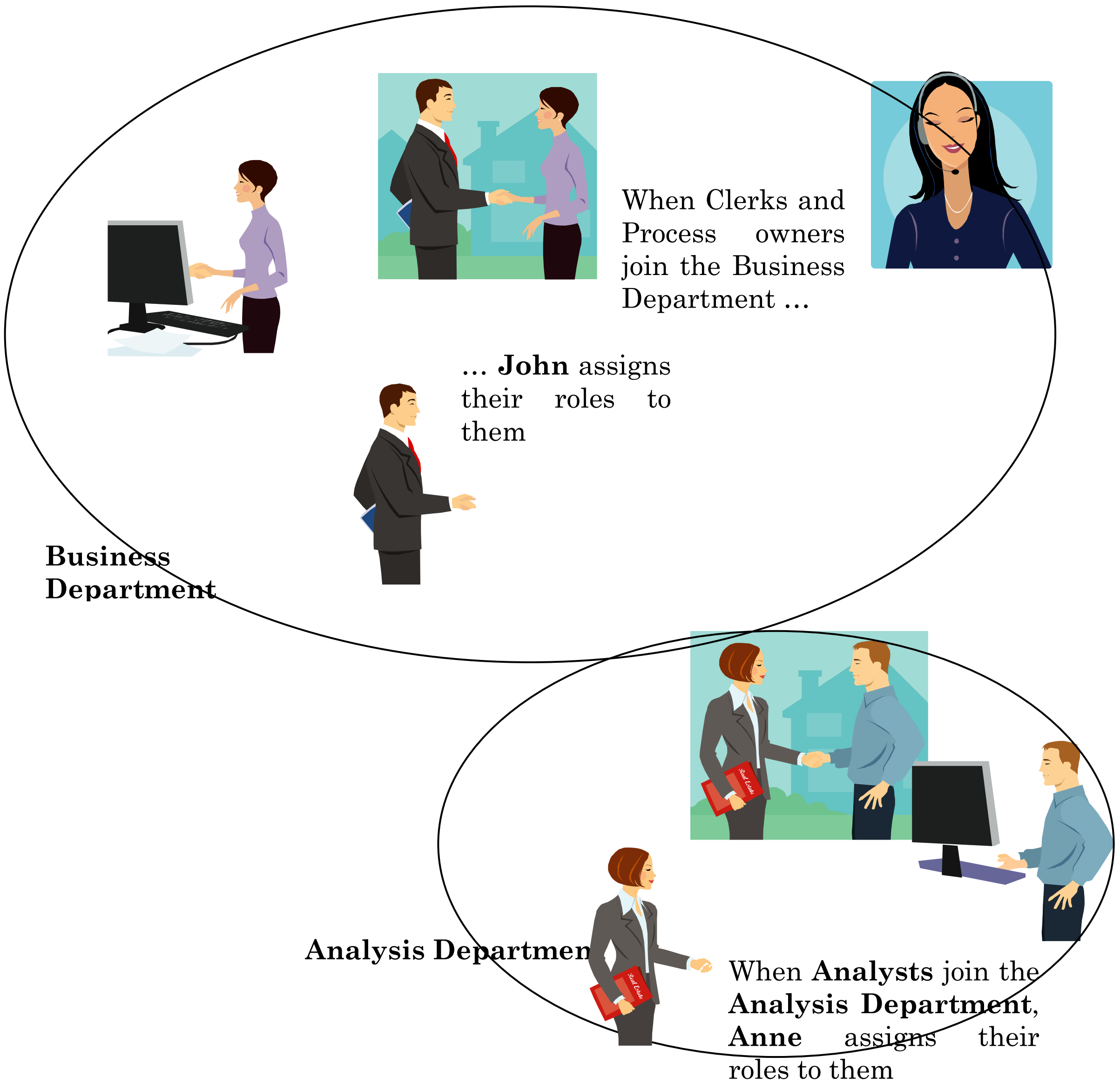
John is the administrator in the Business Department.

Ann is the administrator in the Analysis Department.

Users cannot see the whole of the **Databases**; what they can see depends on their **roles**:


**Clerks** in the **Business Department** can add and change information in all three databases
Business Info
Sales Data
Marketing Data
Supplier Data
Databases
**Process owners** cannot change any data but can read it all
Business Department
**Analysts** can read only sales and marketing data
Analysis Department

Their roles are assigned to them by administrators (John and Anne):


When Clerks and Process owners join the Business Department …
… **John** assigns their roles to them
**Business Department**
**Analysis Department**
When **Analysts** join the **Analysis Department**, **Anne** assigns their roles to them

# 14 Appendix D – Ontology Template of Access Control Policy

```
<?xml version="1.0" encoding="UTF-8"?>

<rdf:RDF

        xmlns="http://sec.cs.kent.ac.uk/permis/policy#"

        xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"

        xmlns:xsd="http://www.w3.org/2001/XMLSchema#"

        xmlns:owl="http://www.w3.org/2002/07/owl#"

        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

        xmlns:protons="http://proton.semanticweb.org/2005/04/protons#"

        xmlns:protont="http://proton.semanticweb.org/2005/04/protont#">


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#PolicyOntology">

        <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Ontology"/>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Permission_has_Condition">

        <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Condition_has_Expression">

        <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>

</rdf:Description>
```

```
<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#All_Condition">

	<rdf:type rdf:resource="http://sec.cs.kent.ac.uk/permis/policy#Condition"/>

</rdf:Description>


<rdf:Description rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Condition">

	<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Administrator_has_Role_Assign
ment_Permission">

	<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Role_Assignment_Permission_h
as_Role">

	<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Role_Assignment_Permission_h
as_User">

	<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#All_Role_Assignment_Permissio
n">

	<rdf:type
rdf:resource="http://sec.cs.kent.ac.uk/permis/policy#Role_Assignment_Permission
"/>
```

```
</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Role_Assignment_Permission">

        <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Role_Superior_to_Role">

        <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Parameter_has_Type">

        <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Action_has_Parameter">

        <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Resource_has_Action">

        <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Permission_has_Resource">

        <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
```

```
</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Permission_has_Action">

        <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Role_has_Permission">

        <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#User_has_Role">

        <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>

</rdf:Description>


<rdf:Description rdf:about="http://sec.cs.kent.ac.uk/permis/policy#All_Role">

        <rdf:type rdf:resource="http://sec.cs.kent.ac.uk/permis/policy#Role"/>

</rdf:Description>


<rdf:Description rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Role">

        <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>

</rdf:Description>


<rdf:Description rdf:about="http://sec.cs.kent.ac.uk/permis/policy#All_Action">

        <rdf:type rdf:resource="http://sec.cs.kent.ac.uk/permis/policy#Action"/>

</rdf:Description>


<rdf:Description rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Action">
```

```
        <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#All_Permission">

        <rdf:type
rdf:resource="http://sec.cs.kent.ac.uk/permis/policy#Permission"/>

</rdf:Description>


<rdf:Description rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Permission">

        <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#All_Parameter">

        <rdf:type
rdf:resource="http://sec.cs.kent.ac.uk/permis/policy#Parameter"/>

</rdf:Description>


<rdf:Description rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Parameter">

        <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Entity_has_Url">

        <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Entity_has_X500Name">
```

```
        <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</rdf:Description>


<rdf:Description rdf:about="http://sec.cs.kent.ac.uk/permis/policy#All_Resource">
        <rdf:type rdf:resource="http://sec.cs.kent.ac.uk/permis/policy#Resource"/>
</rdf:Description>


<rdf:Description rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Resource">
        <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#All_Administrator">
        <rdf:type
rdf:resource="http://sec.cs.kent.ac.uk/permis/policy#Administrator"/>
</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Administrator">
        <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
</rdf:Description>


<rdf:Description rdf:about="http://sec.cs.kent.ac.uk/permis/policy#All_User">
        <rdf:type rdf:resource="http://sec.cs.kent.ac.uk/permis/policy#User"/>
</rdf:Description>


<rdf:Description rdf:about="http://sec.cs.kent.ac.uk/permis/policy#User">
        <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
</rdf:Description>
```

```
<rdf:Description rdf:about="http://sec.cs.kent.ac.uk/permis/policy#All_Entity">

        <rdf:type rdf:resource="http://sec.cs.kent.ac.uk/permis/policy#Entity"/>

</rdf:Description>


<rdf:Description rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Entity">

        <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>

</rdf:Description>


<rdf:Description rdf:about="http://sec.cs.kent.ac.uk/permis/policy#My_Policy">

        <rdf:type rdf:resource="http://sec.cs.kent.ac.uk/permis/policy#Policy"/>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Policy_has_Name">

        <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>

</rdf:Description>


<rdf:Description rdf:about="http://sec.cs.kent.ac.uk/permis/policy#All_Policy">

        <rdf:type rdf:resource="http://sec.cs.kent.ac.uk/permis/policy#Policy"/>

</rdf:Description>


<rdf:Description rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Policy">

        <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Permission_has_Condition">
```

```
        <rdfs:domain
rdf:resource="http://sec.cs.kent.ac.uk/permis/policy#Permission"/>

</rdf:Description>



<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Condition_has_Expression">

        <rdfs:domain
rdf:resource="http://sec.cs.kent.ac.uk/permis/policy#Condition"/>

</rdf:Description>



<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Administrator_has_Role_Assign
ment_Permission">

        <rdfs:domain
rdf:resource="http://sec.cs.kent.ac.uk/permis/policy#Administrator"/>

</rdf:Description>



<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Role_Assignment_Permission_h
as_Role">

        <rdfs:domain
rdf:resource="http://sec.cs.kent.ac.uk/permis/policy#Role_Assignment_Permission
"/>

</rdf:Description>



<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Role_Assignment_Permission_h
as_User">

        <rdfs:domain
rdf:resource="http://sec.cs.kent.ac.uk/permis/policy#Role_Assignment_Permission
"/>

</rdf:Description>



<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Role_Superior_to_Role">
```

```
        <rdfs:domain rdf:resource="http://sec.cs.kent.ac.uk/permis/policy#Role"/>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Parameter_has_Type">

        <rdfs:domain
rdf:resource="http://sec.cs.kent.ac.uk/permis/policy#Parameter"/>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Action_has_Parameter">

        <rdfs:domain
rdf:resource="http://sec.cs.kent.ac.uk/permis/policy#Action"/>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Resource_has_Action">

        <rdfs:domain
rdf:resource="http://sec.cs.kent.ac.uk/permis/policy#Resource"/>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Permission_has_Resource">

        <rdfs:domain
rdf:resource="http://sec.cs.kent.ac.uk/permis/policy#Permission"/>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Permission_has_Action">

        <rdfs:domain
rdf:resource="http://sec.cs.kent.ac.uk/permis/policy#Permission"/>

</rdf:Description>
```

```
<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Role_has_Permission">

        <rdfs:domain rdf:resource="http://sec.cs.kent.ac.uk/permis/policy#Role"/>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#User_has_Role">

        <rdfs:domain rdf:resource="http://sec.cs.kent.ac.uk/permis/policy#User"/>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Entity_has_Url">

        <rdfs:domain
rdf:resource="http://sec.cs.kent.ac.uk/permis/policy#Entity"/>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Entity_has_X500Name">

        <rdfs:domain
rdf:resource="http://sec.cs.kent.ac.uk/permis/policy#Entity"/>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Policy_has_Name">

        <rdfs:domain rdf:resource="http://sec.cs.kent.ac.uk/permis/policy#Policy"/>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Permission_has_Condition">

        <rdfs:range
rdf:resource="http://sec.cs.kent.ac.uk/permis/policy#Condition"/>

</rdf:Description>
```

```
<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Condition_has_Expression">

        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>

</rdf:Description>



<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Administrator_has_Role_Assign
ment_Permission">

        <rdfs:range
rdf:resource="http://sec.cs.kent.ac.uk/permis/policy#Role_Assignment_Permission
"/>

</rdf:Description>



<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Role_Assignment_Permission_h
as_Role">

        <rdfs:range rdf:resource="http://sec.cs.kent.ac.uk/permis/policy#Role"/>

</rdf:Description>



<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Role_Assignment_Permission_h
as_User">

        <rdfs:range rdf:resource="http://sec.cs.kent.ac.uk/permis/policy#User"/>

</rdf:Description>



<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Role_Superior_to_Role">

        <rdfs:range rdf:resource="http://sec.cs.kent.ac.uk/permis/policy#Role"/>

</rdf:Description>



<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Parameter_has_Type">
```

```
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Action_has_Parameter">

        <rdfs:range
rdf:resource="http://sec.cs.kent.ac.uk/permis/policy#Parameter"/>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Resource_has_Action">

        <rdfs:range rdf:resource="http://sec.cs.kent.ac.uk/permis/policy#Action"/>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Permission_has_Resource">

        <rdfs:range
rdf:resource="http://sec.cs.kent.ac.uk/permis/policy#Resource"/>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Permission_has_Action">

        <rdfs:range rdf:resource="http://sec.cs.kent.ac.uk/permis/policy#Action"/>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Role_has_Permission">

        <rdfs:range
rdf:resource="http://sec.cs.kent.ac.uk/permis/policy#Permission"/>

</rdf:Description>
```

```
<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#User_has_Role">

        <rdfs:range rdf:resource="http://sec.cs.kent.ac.uk/permis/policy#Role"/>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Entity_has_Url">

        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Entity_has_X500Name">

        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Policy_has_Name">

        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Role_Assignment_Permission">

        <rdfs:subClassOf
rdf:resource="http://sec.cs.kent.ac.uk/permis/policy#Permission"/>

</rdf:Description>


<rdf:Description rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Resource">

        <rdfs:subClassOf
rdf:resource="http://sec.cs.kent.ac.uk/permis/policy#Entity"/>

</rdf:Description>
```

```
<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Administrator">

        <rdfs:subClassOf
rdf:resource="http://sec.cs.kent.ac.uk/permis/policy#Entity"/>

</rdf:Description>


<rdf:Description rdf:about="http://sec.cs.kent.ac.uk/permis/policy#User">

        <rdfs:subClassOf
rdf:resource="http://sec.cs.kent.ac.uk/permis/policy#Entity"/>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#All_Condition">

        <rdfs:label>all condition</rdfs:label>

        <rdfs:label>All condition</rdfs:label>

</rdf:Description>


<rdf:Description rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Condition">

        <rdfs:label>condition</rdfs:label>

        <rdfs:label>conditions</rdfs:label>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#All_Role_Assignment_Permissio
n">

        <rdfs:label>all role_assignment_permission</rdfs:label>

        <rdfs:label>All role_assignment_permission</rdfs:label>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Role_Assignment_Permission">
```

```
        <rdfs:label>role assignment permission</rdfs:label>

        <rdfs:label>Role Assignment Permission</rdfs:label>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Administrator">

        <rdfs:label>anyone</rdfs:label>

        <rdfs:label>Anyone</rdfs:label>

</rdf:Description>


<rdf:Description rdf:about="http://sec.cs.kent.ac.uk/permis/policy#User">

        <rdfs:label>anywhere</rdfs:label>

        <rdfs:label>Anywhere</rdfs:label>

</rdf:Description>


<rdf:Description rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Resource">

        <rdfs:label>anything</rdfs:label>

        <rdfs:label>Anything</rdfs:label>

        <rdfs:label>target</rdfs:label>

        <rdfs:label>targets</rdfs:label>

        <rdfs:label>Target</rdfs:label>

</rdf:Description>


<rdf:Description rdf:about="http://sec.cs.kent.ac.uk/permis/policy#All_Role">

        <rdfs:label>all role</rdfs:label>

        <rdfs:label>All role</rdfs:label>

</rdf:Description>


<rdf:Description rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Role">
```

```
        <rdfs:label>role</rdfs:label>

        <rdfs:label>roles</rdfs:label>

</rdf:Description>


<rdf:Description rdf:about="http://sec.cs.kent.ac.uk/permis/policy#All_Action">

        <rdfs:label>all action</rdfs:label>

        <rdfs:label>All action</rdfs:label>

</rdf:Description>


<rdf:Description rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Action">

        <rdfs:label>action</rdfs:label>

        <rdfs:label>actions</rdfs:label>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#All_Permission">

        <rdfs:label>all permission</rdfs:label>

        <rdfs:label>All permission</rdfs:label>

</rdf:Description>


<rdf:Description rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Permission">

        <rdfs:label>permission</rdfs:label>

        <rdfs:label>permissions</rdfs:label>

</rdf:Description>


<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#All_Parameter">

        <rdfs:label>all parameter</rdfs:label>

        <rdfs:label>All parameter</rdfs:label>
```

```
</rdf:Description>

<rdf:Description rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Parameter">
        <rdfs:label>parameter</rdfs:label>
        <rdfs:label>parameters</rdfs:label>
</rdf:Description>

<rdf:Description rdf:about="http://sec.cs.kent.ac.uk/permis/policy#All_Resource">
        <rdfs:label>all resource</rdfs:label>
        <rdfs:label>All resource</rdfs:label>
</rdf:Description>

<rdf:Description rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Resource">
        <rdfs:label>resource</rdfs:label>
        <rdfs:label>resources</rdfs:label>
</rdf:Description>

<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#All_Administrator">
        <rdfs:label>all administrator</rdfs:label>
        <rdfs:label>All administrator</rdfs:label>
</rdf:Description>

<rdf:Description
rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Administrator">
        <rdfs:label>administrator</rdfs:label>
</rdf:Description>

<rdf:Description rdf:about="http://sec.cs.kent.ac.uk/permis/policy#All_User">
```

```
        <rdfs:label>all user</rdfs:label>
        <rdfs:label>All user</rdfs:label>
</rdf:Description>

<rdf:Description rdf:about="http://sec.cs.kent.ac.uk/permis/policy#User">
        <rdfs:label>user</rdfs:label>
        <rdfs:label>users</rdfs:label>
</rdf:Description>

<rdf:Description rdf:about="http://sec.cs.kent.ac.uk/permis/policy#All_Entity">
        <rdfs:label>all entity</rdfs:label>
        <rdfs:label>All entity</rdfs:label>
</rdf:Description>

<rdf:Description rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Entity">
        <rdfs:label>entity</rdfs:label>
        <rdfs:label>entities</rdfs:label>
</rdf:Description>

<rdf:Description rdf:about="http://sec.cs.kent.ac.uk/permis/policy#My_Policy">
        <rdfs:label>my policy</rdfs:label>
        <rdfs:label>My policy</rdfs:label>
</rdf:Description>

<rdf:Description rdf:about="http://sec.cs.kent.ac.uk/permis/policy#All_Policy">
        <rdfs:label>all policy</rdfs:label>
        <rdfs:label>All policy</rdfs:label>
</rdf:Description>
```

```
<rdf:Description rdf:about="http://sec.cs.kent.ac.uk/permis/policy#Policy">
        <rdfs:label>policy</rdfs:label>
        <rdfs:label>policies</rdfs:label>
</rdf:Description>

</rdf:RDF>
```